سی شارپ به زبان ساده

مؤلف : یونس ابراهیمی

سر شناسه : ابراهیمی، یونس - ۱۳۶۰

عنوان و نام پدید آورنده : سی‌شارپ به زبان ساده / مؤلف: یونس ابراهیمی

مشخصات نشر : نبض دانش، تهران - ۱۳۹۵

مشخصات ظاهری : ۷۵۴ صفحه مصور

شابک : ۹۷۸-۶۰۰-۷۷۰۳-۹۴-۶

وضعیت فهرست نویسی : فیپا

موضوع : سی‌شارپ (زبان برنامه نویسی کامپیوتر)

رده بندی دیویی : ۰۰۵/۱۳۳

رده بندی کنگره : ۱۳۹۵ ۲ الف ۹۵ س /QA۷۶/۷۳

شماره کتاب شناسی ملی : ۴۳۳۰۵۰۷

عنوان : سی‌شارپ به زبان ساده

مؤلف : یونس ابراهیمی

ناشر : نبض دانش

سال چاپ : ۱۳۹۵

نوبت چاپ : دوم

تیراژ : ۲۰۰

قیمت : ۵۹۰۰۰ تومان

تقدیم به:

همسر و پسر عزیزم

مبانی زبان سی شارپ

ویندوز فرم

دات نت فریم ورک

LINQ

ADO.NET

معماری سه لایه

مقدمه

همگام با پیشرفت فناوری‌های دیگر، زبان‌های برنامه نویسی نیز ارتقا پیدا کردند. وقتی زبان C طراحی و پیاده سازی شد، تحول بزرگی در دنیای برنامه نویسی به وجود آمد. زبان‌های متعددی از خانواده زبان C طراحی و پیاده سازی شدند که محبوب‌ترین آنها زبان سی‌شارپ است. خوشبختانه C#.NET این روزها به عنوان یکی از دروس رشته‌های کامپیوتر در دانشگاه‌های کشور تدریس می‌شود و این نشان از توانایی‌ها و اهمیت این زبان است. منابع متعددی برای معرفی و به کارگیری زبان C#.NET عرضه شده است که جای تقدیر و تشکر دارد. اما کتاب حاضر دارای ویژگی‌های بارزی از جمله، بیان ساده مطالب، ارائه مثال‌های متنوع، بررسی دقیق و موشکافانه موضوعات و سلسله مراتب آموزشی است. مثال‌هایی که در کتاب ارائه شده‌اند همگی دارای هدف خاصی هستند به طوریکه هر کدام، یک یا چند نکته زبان سی‌شارپ را به خواننده آموزش می‌دهند. در این کتاب ما به شما نحوه برنامه نویسی به زبان سی‌شارپ را به صورت تصویری آموزش می‌دهیم. سعی کنید حتماً بعد از خواندن مباحث، آنها را به صورت عملی تمرین کنید و اینکه قابلیت و مفهوم کدها را بفهمید نه آنها را حفظ کنید.

بی شک این اثر، خالی از اشکال نیست و از شما خوانندگان عزیز می‌خواهم که با نظرات و پیشنهادات خود بنده را در تکمیل و رفع نواقص آن از طریق پست الکترونیکی younes.ebrahimi.1391@gmail.com یاری بفرمایید.

در پایان جا دارد از مهندسان عزیز، آقای سیاوش ابراهیمی و محمد ابراهیمی که در تکمیل دو بخش LINQ و ویندوز فرم اینجانب را کمک کردند صمیمانه تشکر کنم. امیدوارم این هدیه ناقابل را از بنده پذیرا باشند.

برای دریافت فایل‌ها و آپدیت‌های جدید این کتاب به سایت **www.w3-farsi.com** مراجعه فرمایید.

راه‌های ارتباط با نویسنده

وب سایت	: www.w3-farsi.com
لینک تلگرام	: https://telegram.me/ebrahimi_younes
ID تلگرام	: @ebrahimi_younes
پست الکترونیکی	: younes.ebrahimi.1391@gmail.com

فصل اول

مبانی زبان سی شارپ

سی‌شارپ چیست؟

سی‌شارپ (C#) یک زبان برنامه نویسی شیء گرا است که توسط شرکت مایکروسافت ساخته شده و ترکیبی از قابلیت‌های خوب C++ و Java است. اگر با این دو زبان آشنایی دارید این شانس را دارید زبان سی‌شارپ را راحت یاد بگیرید. این زبان به قدری راحت است که هم کسانی که قبلاً برنامه نویسی نکرده‌اند و هم دانش آموزان می‌توانند راحت آن را یاد بگیرند. از سی‌شارپ می‌توان برای ساخت برنامه‌های تحت ویندوز، تحت وب، وب سرویس‌ها، برنامه‌های موبایل و بازی‌ها استفاده کرد. می‌توان به جای واژه ویژوال سی‌شارپ از کلمه سی‌شارپ استفاده کرد، اما ویژوال سی‌شارپ به معنای استفاده همزمان از سی‌شارپ و محیط گرافیکی ویژوال استودیو می‌باشد. این زبان برنامه نویسی تنها زبانی است که مخصوصاً برای دات نت فریم ورک طراحی شده است.

سی‌شارپ از کتابخانه کلاس دات نت که شامل مجموعه بزرگی از اجزاء از قبل ساخته شده است، استفاده می‌کند. این اجزا به ساخت هر چه سریع‌تر برنامه‌ها کمک می‌کنند. سی‌شارپ یک برنامه بسیار قدرتمند و شیء گرا است و با آن می‌توان برنامه‌هایی با قابلیت مدیریت بیشتر و درک آسان ایجاد کرد. ساختار این زبان نسبت به زبان‌های دیگر بسیار آسان و قابل فهم است. برای اجرای یک برنامه سی‌شارپ ابتدا باید دات نت فریم ورک نصب شود. سی‌شارپ یکی از زبان‌هایی است که از تکنولوژی‌های دیگر دات نت مانند ASP.NET، Silverlight و XNA پشتیبانی می‌کند. همچنین یک محیط توسعه یکپارچه دارد که آن نیز به نوبه خود دارای ابزارهای مفیدی است که به شما در کدنویسی در سی‌شارپ کمک می‌کند.

با ظهور C# 7.0 قابلیت های جدیدی به این زبان اضافه شد که به شما امکان می دهند که برنامه هایی بهینه تر و پربار تر با کدنویسی کمتر بنویسید. حال که اسم نسخه ۷ سی شارپ به میان آمد بهتر است که با نسخه های مختلف این زبان از ابتدا تاکنون که در جدول زیر آمده است آشنا شوید:

نسخه سی‌شارپ	نسخه .NET Framework	نسخه Visual Studio	تاریخ ارائه
C# 1.0	.NET Framework 1.0	Visual Studio .NET 2002	January 2002
C# 1.1	.NET Framework 1.1	Visual Studio .NET 2003	April 2003
C# 2.0	.NET Framework 2.0	Visual Studio 2005	November 2005
C# 3.0	.NET Framework 3.0\3.5	Visual Studio 2008	November 2007
C# 4.0	.NET Framework 4.0	Visual Studio 2010	April 2010
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	August 2012
C# 6.0	.NET Framework 4.6	Visual Studio 2015	July 2015
C# 7.0	.NET Framework 4.6.2	Visual Studio 2017	March 2017

دلیل پیدایش این زبان بر طبق دانشنامه Wikipedia بدین شرح است که:

در سال ۱۹۹۹، شرکت Sun Microsystems اجازه استفاده از زبان برنامه‌نویسی JAVA را در اختیار Microsoft قرار داد تا در سیستم عامل خود از آن استفاده کند. جاوا در اصل به هیچ پلت فرم یا سیستم عاملی وابسته نبود، ولی مایکروسافت برخی از مفاد قرار داد را زیر پا گذاشت و قابلیت مستقل از سیستم عامل بودن جاوا را از آن برداشت. شرکت Sun Microsystems پرونده‌ای علیه مایکروسافت درست کرد و مایکروسافت مجبور شد تا زبان شیءگرای جدیدی با کامپایلر جدید که به C++ شبیه بود را درست کند. آندرس هلزبرگ (Anders Hejlsberg) سرپرستی و مدیریت این پروژه را بر عهده گرفت و گروهی را برای طراحی زبانی جدید تشکیل داد و نام آن را Cool گذاشت. مایکروسافت در نظر داشت اسم این زبان را تا آخر Cool قرار دهد، ولی به دلیل مناسب نبودن برای اهداف تجاری این کار را نکرد. در ارائه و معرفی رسمی چارچوب دات‌نت در سال ۲۰۰۰ این زبان به سی‌شارپ تغییر نام یافت.

برای آشنایی بیشتر با این زبان به لینک زیر مراجعه کنید:

```
http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx
```

سی‌شارپ به طور دائم توسط مایکروسافت به روز شده و ویژگیهای جدیدی به آن اضافه می‌شود و یکی از بهترین زبان‌های برنامه نویسی دات نت است.

دات نت فریم ورک (.NET Framework) چیست؟

.NET Framework یک چارچوب است که توسط شرکت مایکروسافت برای توسعه انواع نرم افزارها علی الخصوص ویندوز طراحی شد. .NET Framework همچنین می‌تواند برای توسعه نرم افزارهای تحت وب مورد استفاده قرار بگیرد. تاکنون چندین نسخه از .NET Framework انتشار یافته که هر بار قابلیت‌های جدیدی به آن اضافه شده است.

.NET Framework شامل کتابخانه کلاس محیط کاری (FCL) که در بر گیرنده کلاس‌ها، ساختارها، داده‌های شمارشی و.... می‌باشد. مهم‌ترین قسمت .NET Framework زبان مشترک زمان اجرا (CLR) است که محیطی را فراهم می‌آورد که برنامه‌ها در آن اجرا شوند. این چارچوب ما را قادر می‌سازد که برنامه‌هایی که تحت آن نوشته شده‌اند اعم از C#.Net، Visual Basic.Net و C++ را بهتر درک کنیم. کدهایی که تحت CLR و دات‌نت اجرا می‌شوند، کدهای مدیریت شده نامیده می‌شوند، چون CLR جنبه‌های مختلف نرم‌افزار را در زمان اجرا مدیریت می‌کند. در زمان کامپایل کدها به زبان مشترک میانی (CIL) که نزدیک و تقریباً شبیه به زبان اسمبلی است ترجمه می‌شوند. ما باید کدهایمان را به این زبان ترجمه کنیم، چون فقط این زبان برای دات‌نت قابل فهم است. برای مثال کدهای C# و Visual Basic.Net هر دو به زبان مشترک میانی (CIL) ترجمه می‌شوند. به همین دلیل است که برنامه‌های مختلف در دات‌نت که با زبان‌های متفاوتی نوشته شده‌اند می‌توانند با هم ارتباط برقرار کنند. اگر یک زبان سازگار با دات‌نت می‌خواهید باید یک کامپایلر ایجاد کنید که کدهای شما را به زبان میانی ترجمه کند. کدهای ترجمه شده توسط CIL در یک فایل اسمبلی مانند exe. یا dll. ذخیره می‌شوند. کدهای ترجمه شده به زبان میانی به کامپایلر فقط در زمان (JIT) منتقل می‌شوند. این کامپایلر

در لحظه فقط کدهایی را که برنامه در آن زمان نیاز دارد به زبان ماشین ترجمه می‌کند. در زیر نحوه تبدیل کدهای سی‌شارپ به یک برنامه اجرایی به طور خلاصه آمده است:

- برنامه نویس برنامه خود را با یک زبان دات‌نت مانند سی‌شارپ می‌نویسد.
- کدهای سی‌شارپ به کدهای معادل آن در زبان میانی تبدیل می‌شوند.
- کدهای زبان میانی در یک فایل اسمبلی ذخیره می‌شوند.
- وقتی کدها اجرا می‌شوند کامپایلر JIT کدهای زبان میانی را در لحظه به کدهایی که برای کامپیوتر قابل خواندن باشند تبدیل می‌کند.

دات نت ویژگی دیگری به نام سیستم نوع مشترک (CTS) نیز دارد که بخشی از CLR است و نقشه‌ای برای معادل سازی انواع داده‌ها در دات‌نت می‌باشد. با CTS نوع int در سی‌شارپ و نوع Integer در ویژوال بیسیک یکسان هستند، چون هر دو از نوع System.Int32 مشتق می‌شوند. پاک کردن خانه‌های بلا استفاده حافظه در یک فایل (Garbage collection) یکی دیگر از ویژگی‌های دات‌نت فریم ورک است. هنگامی که از منابعی، زیاد استفاده نشود دات‌نت فریم ورک حافظه استفاده شده توسط برنامه را آزاد می‌کند.

ویژوال استودیو

ویژوال استودیو محیط توسعه یکپارچه ای است، که دارای ابزارهایی برای کمک به شما برای توسعه برنامه های سی شارپ و دات نت می باشد.شما می توانید یک برنامه سی شارپ را با استفاده از برنامه notepad یا هر برنامه ویرایشگر متن دیگر بنویسید و با استفاده از کامپایلر سی شارپ از آن استفاده کنید، اما این کار بسیار سخت است چون اگر برنامه شما دارای خطا باشد خطایابی آن سخت می شود.

توجه کنید که کلمه ویژوال استودیو هم به ویژوال استودیو و هم به ویژوال سی شارپ اشاره دارد.توصیه می کنیم که از محیط ویژوال استودیو برای ساخت برنامه استفاده کنید چون این محیط دارای ویژگی های زیادی برای کمک به شما جهت توسعه برنامه های سی شارپ می باشد.تعداد زیادی از پردازش ها که وقت شما را هدر می دهند به صورت خودکار توسط ویژوال استودیو انجام می شوند.

یکی از این ویژگی ها اینتلی سنس (Intellisense) است که شما را در تایپ سریع کدهایتان کمک می کند. یکی دیگر از ویژگیهای اضافه شده، break point است که به شما اجازه می دهد در طول اجرای برنامه مقادیر موجود در متغیرها را چک کنید.ویژوال استودیو برنامه شما را خطایابی می کند و حتی خطاهای کوچک)مانند بزرگ یا کوچک نوشتن حروف) را برطرف می کند ،همچنین دارای ابزارهای طراحی برای ساخت یک رابط گرافیکی است که بدون ویژوال استودیو برای ساخت همچین رابط گرافیکی باید کدهای زیادی نوشت.با این برنامه های قدرتمند بازدهی شما افزایش می یابد و در وقت شما با وجود این ویژگیهای شگفت انگیز صرفه جویی می شود.

در حال حاضر آخرین نسخه ویژوال استودیو Visual Studio 2017 است. این نسخه به دو نسخه Visual Studio Professional (ارزان قیمت) و Visual Studio Enterprise (گرانقیمت) تقسیم می شود و دارای ویژگی های متفاوتی هستند. خبر خوب برای توسعه دهندگان نرم افزار این است که مایکروسافت تصمیم دارد که ویژوال استودیو را به صورت متن باز ارائه دهد. یکی از نسخه های ویژوال استودیو،

Visual Studio Community می باشد که آزاد است و می توان آن را دانلود و از آن استفاده کرد.این برنامه ویژگیهای کافی را برای شروع برنامه نویسی #C در اختیار شما قرار می دهد. این نسخه (Community) کامل نیست و خلاصه شده نسخه اصلی است. به هر حال استفاده از Visual Studio Community که جایگزین Visual Studio Express شده و به نوعی همان نسخه Visual Studio Professional است، برای انجام تمرینات این سایت کافی است.

Visual Studio Enterprise 2017 دارای محیطی کاملتر و ابزارهای بیشتری جهت عیب یابی و رسم نمودارهای مختلف است که در Visual Studio Community وجود ندارند. ویژوال استودیو فقط به سی شارپ خلاصه نمی شود و دارای زبانهای برنامه نویسی دیگری از جمله ویژوال بیسیک نیز می باشد.رابط کاربری سی شارپ و ویژوال استودیو بسیار شبیه هم است و ما در این کتاب بیشتر تمرینات را با استفاده از سی شارپ انجام می دهیم.

دانلود و نصب ویژوال استودیو

در این درس می‌خواهیم نحوه دانلود و نصب نرم افزار Visual Studio Community 2017 را آموزش دهیم. در جدول زیر لیست نرم افزارها و سخت افزارهای لازم جهت نصب ویژوال استودیو ۲۰۱۷ آمده است:

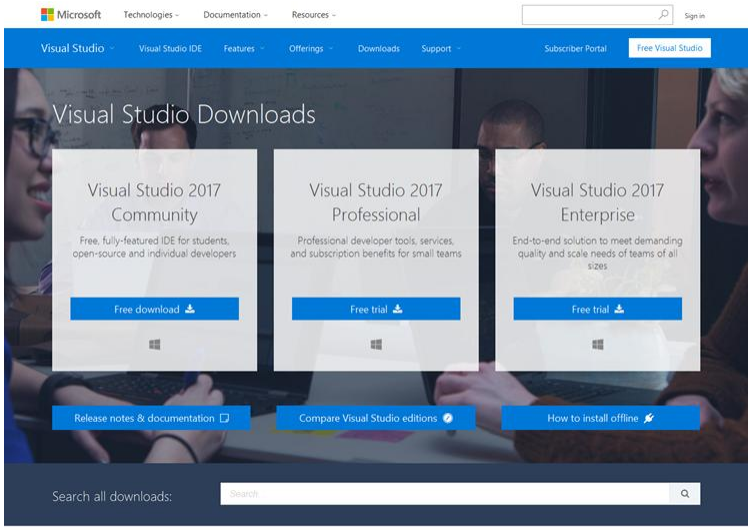
سیستم عامل	سخت افزار
Windows 10	1.6 GHz or faster processor
Windows 8.1	1 GB of RAM (1.5 GB if running on a virtual machine)
Windows 8	4 GB of available hard disk space
Windows 7 Service Pack 1	5400 RPM hard disk drive
Windows Server 2012 R2	DirectX 9-capable video card that runs at 1024 x 768 or higher display resolution
Windows Server 2012	
Windows Server 2008 R2 SP1	

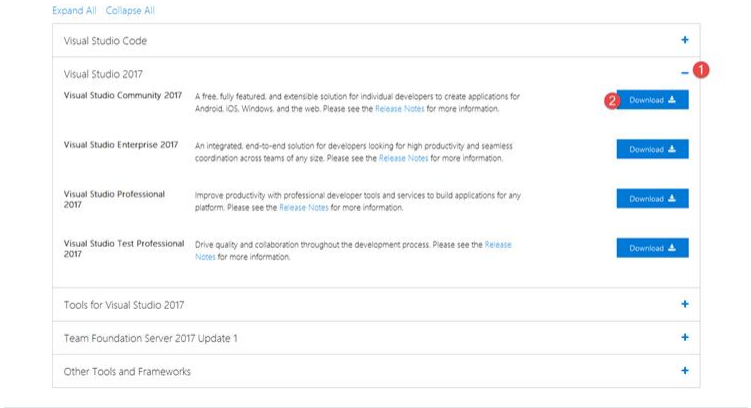
دانلود Visual Studio Community 2017

Visual Studio Community 2017 به صورت آزاد در دسترس است و می‌توانید آن را از لینک زیر دانلود کنید:

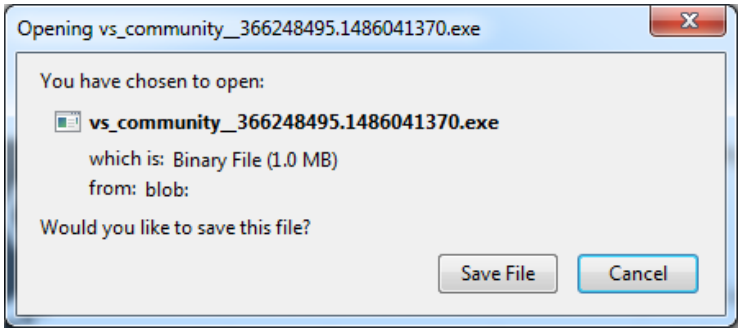
```
https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx
```

با کلیک بر روی لینک بالا صفحه ای به صورت زیر ظاهر می شود که در داخل این صفحه می توان با کلیک بر روی Visual Studio Community 2017 آن را دانلود کرد:

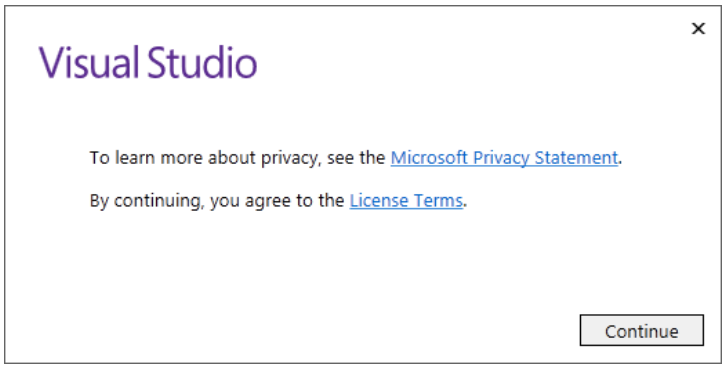


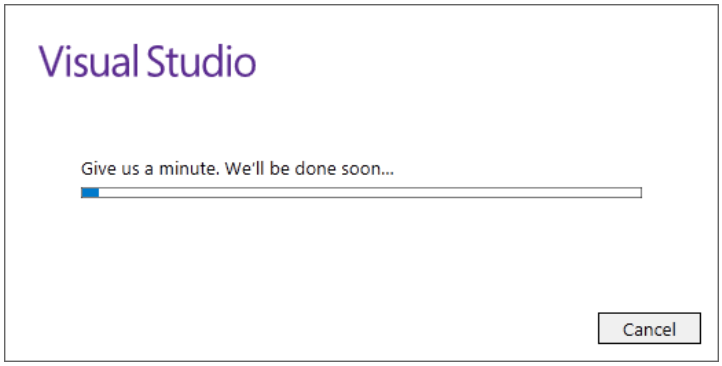


بعد از کلیک بر روی گزینه `Download` یک صفحه به صورت زیر باز می شود و از شما می خواهد که فایلی با نام `vs_community.exe` را ذخیره کنید :

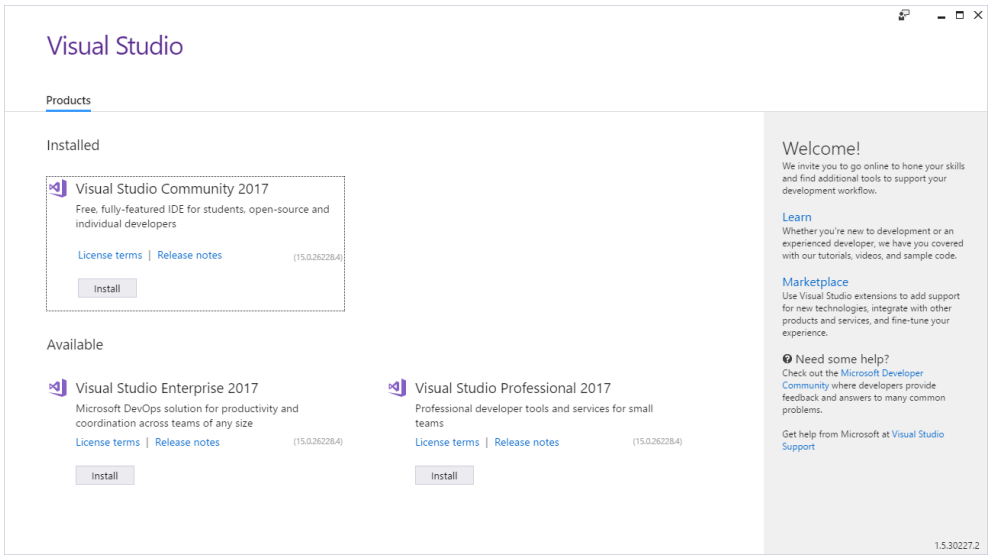


با ذخیره و اجرای این فایل مراحل نصب `Visual Studio Community 2017` آغاز می شود (`Visual Studio Community 2017` حدود ۵ گیگابایت حجم دارد و برای دانلود آن به یک اینترنت پر سرعت دارید):

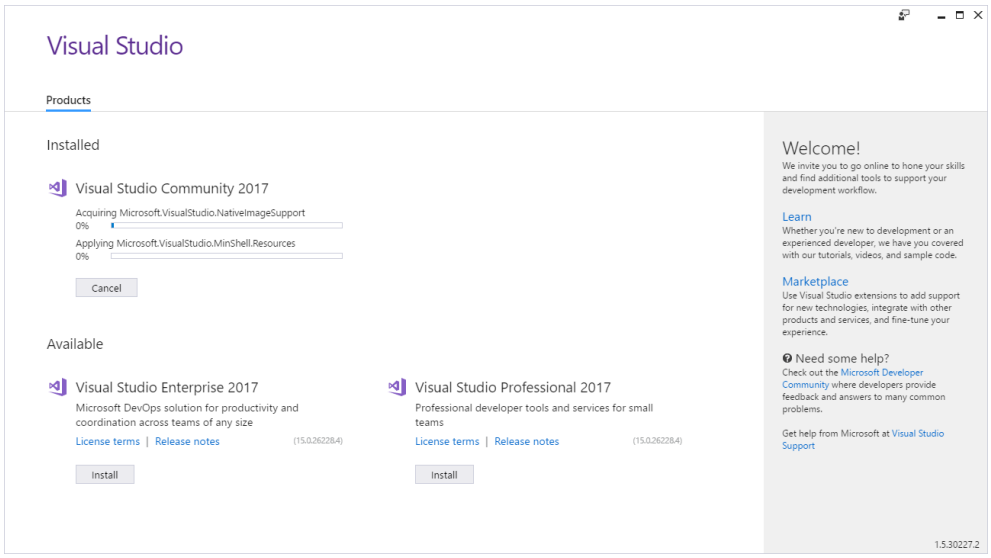




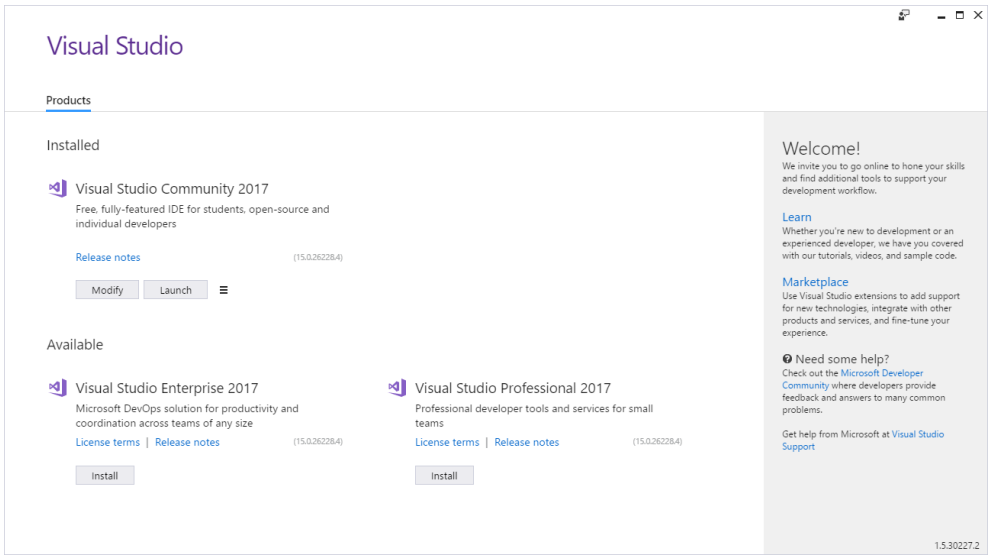
بعد از گذراندن دو صفحه بالا صفحه ای به صورت زیر باز می شود که در آن نسخه های مختلف ویژوال استودیو به شما نمایش داده می شود. بر روی گزینه `Install` روبروی `Visual Studio Community` کلیک کنید:



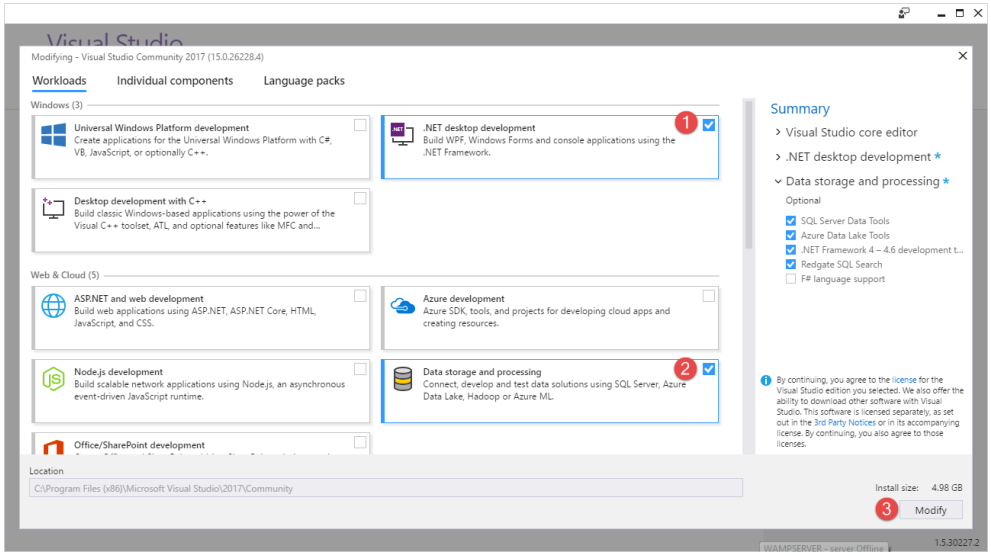
بعد از کلیک بر روی دکمه `Install` مرحله نصب شروع می شود:



بعد از اتمام مرحله بالا صفحه ای به صورت زیر باز می شود :



در صفحه بالا بر روی گزینه Modify کلیک کنید و گزینه های زیر را تیک بزنید و سپس بر روی دکمه Modify کلیک کنید :

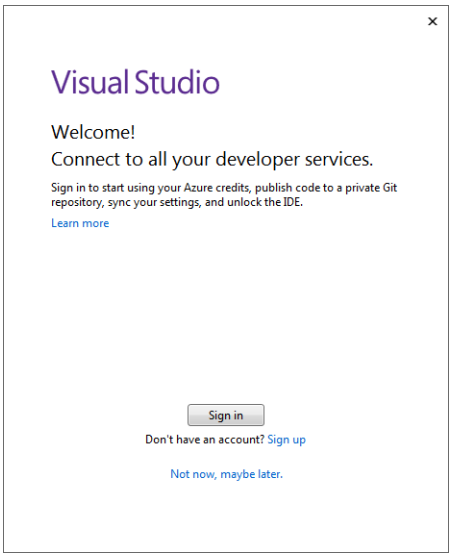


بعد از این مرحله ویژوال استودیو به صورت کامل نصب شده و شما می توانید از آن استفاده کنید.

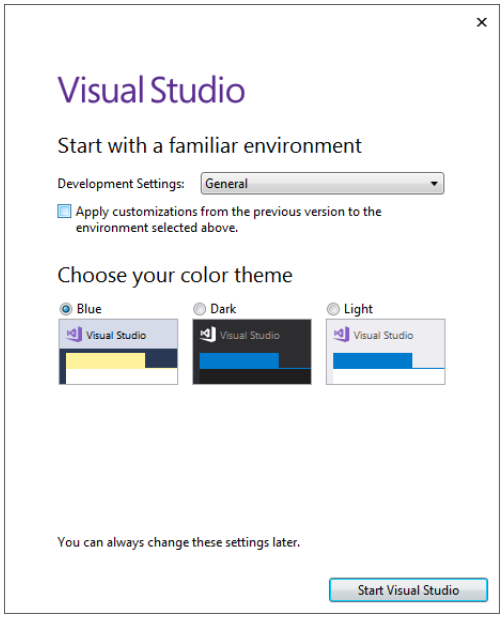
شروع کار با `Visual Studio Community`

برنامه ویژوال استودیو را اجرا کرده و منتظر بمانید تا صفحه آن بارگذاری شود:

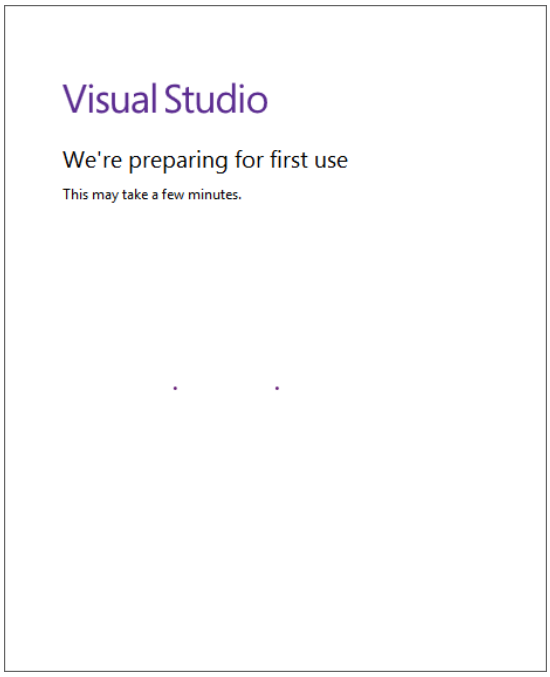
اگر دارای یک اکانت مایکروسافت باشید می توانید تغییراتی که در ویژوال استودیو می دهید را در فضای ابری ذخیره کرده و اگر آن را در کامپیوتر دیگر نصب کنید، می توانید با وارد شده به اکانت خود، تغییرات را به صورت خودکار بر روی ویژوال استودیویی که تازه نصب شده اعمال کنید. البته می توانید این مرحله را با زدن دکمه `Not now, maybe later` رد کنید:



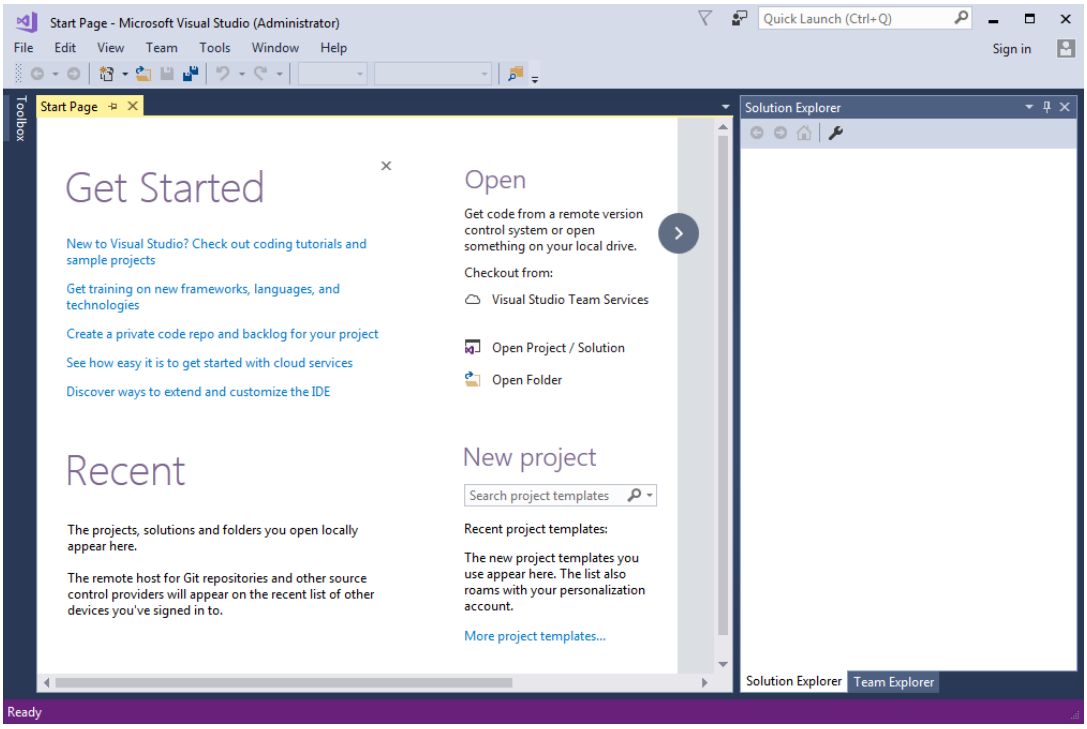
شما می توانید از بین سه ظاهر از پیش تعریف شده در ویژوال استودیو یکی را انتخاب کنید. من به صورت پیشفرض ظاهر `Blue` را انتخاب می کنم ولی شما می توانید بسته به سلیقه خود، ظاهر دیگر را انتخاب کنید:



بعد از زدن دکمه `Start Visual Studio` صفحه ای به صورت زیر ظاهر می شود:

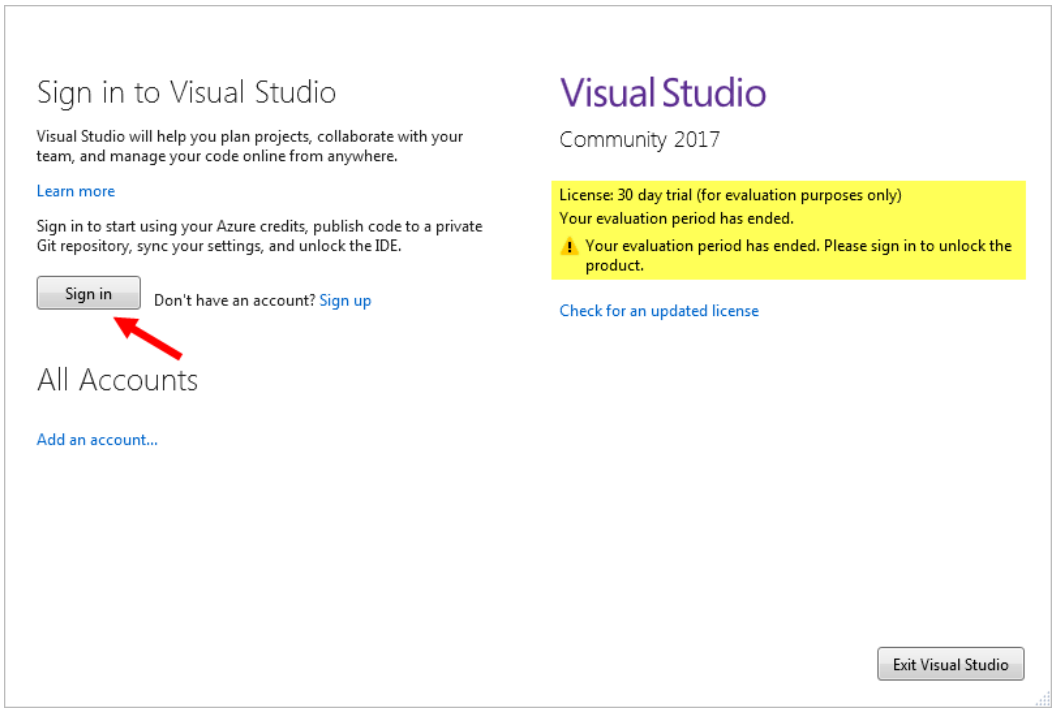


بعد از بارگذاری کامل `Visual Studio Community` صفحه اصلی برنامه به صورت زیر نمایش داده می شود که نشان از نصب کامل آن دارد :



قانونی کردن ویژوال استودیو

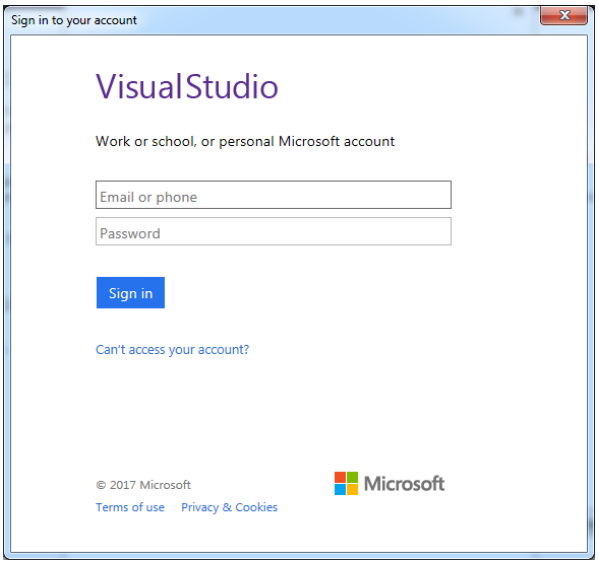
Visual Studio Community 2017 رایگان است. ولی گاهی اوقات ممکن است با پیغامی به صورت زیر مبنی بر منقضی شدن آن مواجه شوید :



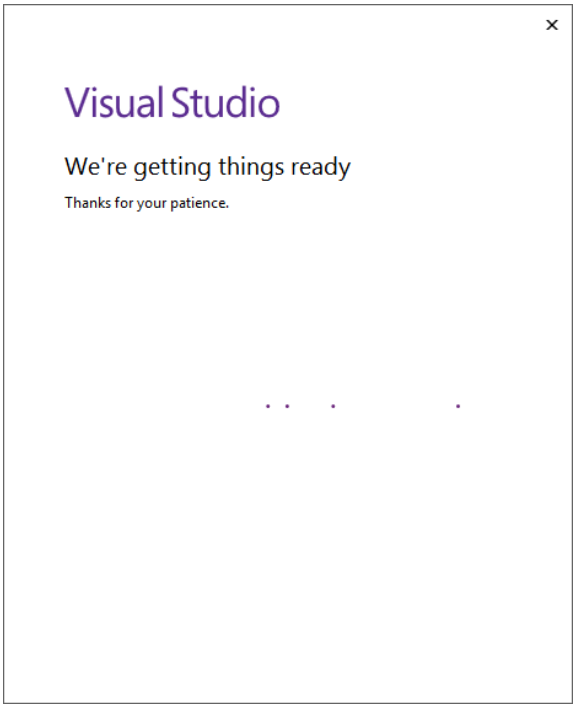
همانطور که در شکل بالا مشاهده می کنید، بر روی دکمه Signin کلیک می کنید تا وارد اکانت مایکروسافت خود شوید. اگر اکانت ندارید، می توانید از لینک زیر یک اکانت ایجاد کنید :

```
goo.gl/hMPYnE
```

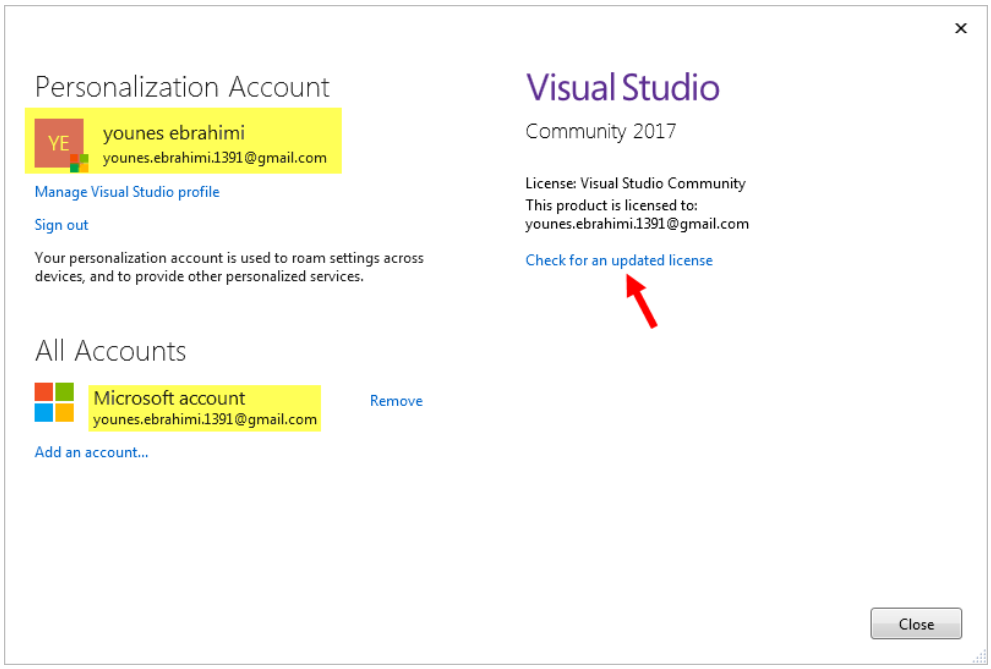
بعد از ایجاد اکانت همانطور که در شکل بالا مشاهده می کنید، بر روی گزینه Singin کلیک می کنیم. با کلیک بر روی این گزینه صفحه ای به صورت زیر ظاهر می شود که از شما مشخصات اکانتتان را می خواهد، آنها را وارد کرده و بر روی گزینه Singin کلیک کنید :



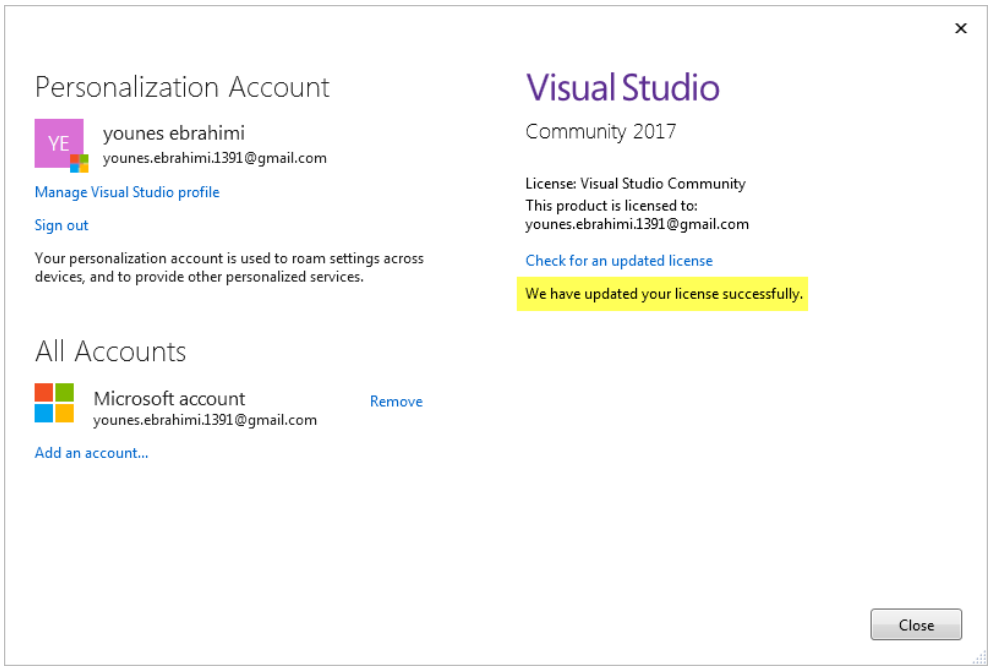
با کلیک بر روی گزینه `Signin` پنجره ای به صورت زیر نمایش داده می شود، منتظر می مانید تا پنجره بسته شود:



با بسته شدن پنجره بالا، پنجره ای به صورت زیر ظاهر می شود که مشخصات اکانت شما در آن نمایش داده می شود، که نشان از ورود موفقیت آمیز شما دارد. در این صفحه بر روی گزینه `Check an updated license` کلیک کنید:



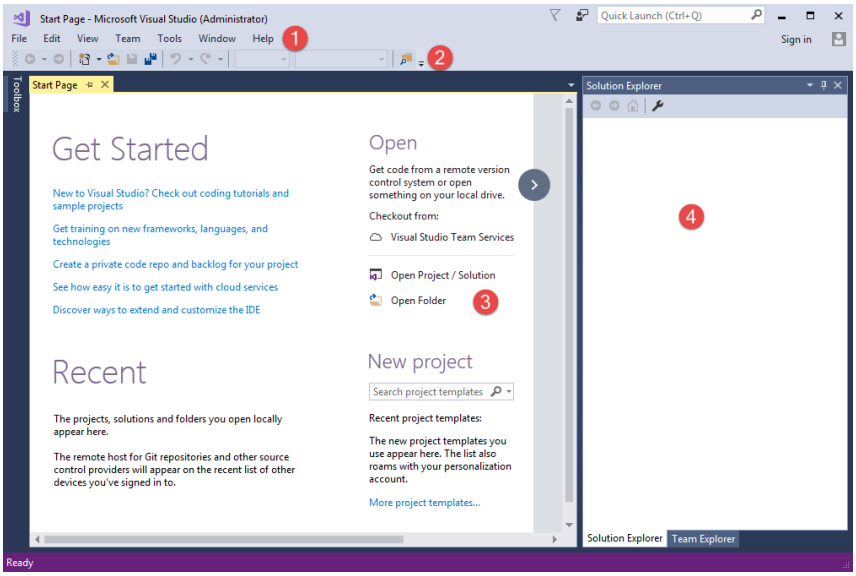
با کلیک بر روی این گزینه بعد از چند ثانیه پیغام `we have updated your license successfully` نمایش داده می شود و به این صورت ویژوال استودیو قانونی می شود :



به ویژوال استودیو خوش آمدید

در این بخش می‌خواهیم درباره قسمت‌های مختلف محیط ویژوال استودیو به شما مطالبی آموزش دهیم. لازم است که با انواع ابزارها و ویژگی‌های این محیط آشنا شوید. برنامه ویژوال استودیو را اجرا کنید :

بعد از اینکه صفحه بالا بسته شد وارد صفحه آغازین ویژوال استودیو می‌شویم :



این صفحه بر طبق عناوین خاصی طبقه‌بندی شده که در مورد آنها توضیح خواهیم داد.

منو بار (Menu Bar)

منو بار (۱)، شامل منوهای مختلفی برای ساخت، توسعه، نگهداری، خطایابی و اجرای برنامه‌ها است. با کلیک بر روی هر منو دیگر منوهای وابسته به آن ظاهر می‌شوند. به این نکته توجه کنید که منوبار دارای آیتم‌های مختلفی است که فقط در شرایط خاصی ظاهر می‌شوند. به عنوان مثال آیتم‌های منوی Project در صورتی نشان داده خواهند شد که پروژه فعال باشد. در زیر برخی از ویژگی‌های منوها آمده است:

منو	توضیح
File	شامل دستوراتی برای ساخت پروژه یا فایل، باز کردن و ذخیره پروژه‌ها و خروج از آنها می‌باشد.
Edit	شامل دستوراتی جهت ویرایش از قبیل کپی کردن، جایگزینی و پیدا کردن یک مورد خاص می‌باشد.
View	به شما اجازه می‌دهد تا پنجره‌های بیشتری باز کرده و یا به آیتم‌های toolbar آیتمی اضافه کنید.
Project	شامل دستوراتی در مورد پروژه‌ای است که شما بر روی آن کار می‌کنید.
Debug	به شما اجازه کامپایل، اشکال زدایی و اجرای برنامه را می‌دهد.
Data	شامل دستوراتی برای اتصال به دیتابیس‌ها می‌باشد.
Format	شامل دستوراتی جهت مرتب کردن اجزای گرافیکی در محیط گرافیکی برنامه می‌باشد.
Tools	شامل ابزارهای مختلف، تنظیمات و ... برای ویژوال سی‌شارپ و ویژوال استودیو می‌باشد.
Window	به شما اجازه تنظیمات ظاهری پنجره‌ها را می‌دهد.
Help	شامل اطلاعاتی در مورد برنامه ویژوال استودیو می‌باشد.

نوار ابزار (Toolbars)

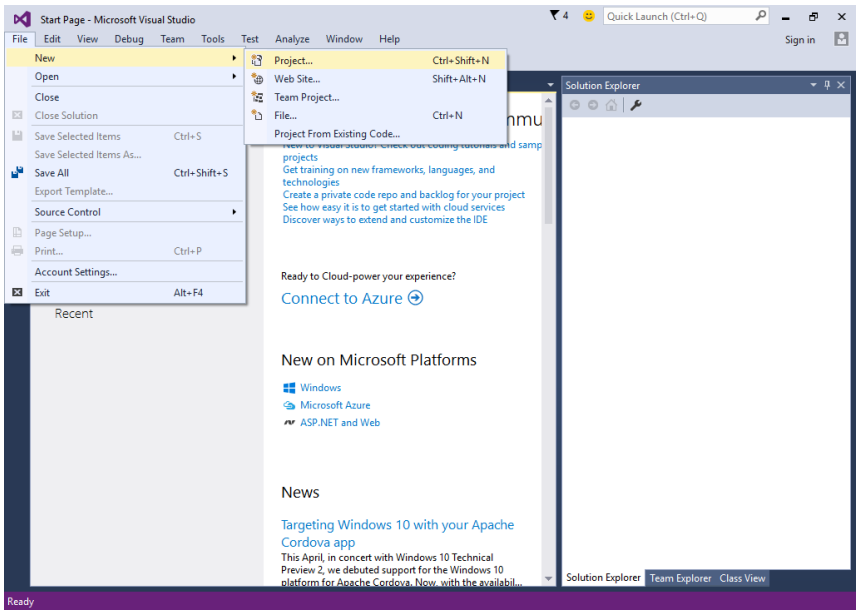
Toolbar (۲)، به طور معمول شامل همان دستوراتی است که در داخل منوها قرار دارند. Toolbar همانند یک میانبر عمل می‌کند. هر دکمه در Toolbar دارای آیکونی است که کاربرد آنرا نشان می‌دهد. اگر در مورد عملکرد هر کدام از این دکمه‌ها شک داشتید، می‌توانید با نشانگر ماوس بر روی آن مکث کوتاهی بکنید تا کاربرد آن به صورت یک پیام (tool tip) نشان داده شود. برخی از دستورات مخفی هستند و تحت شرایط خاص ظاهر می‌شوند. همچنین می‌توانید با کلیک راست بر روی منطقه خالی از Toolbar و یا از مسیر View > Toolbars دستورات بیشتری به آن اضافه کنید. برخی از دکمه‌ها دارای فلش‌های کوچکی هستند که با کلیک بر روی آنها دیگر دستورات وابسته به آنها ظاهر می‌شوند. سمت چپ هر Toolbar به شما اجازه جا به جایی آن را می‌دهد.

صفحه آغازین(Start Page)

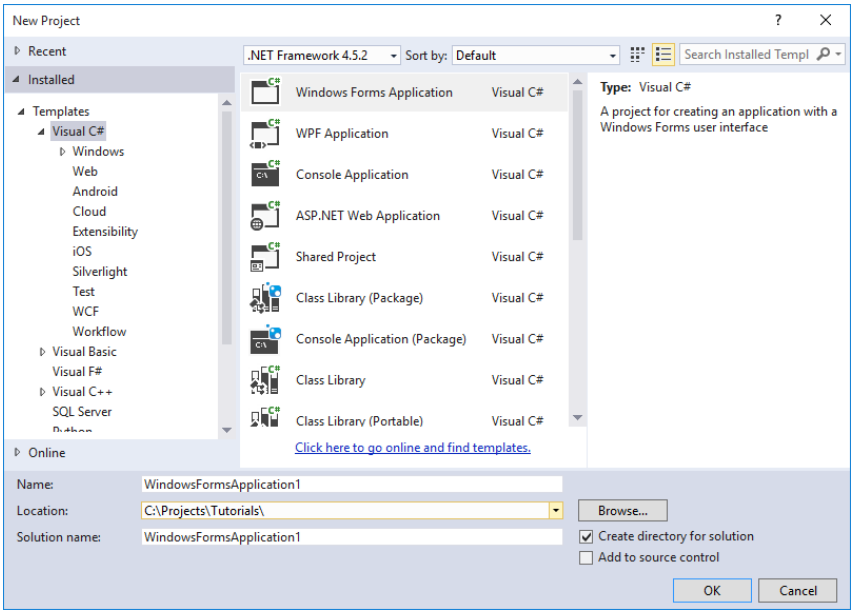
Start Page (۳)، برای ایجاد یک پروژه و باز کردن، مورد استفاده قرار می‌گیرد. همچنین اگر از قبل پروژه‌ای ایجاد کرده‌اید می‌توانید آن را در Recent Projects مشاهده و اجرا کنید. بخشهای مهم ویژوال استودیو توضیح داده شد در مورد بخش‌های بعدی در درس‌های آینده توضیحات بیشتری خواهیم داد.

گردشی در ویژوال استودیو

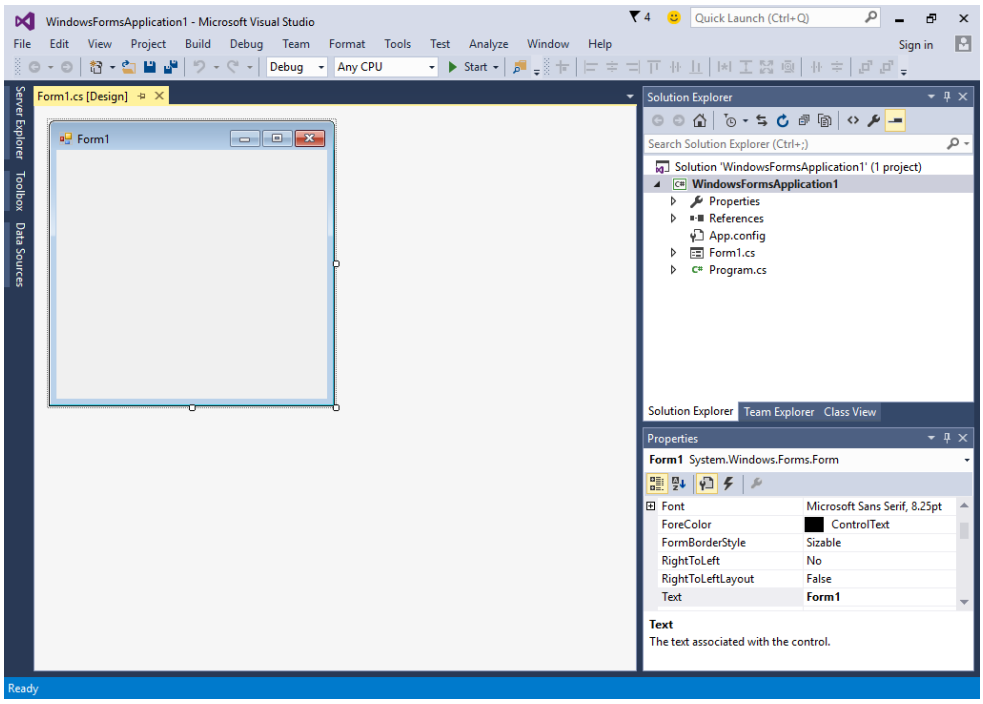
Visual Studio Community از تعداد زیادی پنجره و منو تشکیل شده است که هر کدام برای انجام کار خاصی به کار می‌روند. اجازه دهید با نفوذ بیشتر در محیط ویژوال استودیو با این قسمت‌ها آشنا شویم. از مسیر File > New Project یک پنجره فرم ایجاد کنید.



پنجره‌ای به شکل زیر نمایش داده خواهد شد.



همانطور که در شکل بالا نشان داده شده است، گزینه `Windows Forms Application` و یک اسم برای پروژه انتخاب می‌کنیم و بر روی دکمه OK کلیک می‌کنیم تا صفحه زیر نمایان شود:



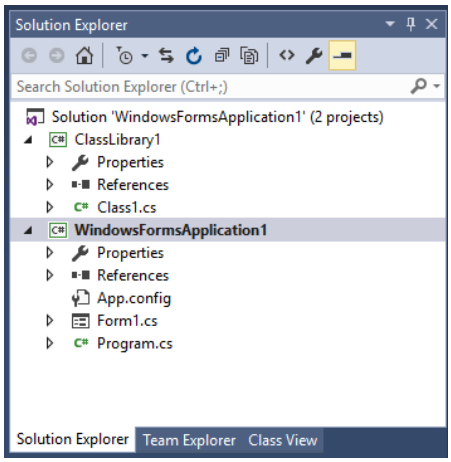
صفحه طراحی(Design)

این صفحه در حکم یک ناحیه برای طراحی فرم‌های ویندوزی شما است. فرم‌های ویندوزی رابط‌های گرافیکی بین کاربر و کامپیوتر هستند و محیط ویندوز نمونه بارزی از یک رابط گرافیکی یا GUI است. شما در این صفحه می‌توانید کنترل‌هایی مانند دکمه‌ها، برچسب‌ها و ... به فرمتان اضافه

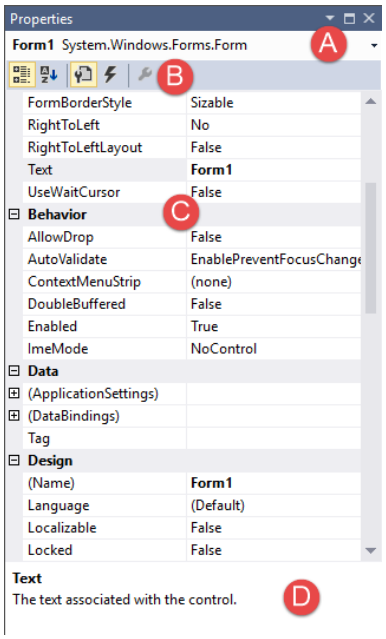
کنید. جزییات بیشتر در مورد فرم‌های ویندوزی و کنترل‌ها و برنامه‌نویسی شیء گرا در فصل فرم‌های ویندوزی آمده است. اما توصیه می‌شود ابتدا مبانی برنامه‌نویسی را مطالعه کنید.

مرورگر پروژه(Solution Explorer)

پروژه و فایل‌های مربوط به آن را نشان می‌دهد. یک Solution برنامه‌ای که توسط شما ساخته شده است را نشان می‌دهد. ممکن است این برنامه یک پروژه ساده یا یک پروژه چند بخشی باشد. اگر Solution Explorer در صفحه شما نمایش داده نمی‌شود می‌توانید از مسیر View > Other Windows > Solution Explorer و یا با کلیدهای میانبر Ctrl+Alt+L آنرا نمایان کنید. اگر چندین پروژه در حال اجرا هستند پروژه‌ای که با خط برجسته (Bold) نشان داده شده پروژه فعال می‌باشد و هنگام اجرای برنامه اجرا می‌شود. اگر بخواهید پروژه‌ای را که فعال نیست اجرا کنید، بر روی نام پروژه در Solution Explorer کلیک راست کنید و سپس گزینه Set as StartUp Project را انتخاب نمایید. Solution Explorer زیر یک Solution با ۲ پروژه را نشان می‌دهد. هر پروژه شامل فایل‌ها و فولدرهای مربوط به خود است.



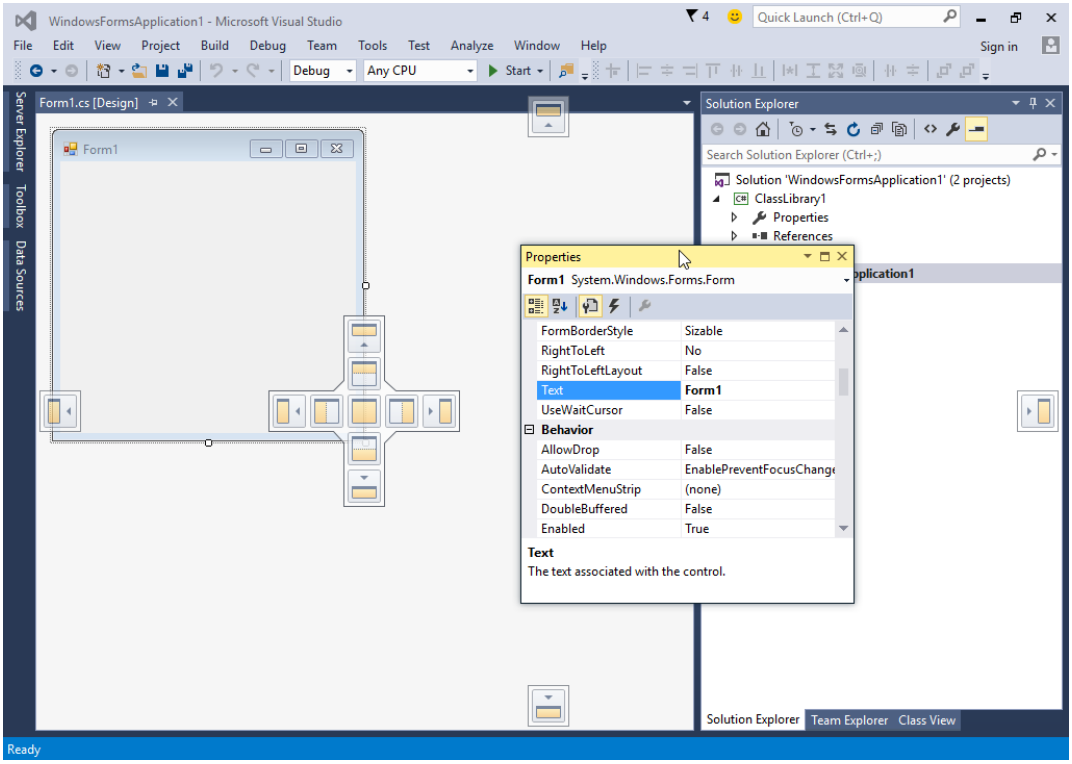
پنجره خواص(Properties)



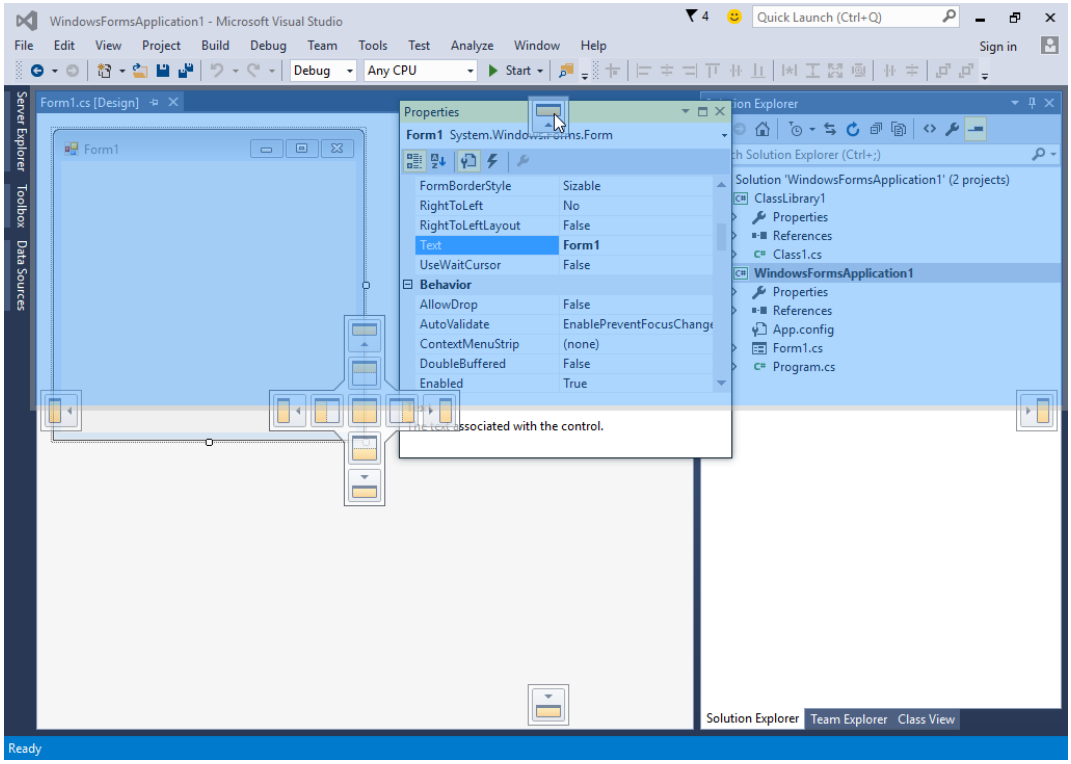
پنجره خواص (Properties)، خواص و رویدادهای مختلف هر آیتم انتخاب شده اعم از فرم، فایل، پروژه و کنترل را نشان می‌دهد. اگر این پنجره مخفی است می‌توانید از مسیر View > Other Windows > Properties Window یا کلید میانبر F4 آنرا ظاهر کنید. در مورد خواص در درس‌های آینده مفصل توضیح خواهیم داد. خاصیت‌ها، ویژگی‌ها و صفات اشیاء را نشان می‌دهند. به عنوان مثال یک ماشین دارای خواصی مانند رنگ، سرعت، اندازه و مدل است. اگر یک فرم یا کنترل را در صفحه طراحی و یا یک پروژه یا فایل را در SolutionExplorer انتخاب کنید، پنجره خواص مربوط به آنها نمایش داده خواهد شد. این پنجره همچنین دارای رویدادهای مربوط به فرم یا کنترل انتخاب شده می‌باشد. یک رویداد (event) اتفاقی است که در شرایط خاصی پیش می‌آید. مانند وقتی که بر روی دکمه (button) کلیک و یا متنی را در داخل جعبه متن (text box)اصلاح می‌کنیم. کمبو باکس (combo box) شکل بالا که با حرف A نشان داده شده است به شما اجازه می‌دهد که شیء مورد نظرتان (دکمه، فرم و...) را که می‌خواهید خواص آنرا تغییر دهید، انتخاب کنید. این کار زمانی مفید است که کنترل‌های روی فرم بسیار کوچک یا به هم نزدیک بوده و انتخاب آنها سخت باشد. در زیر کمبو باکس بالا دکمه‌های مفیدی قرار دارند (B). برخی از این دکمه‌ها در شرایط خاصی فعال می‌شوند. دکمه اول خاصیت اشیاء را بر اساس دسته‌های مختلف و دومین دکمه خواص را بر اساس حروف الفبا مرتب می‌کند که پیشنهاد می‌کنیم از این دکمه برای دسترسی سریع به خاصیت مورد نظرتان استفاده کنید. سومین دکمه هم وقتی ظاهر می‌شود که یک کنترل یا یک فرم را در محیط طراحی انتخاب کنیم. این دکمه به شما اجازه دسترسی به خواص فرم و یا کنترل انتخاب شده را می‌دهد. چهارمین دکمه (که به شکل یک جرقه نمایش داده شده) رویدادهای فرم و یا کنترل انتخاب شده را نشان می‌دهد. در پایین شکل بالا توضیحات کوتاهی در مورد خاصیت‌ها و رویدادها نشان داده می‌شود. بخش اصلی پنجره خواص (C) شامل خواص و رویدادها است. در ستون سمت چپ نام رویداد یا خاصیت و در ستون سمت راست مقدار آنها آمده است. در پایین پنجره خواص جعبه توضیحات (D) قرار دارد که توضیحاتی درباره خواص و رویدادها در آن نمایش داده می‌شود.

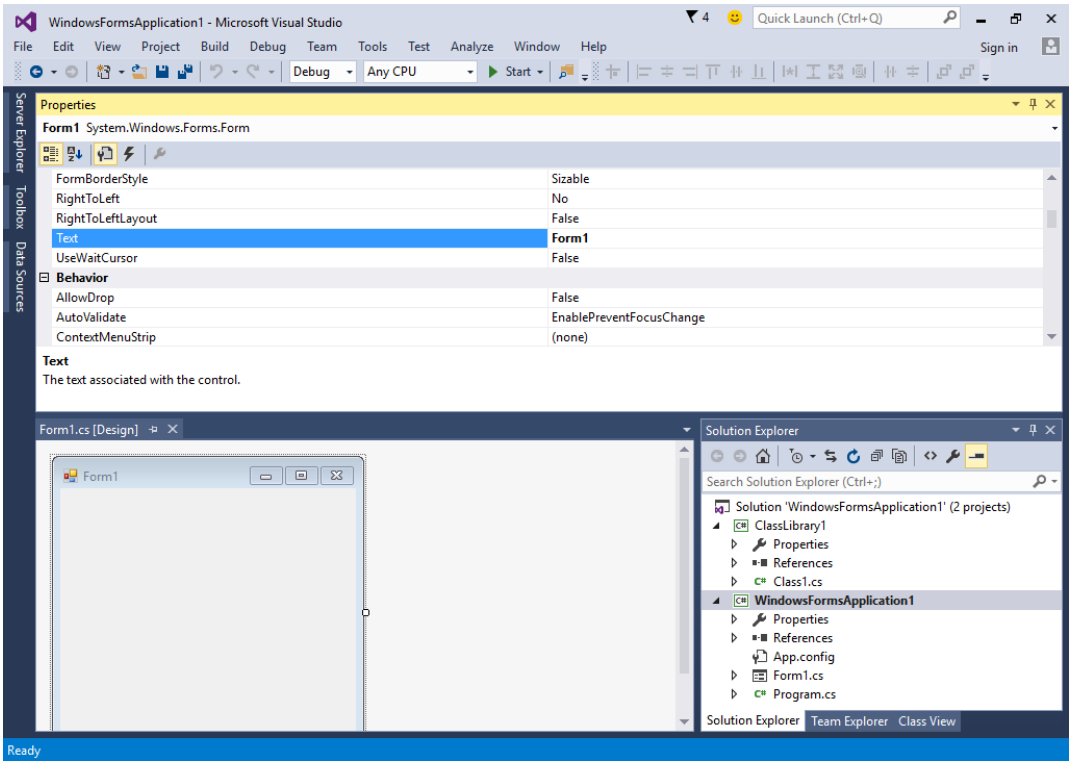
تغییر ظاهر ویژوال استودیو

اگر موقعیت پنجره‌ها و یا ظاهر برنامه ویژوال سی‌شارپ را دوست نداشته باشید، می‌توانید به دلخواه آن را تغییر دهید. برای این کار بر روی نوار عنوان (`title bar`) کلیک کرده و آنرا می‌کشید تا پنجره به شکل زیر به حالت شناور در آید:

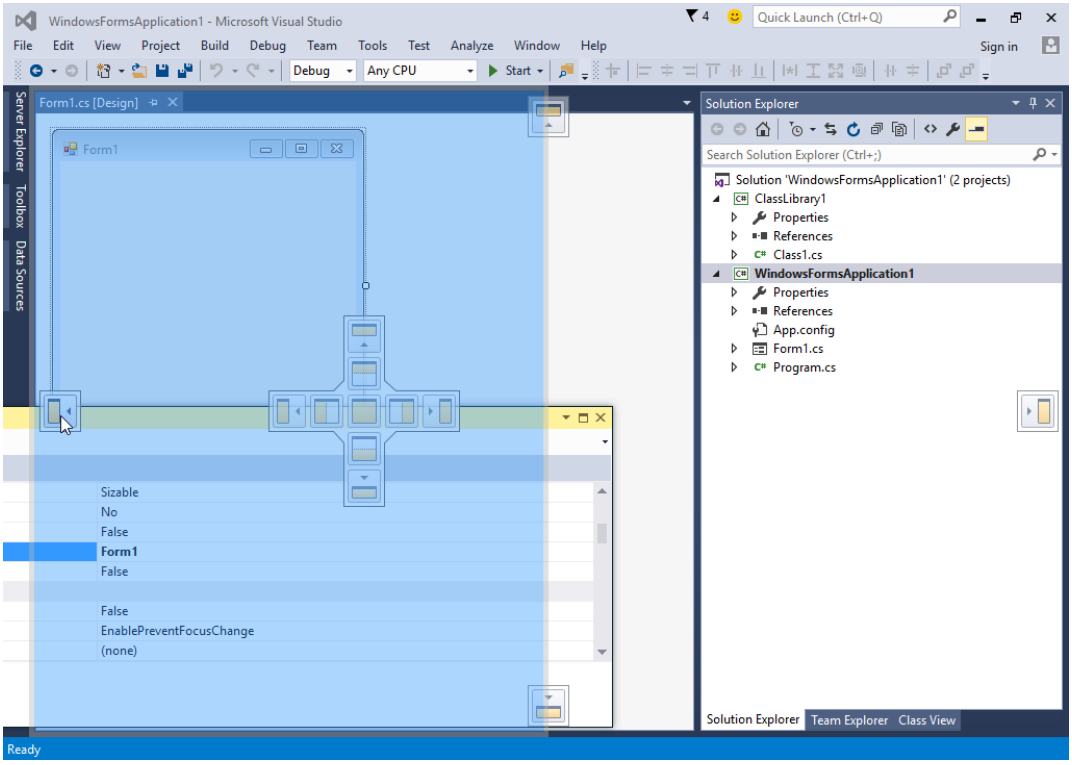


در حالی که هنوز بر روی پنجره کلیک کرده‌اید و آن را می‌کشید یک راهنما (فلشی با چهار جهت) ظاهر می‌شود و شما را در قرار دادن پنجره در محل دلخواه کمک می‌کند. به عنوان مثال شما می‌توانید پنجره را در بالاترین قسمت محیط برنامه قرار دهید. منطقه‌ای که پنجره قرار است در آنجا قرار بگیرد به رنگ آبی در می‌آید:

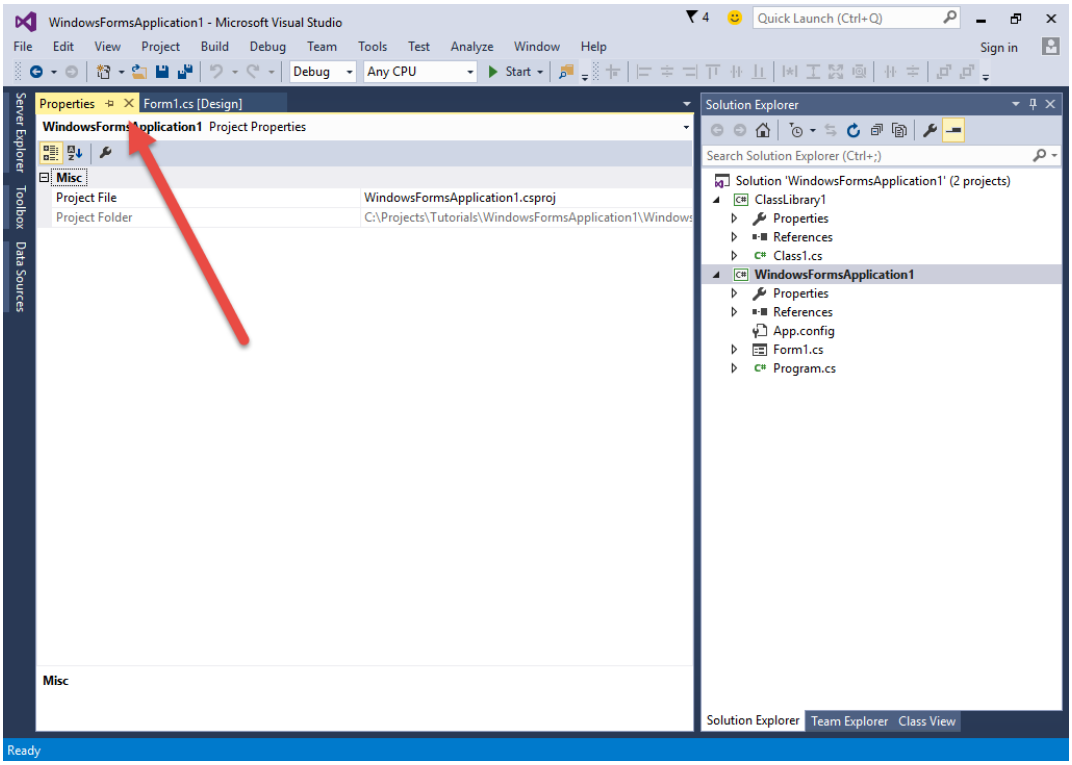

WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Form1.cs [Design]
Form1
Properties
Form1 System.Windows.Forms.Form
FormBorderStyle Sizable
RightToLeft No
RightToLeftLayout False
Text Form1
UseWaitCursor False
Behavior
AllowDrop False
AutoValidate EnablePreventFocusChange
ContextMenuStrip (none)
DoubleBuffered False
Enabled True
Solution Explorer
Solution 'WindowsFormsApplication1' (2 projects)
ClassLibrary1
Properties
References
Class1.cs
WindowsFormsApplication1
Properties
References
App.config
Form1.cs
Program.cs
Solution Explorer Team Explorer Class View
Ready


WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Properties
Form1 System.Windows.Forms.Form
FormBorderStyle Sizable
RightToLeft No
RightToLeftLayout False
Text Form1
UseWaitCursor False
Behavior
AllowDrop False
AutoValidate EnablePreventFocusChange
ContextMenuStrip (none)
Text
The text associated with the control.
Form1.cs [Design]
Form1
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WindowsFormsApplication1' (2 projects)
ClassLibrary1
Properties
References
Class1.cs
WindowsFormsApplication1
Properties
References
App.config
Form1.cs
Program.cs
Solution Explorer Team Explorer Class View
Ready

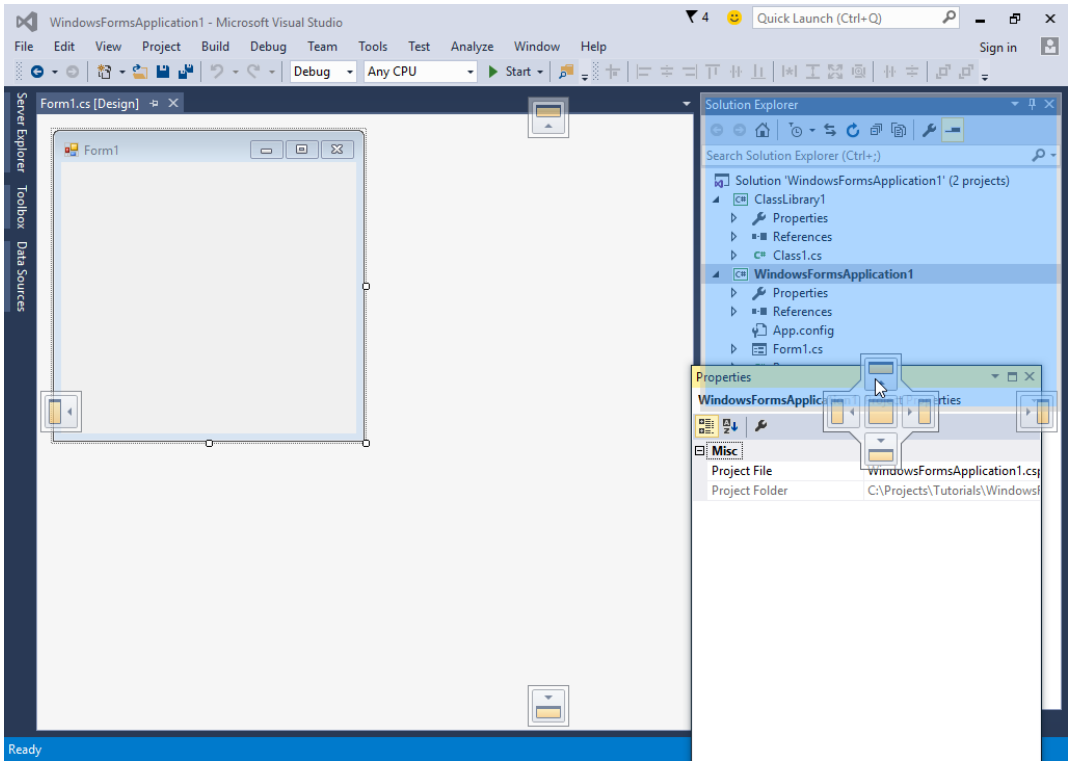
پنجره در قسمت بالای محیط قرار داده شده است. راهنمای صلیب شکل حاوی جعبه‌های مختلفی است که به شما اجازه می‌دهد پنجره انتخاب شده را در محل دلخواه محیط ویژوال استودیو قرار دهید. به عنوان مثال پنجره Properties را انتخاب و آنرا به چپ‌ترین قسمت صلیب در پنجره نمایش داده شده نزدیک و رها کنید، مشاهده می‌کنید که پنجره مذکور در سمت چپ پنجره Design View قرار می‌گیرد:



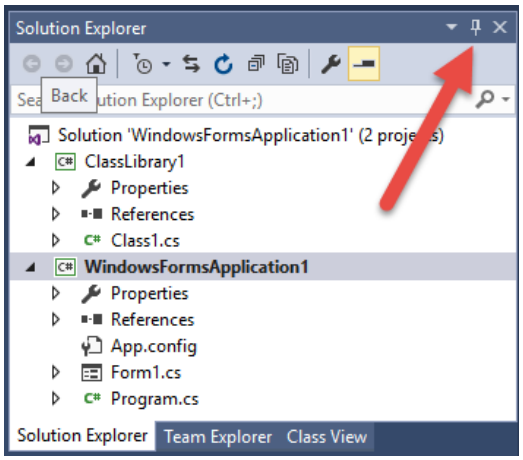
کشیدن پنجره به مرکز صلیب راهنما باعث ترکیب آن با پنجره مقصد می‌شود که در مثال بالا شما می‌توانید به عنوان یک تب به پنجره Properties دست پیدا کنید.



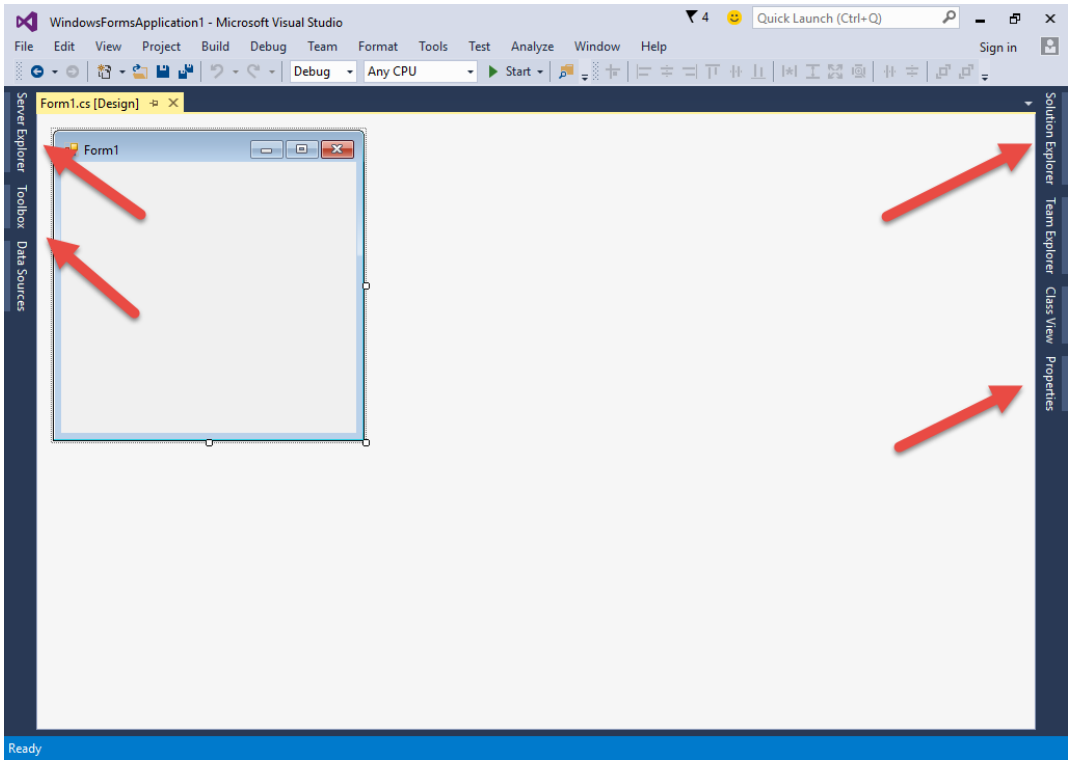
اگربه عنوان مثال پنجره Properties را روی پنجره Solution Explorer بکشید، یک صلیب راهنمای دیگر نشان داده می‌شود. با کشیدن پنجره به قسمت پایینی صلیب پنجره Properties، زیر پنجره Solution Explorer قرار خواهد گرفت.



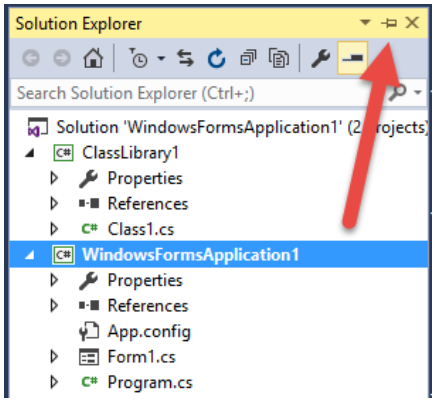
قسمتی از محیط برنامه که می‌خواهید پنجره در آنجا قرار بگیرد به رنگ آبی در می‌آید. ویژوال استودیو همچنین دارای خصوصیتی به نام autohide است که به صورت اتوماتیک پنجره‌ها را مخفی می‌کند. هر پنجره دارای یک آیکون سنجاق مانند نزدیک دکمه close می‌باشد.



بر روی این آیکون کلیک کنید تا قابلیت auto-hide فعال شود. برای دسترسی به هر یک از پنجره‌ها می‌توان با ماوس بر روی آنها توقف یا بر روی تب‌های کنار محیط ویژوال سی‌شارپ کلیک کرد.



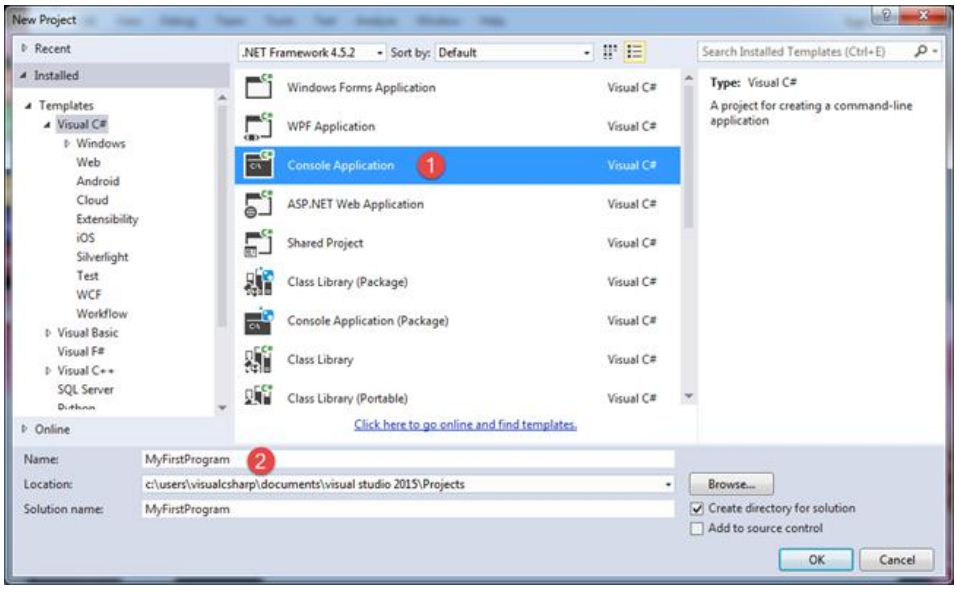
برای غیر فعال کردن این ویژگی در هر کدام از پنجره‌ها کافیست پنجره را انتخاب کرده و دوباره بر روی آیکون مورد نظر کلیک کنید.



به این نکته توجه کنید که اگر شکل آیکون افقی بود بدین معناست که ویژگی فعال و اگر شکل آن عمودی بود به معنای غیر فعال بود ویژگی auto-hide می‌باشد.

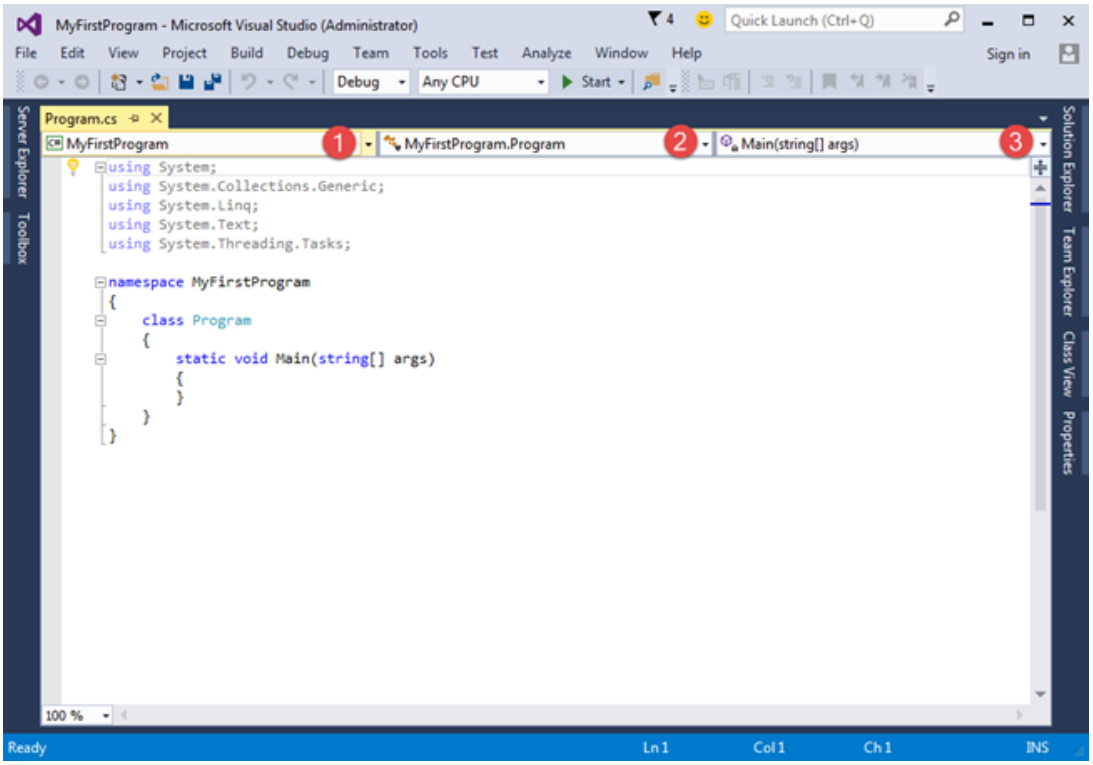
ساخت یک برنامه ساده

اجازه بدهید یک برنامه بسیار ساده به زبان سی‌شارپ بنویسیم. این برنامه یک پیغام را در محیط کنسول نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده سی‌شارپ را توضیح دهم. برنامه Visual Studio Community را اجرا کنید. از مسیر File > New یک پروژه جدید ایجاد کنید. حال با یک صفحه مواجه می‌شوید که از شما می‌خواهد نام پروژه‌تان را انتخاب و آن را ایجاد کنید (شکل زیر):



گزینه Console Application را انتخاب کرده و نام پروژه‌تان را MyFirstProgram بگذارید. یک Console Application برنامه‌ای تحت داس در محیط ویندوز است و فاقد محیط گرافیکی می‌باشد. بهتر است برنامه خود را در محیط کنسول بنویسید تا بیشتر با مفهوم برنامه‌نویسی آشنا شوید. بعد از اینکه آموزش مبانی زبان سی‌شارپ به پایان رسید، برنامه نویسی در محیط ویندوز و بخش بصری آن را آموزش خواهیم داد.

بعد از فشردن دکمه OK، برنامه Visual Studio یک solution در یک فولدر موقتی ایجاد می‌کند. یک solution مجموعه‌ای از پروژه‌هاست، اما در بیشتر تمرینات شامل یک پروژه می‌باشد. فایل solution دارای پسوند sln می‌باشد و شامل جزییاتی در مورد پروژه‌ها و فایل‌های وابسته به آن می‌باشد. پروژه جدید همچنین حاوی یک فایل با پسوند csproj. می‌باشد که آن نیز شامل جزییاتی در مورد پروژه‌ها و فایل‌های وابسته به آن می‌باشد. حال می‌خواهیم شما را با محیط کد نویسی آشنا کنیم.



محیط کدنویسی جایی است که ما کدها را در آن تایپ می‌کنیم. کدها در محیط کدنویسی به صورت رنگی تایپ می‌شوند. در نتیجه تشخیص بخشهای مختلف کد را راحت می‌کند. منوی سمت چپ (شماره ۱) شامل نام پروژه‌ای که ایجاد کرده‌اید، منوی وسط (شماره ۲) شامل لیست کلاس‌ها، ساختارها، انواع شمارشی و منوی سمت راست (شماره ۳) شامل اعضای کلاس‌ها، ساختارها، انواع شمارشی و... می‌باشد. نگران اصطلاحاتی که به کار بردیم نباشید آنها را در فصول بعد توضیح خواهم داد. همه فایل‌های دارای کد در سی‌شارپ دارای پسوند cs. هستند. در محل کد نویسی کدهایی از قبل نوشته شده که برای شروع شما آنها را پاک کنید و کدهای زیر را در محل کدنویسی بنویسید:

```
1  namespace MyFirstProgram
2  {
3      class Program
4      {
5          static void Main()
6          {
7              System.Console.WriteLine("Welcome to Visual C# Tutorials!");
8          }
9      }
10 }
```

ساختار یک برنامه در سی‌شارپ

مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در سی‌شارپ بنویسید. هدف از مثال بالا، نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه‌نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم. در خط اول فضای نام (namespace) تعریف شده است که شامل کدهای نوشته شده توسط شما است و از تداخل نام‌ها جلوگیری می‌کند. درباره فضای نام در درس‌های آینده توضیح خواهیم داد. در خط دوم آکولاد ({) نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. سی‌شارپ یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز ({) در سی‌شارپ باید دارای یک آکولاد بسته (}) نیز باشد. همه کدهای نوشته شده از خط ۲ تا خط ۱۰ یک بلوک کد یا بدنه فضای نام است. در خط ۳ یک کلاس تعریف شده است. درباره کلاس‌ها در فصل‌های آینده توضیح خواهیم داد.

در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود. بدنه کلاس شامل کدهای نوشته شده از خط ۴ تا ۹ می‌باشد. خط ۵ متد Main() یا متد اصلی نامیده می‌شود. هر متد شامل یک سری کد است که وقتی اجرا می‌شوند که متد را صدا بزنیم. درباره متد و نحوه صدا زدن آن در فصول بعدی توضیح خواهیم داد. متد Main() نقطه آغاز اجرای برنامه است. این بدان معناست که ابتدا تمام کدهای داخل متد Main() و سپس بقیه کدها اجرا می‌شود. درباره متد Main() در فصول بعدی توضیح خواهیم داد. متد Main() و سایر متدها دارای آکولاد و کدهایی در داخل آنها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. هر خط کد در سی‌شارپ به یک سمیکالن (;) ختم می‌شود. اگر سمیکالن در آخر خط فراموش شود، برنامه با خطا مواجه می‌شود. مثالی از یک خط کد در سی‌شارپ به صورت زیر است:

```
System.Console.WriteLine("Welcome to Visual C# Tutorials!");
```

این خط کد پیغام Welcome to Visual C# Tutorials! را در صفحه نمایش نشان می‌دهد. از متد WriteLine() برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است که به وسیله دابل کوتیشن (") محصور شده است، مانند : " Welcome to Visual C# Tutorials!".

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از متد WriteLine() است که در داخل کلاس Console که آن نیز به نوبه خود در داخل فضای نام MyFirstProgram قرار دارد را نشان می‌دهد. توضیحات بیشتر در درس‌های آینده آمده است. سی‌شارپ فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید. اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. یکی از خطاهای معمول در برنامه‌نویسی فراموش کردن سمیکالن در پایان هر خط کد است. به مثال زیر توجه کنید:

```
System.Console.WriteLine(
    "Welcome to Visual C# Tutorials!");
```

سی‌شارپ فضای خالی بالا را نادیده می‌گیرد و از کد بالا اشکال نمی‌گیرد. اما از کد زیر ایراد می‌گیرد:

```
System.Console.WriteLine(;
    "Welcome to Visual C# Tutorials!");
```

به سمیکالن آخر خط اول توجه کنید. برنامه با خطای نحوی مواجه می‌شود، چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سمیکالن در آخر آن قرار دهید. همیشه به یاد داشته باشید که سی‌شارپ به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در سی‌شارپ با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
system.console.writeline("Welcome to Visual C# Tutorials!");
SYSTEM.CONSOLE.WRITELINE("Welcome to Visual C# Tutorials!");
sYsTem.cONsoLe.wRItelIne("Welcome to Visual C# Tutorials!");
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

```
System.Console.WriteLine("WELCOME TO VISUAL C# TUTORIALS!");
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

```
{
   statement1;
}
```

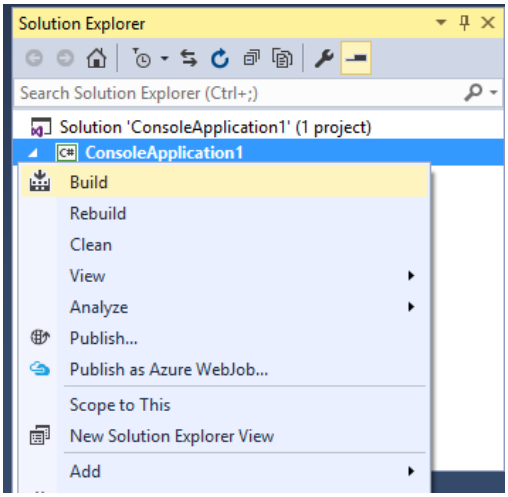
این کار باعث می‌شود که کدنویسی شما بهتر به چشم بیاید و تشخیص خطاها راحت تر باشد. یکی از ویژگی‌های مهم سی‌شارپ نشان دادن کدها به صورت تو رفتگی است. بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می‌کند و این در خوانایی برنامه بسیار مؤثر است.

ذخیره پروژه و برنامه

برای ذخیره پروژه و برنامه می‌توانید به مسیر File > Save All بروید یا از کلیدهای میانبر Ctrl+Shift+S استفاده کنید. همچنین می‌توانید از قسمت Toolbar بر روی شکل  کلیک کنید. برای ذخیره یک فایل ساده می‌توانید به مسیر File > Save (FileName) رفته یا از کلیدهای میانبر Ctrl+S استفاده کنید. همچنین می‌توانید از قسمت Toolbar بر روی شکل  کلیک کنید. برای باز کردن یک پروژه یا برنامه از منوی File گزینه Open را انتخاب و یا بر روی آیکون  در toolbar کلیک کنید. سپس به محلی که پروژه در آنجا ذخیره شده رفته و فایلی با پسوند sln یا پروژه‌ای با پسوند .csproj را باز کنید.

کامپایل برنامه

قبلاً ذکر شد که کدهای ما قبل از اینکه آنها را اجرا کنیم، ابتدا به زبان میانی ترجمه می‌شوند. برای کامپایل برنامه از منوی Debug گزینه Build Solution را انتخاب کنید یا دکمه F6 را بر روی صفحه کلید فشاردهید. این کار همه پروژه‌های داخل solution را کامپایل می‌کند. برای کامپایل یک قسمت از solution به Solution Explorer رفته و بر روی آن قسمت راست کلیک کرده و از منوی باز شده گزینه build را انتخاب کنید. مانند شکل زیر:



اجرای برنامه

وقتی ما برنامه مان را اجرا می‌کنیم سی‌شارپ به صورت اتوماتیک کدهای ما را به زبان میانی کامپایل می‌کند. دو راه برای اجرای برنامه وجود دارد :

- اجرا همراه با اشکال زدایی(Debug)
- اجرا بدون اشکال زدایی(Non-Debug)

اجرای بدون اشکال زدایی برنامه، خطاهای برنامه را نادیده می‌گیرد. با اجرای برنامه در حالت Non-Debug سریعاً برنامه اجرا می‌شود و شما با زدن یک دکمه از برنامه خارج می‌شوید. در حالت پیش فرض حالت Non-Debug مخفی است و برای استفاده از آن می‌توان از منوی Debug گزینه Start Without Debuging را انتخاب کرد یا از دکمه‌های ترکیبی Crl + F5 استفاده نمود:

```
Welcome to Visual C# Tutorials!
Press any key to continue . . .
```

به این نکته توجه کنید که پیغام ...Press any key to continue جزء خروجی به حساب نمی‌آید و فقط نشان دهنده آن است که برنامه در حالت Non-Debug اجرا شده است و شما می‌توانید با زدن یک کلید از برنامه خارج شوید. دسترسی به حالت Debug Mode آسان تر است و به صورت پیش‌فرض برنامه‌ها در این حالت اجرا می‌شوند. از این حالت برای رفع خطاها و اشکال زدایی برنامه‌ها استفاده می‌شود که در درس‌های آینده توضیح خواهیم داد.

شما همچنین می‌توانید از Break Points و قسمت Help برنامه در مواقعی که با خطا مواجه می‌شوید استفاده کنید. برای اجرای برنامه با حالت Debug Mode می‌توانید از منوی Debug گزینه Start Debugging را انتخاب کرده و یا دکمه F5 را فشار دهید. همچنین می‌توانید بر روی شکل ▶ Start در toolbar کلیک کنید. اگر از حالت Debug Mode استفاده کنید برنامه نمایش داده شده و فوراً ناپدید می‌شود. برای جلوگیری از این اتفاق شما می‌توانید از کلاس و متد System.Console.ReadKey() برای توقف برنامه و گرفتن ورودی از کاربر جهت خروج از برنامه استفاده کنید (درباره متدها در درس‌های آینده توضیح خواهیم داد).

```
namespace MyFirstProgram
```

```
{
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Welcome to Visual C# Tutorials!");
            System.Console.ReadKey();
        }
    }
}
```

حال برنامه را در حالت Debug Mode اجرا می‌کنیم. مشاهده می‌کنید که برنامه متوقف شده و از شما در خواست ورودی می‌کند، به سادگی و با زدن دکمه Enter از برنامه خارج شوید. من از حالت Non-Debug به این علت استفاده کرده‌ام تا نیازی به نوشتن کد اضافی Console.ReadKey() نباشد. از این به بعد هر جا ذکر شد که برنامه را اجرا کنید برنامه را در حالت Non-Debug اجرا کنید. وقتی به مبحث استثناءها رسیدیم از حالت Debug استفاده می‌کنیم.

وارد کردن فضای نام در برنامه

فضای نام (Namespace) در برگیرنده کدهایی است که شما در برنامه‌تان از آنها استفاده می‌کنید. در برنامه فوق ما یک فضای نام در برنامه مان با نام MyFirstProgram داریم، اما دات‌نت دارای هزاران فضای نام می‌باشد. یکی از این فضاهای نامی، فضای نام System است که شامل کدهایی است که در یک برنامه ابتدایی C# به کار می‌روند. کلاس Console که ما از آن در برنامه بالا استفاده کردیم در این فضای نام قرار دارد.

```
System.Console.WriteLine("Welcome to Visual C# Tutorials!");
System.Console.ReadKey();
```

اینکه قبل از استفاده از هر کلاس ابتدا فضای نام آن را مانند کد بالا بنویسیم کمی خسته کننده است. خوشبختانه دات‌نت به ما اجازه می‌دهد که برای جلوگیری از تکرار مکررات، فضاهای نامی را که قرار است در برنامه استفاده کنیم با استفاده از دستور using در ابتدای برنامه وارد نماییم:

```
using namespace;
```

دستور بالا نحوه وارد کردن یک فضای نام در برنامه را نشان می‌دهد. در نتیجه به جای آنکه به صورت زیر ابتدا نام فضای نام و سپس نام کلاس را بنویسیم:

```
System.Console.WriteLine("Hello World!");
```

می‌توانیم فضای نام را با دستوری که ذکر شد وارد برنامه کرده و کد بالا را به صورت خلاصه شده زیر بنویسیم:

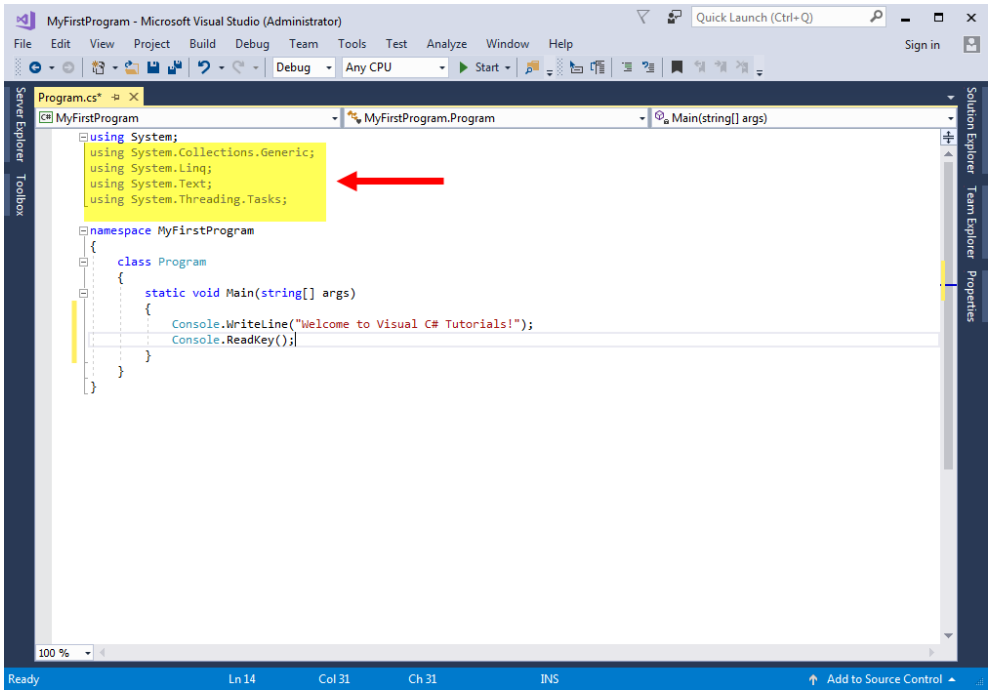
```
Console.WriteLine("Hello World!");
```

دستورات using که باعث وارد شدن فضاهای نامی به برنامه می‌شوند عموماً در ابتدای برنامه و قبل از همه کدها نوشته می‌شوند، پس برنامه‌ی این درس را می‌توان به صورت زیر نوشت:

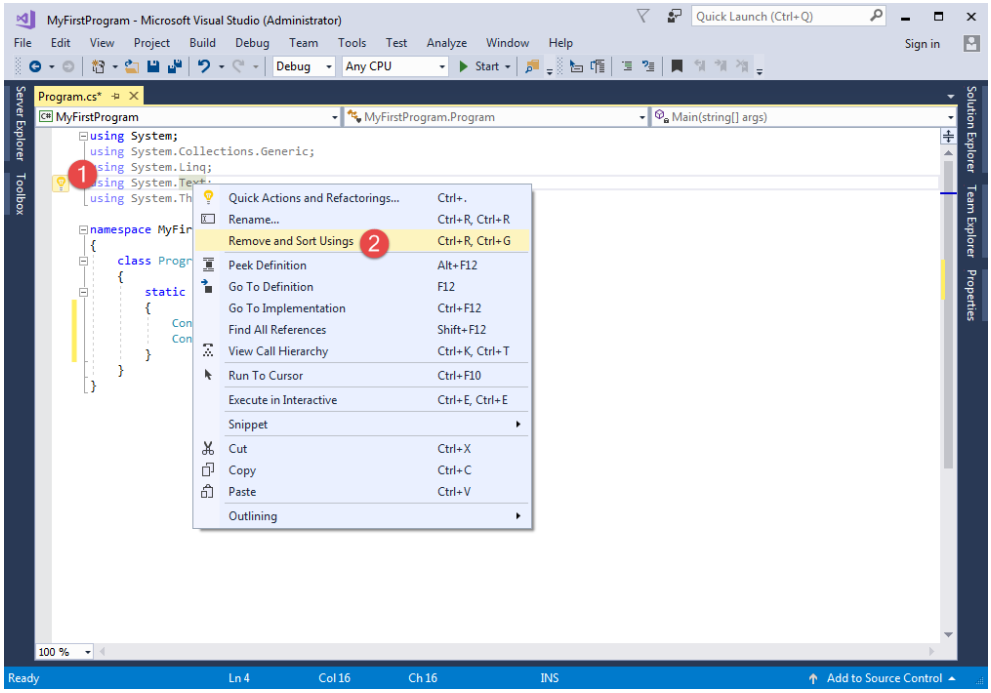
```
using System;

namespace MyFirstProgram
{
```

```
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Welcome to Visual C# Tutorials!");
            Console.ReadKey();
        }
    }
}
```

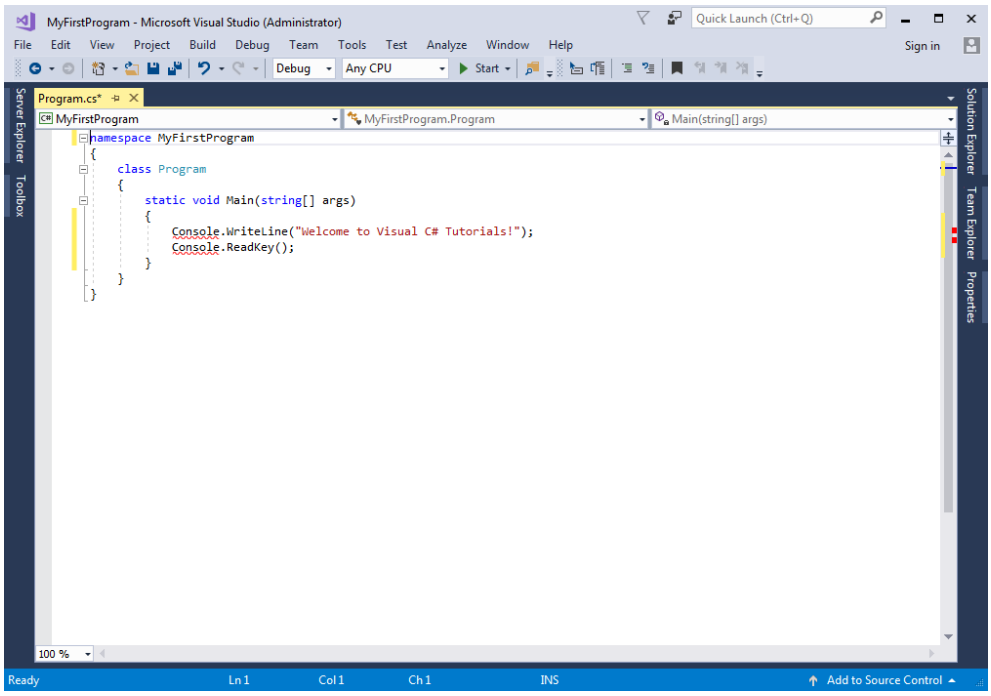
هنگامی که یک برنامه در ویژوال استودیو ایجاد می کنید، اگر وجود برخی از فضاهای نام الزامی نباشد، ویژوال استودیو ۲۰۱۷ آنها را به صورت کم رنگ نمایش می دهد، و شما می توانید بدون هیچ مشکلی این فضاهای نام را پاک کنید:



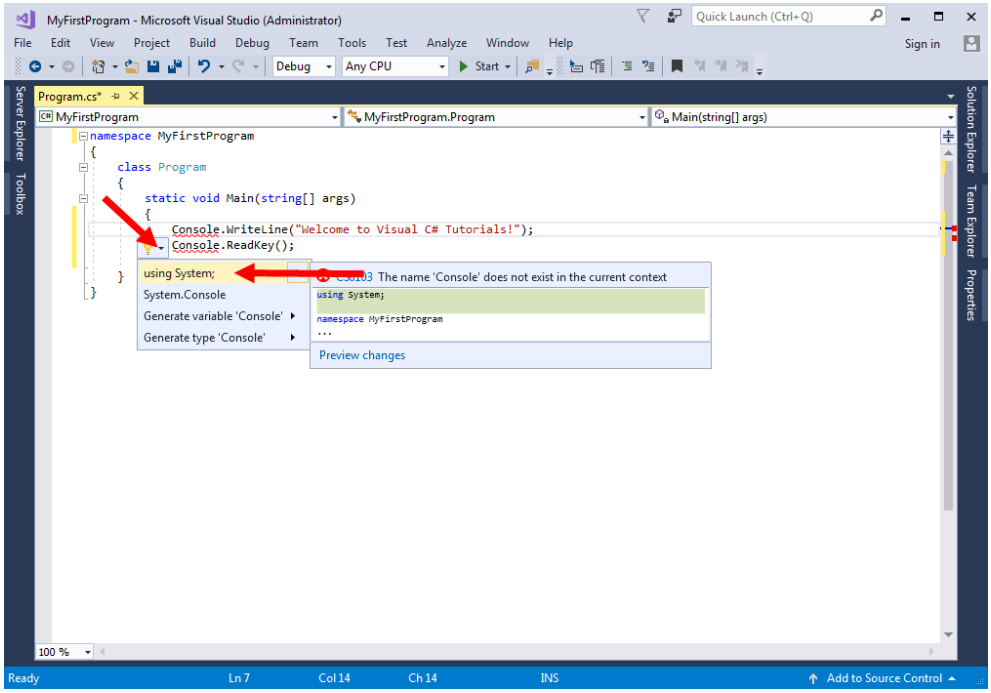
در نسخه های قبلی ویژوال استودیو هم برای پاک کردن فضاهای نام غیر قابل استفاده، ابتدا بر روی یکی از آنها راست کلیک کرده و سپس بر روی Remove and Sort Usings کلیک کنید:



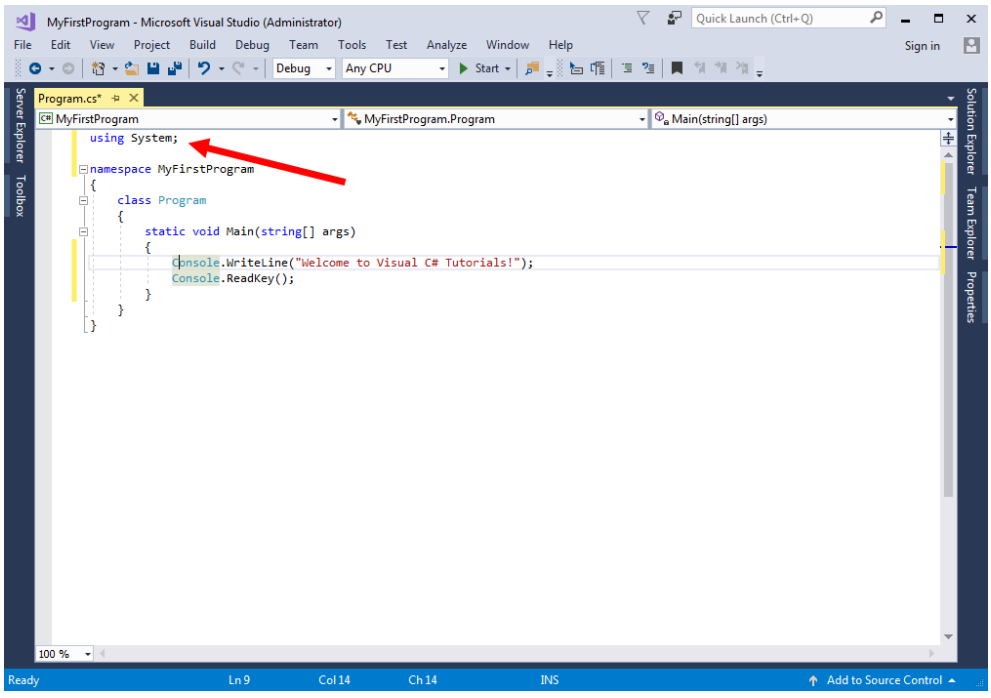
اگر از کلاسی استفاده کنید که از قبل فضای نام مربوط به آن را وارد برنامه نکرده باشید در زیر آن کلاس خط قرمز کشیده می شود :



برای رفع این مشکل، اگر از قبل نام فضای مربوطه را بلد باشید که باید آن را در قسمت فضای نام وارد کنید. در غیر اینصورت، بر روی نام کلاس با ماوس کمی مکث کنید تا یک پنجره popup ظاهر شده و آن را به شما معرفی کند :



در این صورت با کلیک بر روی آن، ویژوال استودیو به طور خودکار فضای نام را وارد برنامه می کند :



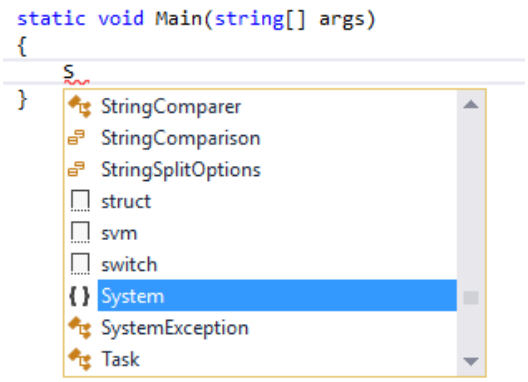
در مورد فضای نام در درس های آینده بیشتر توضیح می دهیم. حال که با خصوصیات و ساختار اولیه سی شارپ آشنا شدید در درسهای آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

استفاده از IntelliSense

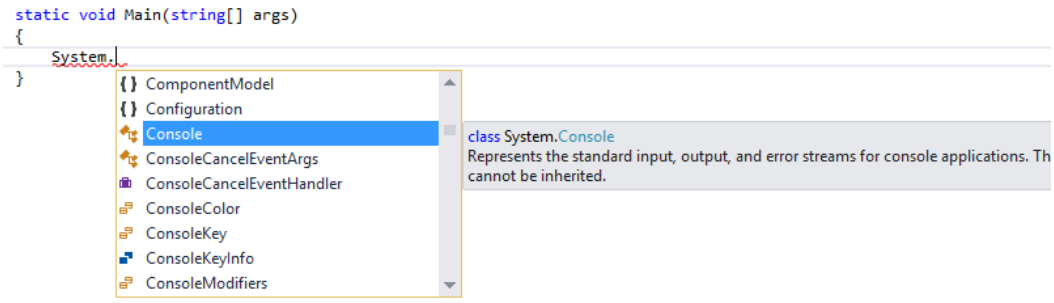
شاید یکی از ویژگی‌های مهم Visual Studio، اینتلی سنس باشد. IntelliSense ما را قادر می‌سازد که به سرعت به کلاس‌ها، متدها و...دسترسی پیدا کنیم. وقتی که شما در محیط کدنویسی حرفی را تایپ کنید IntelliSense فوراً فعال می‌شود. کد زیر را در داخل متد Main() بنویسید.

```
System.Console.WriteLine("Welcome to Visual C# Tutorials!");
```

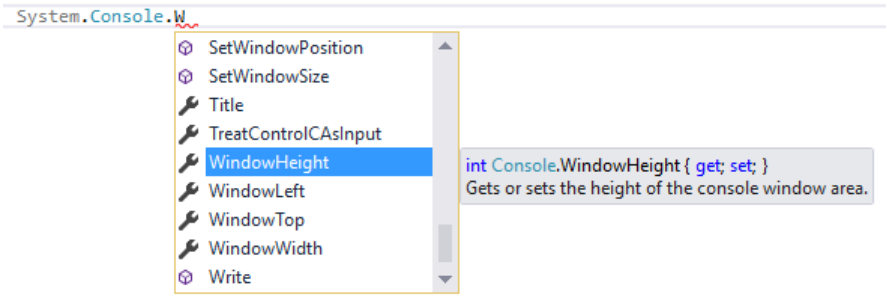
اولین حرف را تایپ کنید تا IntelliSense فعال شود.



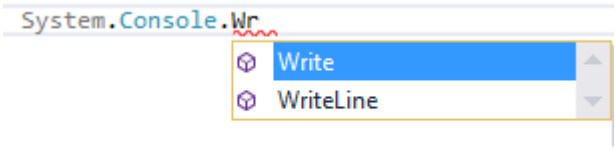
IntelliSense لیستی از کلمات به شما پیشنهاد می‌دهد که بیشترین تشابه را با نوشته شما دارند. شما می‌توانید با زدن دکمه Tab گزینه مورد نظرتان را انتخاب کنید. با تایپ نقطه (.) شما با لیست پیشنهادی دیگری مواجه می‌شوید.



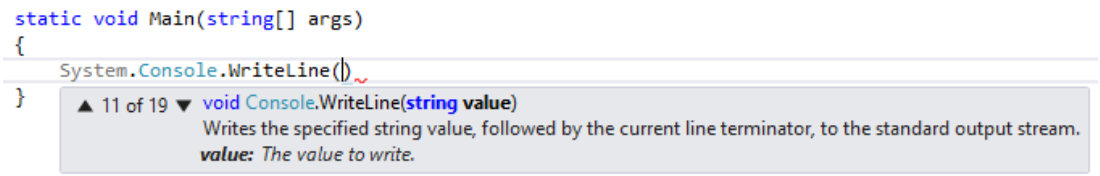
اگر بر روی گزینه‌ای که می‌خواهید انتخاب کنید لحظه‌ای مکث کنید، توضیحی در رابطه با آن مشاهده خواهید کرد، مانند شکل بالا. هر چه که به پایان کد نزدیک می‌شوید لیست پیشنهادی محدود تر می‌شود. برای مثال با تایپ حرف W، IntelliSense فقط کلماتی که با حرف W شروع می شوند، را نمایش می‌دهد.



با تایپ حرف‌های بیشتر لیست محدودتر شده و فقط دو کلمه را نشان می‌دهد.



اگر IntelliSense نتواند چیزی را که شما تایپ کرده‌اید پیدا کند، هیچ چیزی را نمایش نمی‌دهد. برای ظاهر کردن IntelliSense کافیست دکمه ترکیبی Ctrl+Space را فشار دهید. برای انتخاب یکی از متدهایی که دارای چند حالت هستند، می‌توان با استفاده از دکمه‌های مکان نما (بالا و پایین) یکی از حالت‌ها را انتخاب کرد. مثلاً متد Writeline() همانطور که در شکل زیر مشاهده می‌کنید دارای ۱۹ حالت نمایش پیغام در صفحه است.

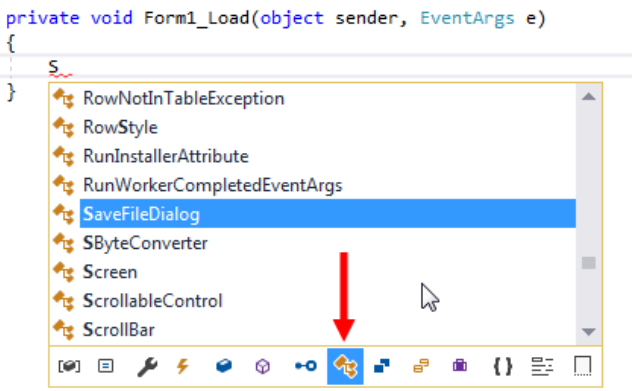


IntelliSense به طور هوشمند کدهایی را به شما پیشنهاد می‌دهد و در نتیجه زمان نوشتن کد را کاهش می‌دهد. در ویژوال استودیو هر جزء دارای یک آیکون منحصر به فرد می باشد. در زیر لیست آیکون های ویزوال استودیو آمده است:

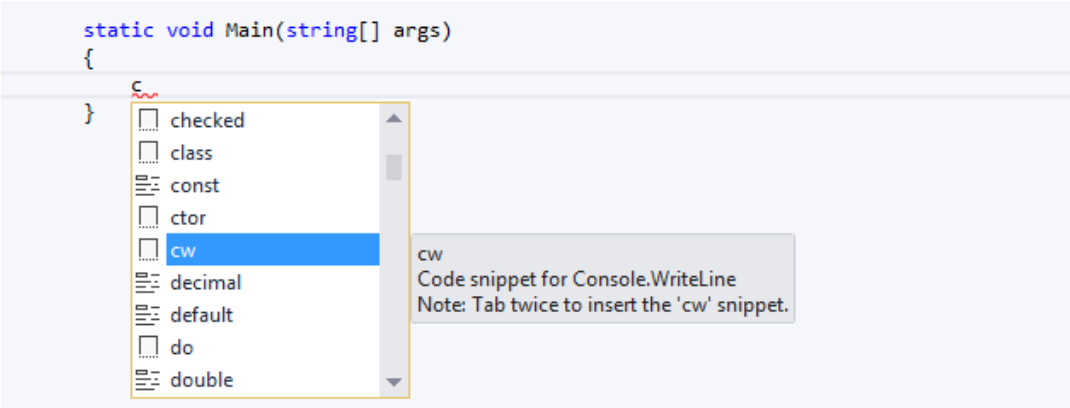
آیکون	مربوط به
	پارامترها و متغیرهای محلی (Locals and Parameters)
	ثابت (Constant)
	خاصیت (Property)
	رویداد (Event)
	فیلد (Field)

	متد (Method)
	رابط (Interface)
	کلاس (Class)
	ساختار (Structure)
	نوع شمارشی (Enum)
	نماینده (Delegate)
	فضای نام (Namespace)
	کلمه کلیدی (Keyword)
	کد کوتاه (Code Snippet)

نگران اسامی ذکر شده در جدول بالا نباشید. آنها را در درس های آینده توضیح خواهیم داد. یکی از قابلیت های جدید که در ویژوال استودیو ۲۰۱۷ اضافه شده است، مرتب کردن لیست IntelliSense می باشد. فرض کنید که شما می خواهید همه کلاس هایی دارای حرف S هستند را در لیست داشته باشید. برای این کار کافیست بر روی آیکون کلاس در IntelliSense کلیک کنید:



همانطور که در شکل بالا مشاهده می کنید همه کلاس هایی که دارای حرف S هستند، لیست می شوند. در زیر یکی دیگر از امکانات ویژوال استودیو که باعث راحتی در کدنویسی می شوند، Code Snippet ها هستند. Code Snippet ها در واقعا مخفف برخی کلمات یا عبارات در ویژوال استودیو هستند. مثلا به جای نوشتن عبارت System.Console.WriteLine(); می توانید cw را نوشته و سپس دو بار دکمه Tab را بزنید تا ویژوال استودیو عبارت مذکور را برای شما کامل کند:



لیست Code Snippet های ویژوال استودیو در لینک زیر آمده است:

```
http://www.w3-farsi.com/?p=2973
```

رفع خطاها

بیشتر اوقات هنگام برنامه‌نویسی با خطا مواجه می‌شویم. تقریباً همه برنامه‌هایی که امروزه می‌بینید حداقل از داشتن یک خطا رنج می‌برند. خطاها می‌توانند برنامه شما را با مشکل مواجه کنند. در سی‌شارپ سه نوع خطا وجود دارد:

خطای کامپایلری

این نوع خطا از اجرای برنامه شما جلوگیری می‌کند. این خطاها شامل خطای دستور زبان می‌باشد. این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده‌اید. یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده می‌کنید که نه وجود دارد و نه ساخته شده است. حذف فایل‌ها یا اطلاعات ناقص در مورد پروژه ممکن است باعث به وجود آمدن خطای کامپایلری شود. استفاده از برنامه بوسیله برنامه دیگر نیز ممکن است باعث جلوگیری از اجرای برنامه و ایجاد خطای کامپایلری شود.

خطاهای منطقی

این نوع خطا در اثر تغییر در یک منطق موجود در برنامه به وجود می‌آید. رفع این نوع خطاها بسیار سخت است چون شما برای یافتن آنها باید کد را تست کنید. نمونه‌ای از یک خطای منطقی برنامه‌ای است که دو عدد را جمع می‌کند ولی حاصل تفریق دو عدد را نشان می‌دهد. در این حالت ممکن است برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

استثناء

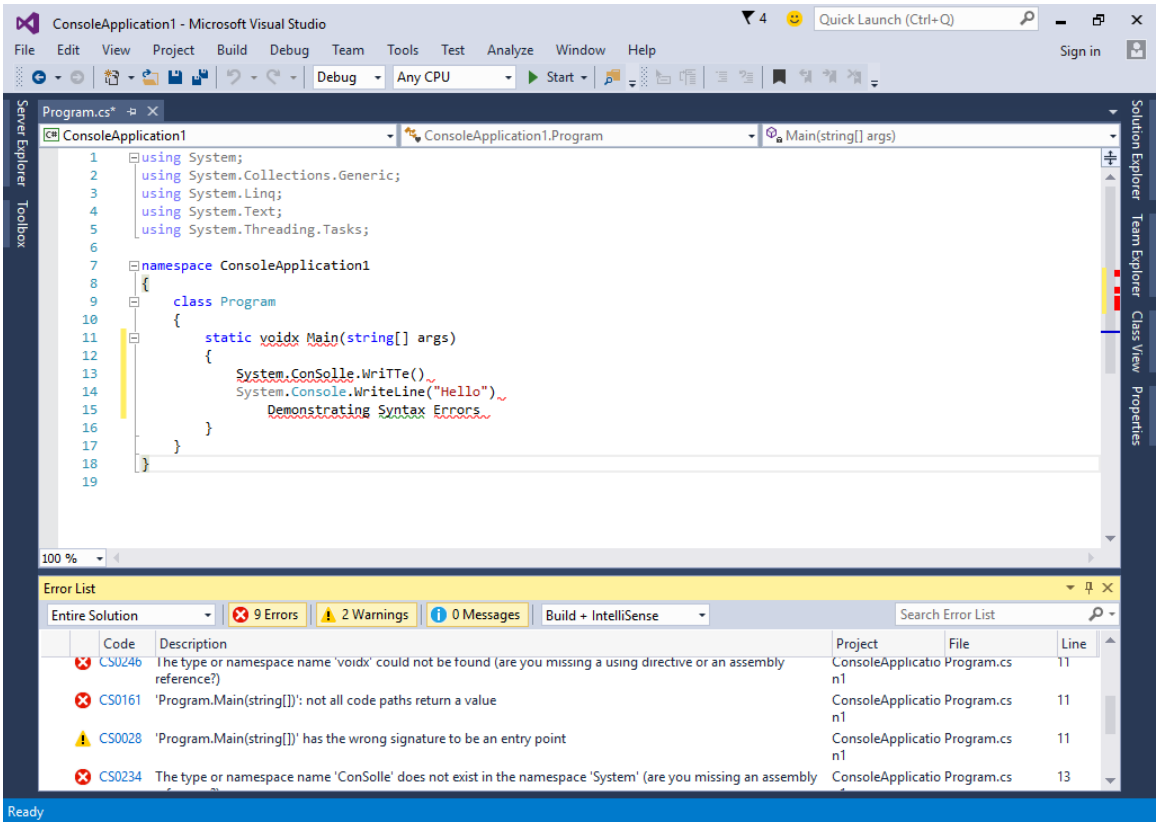
این نوع خطاها هنگامی رخ می‌دهند که برنامه در حال اجراست. این خطا هنگامی روی می‌دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند. ویژوال استودیو و ویژوال سی‌شارپ دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاها هستند. وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگی‌های ویژوال استودیو تشخیص خطاهای ممکن قبل از اجرای برنامه است. زیر کدهایی که دارای خطای کامپایلری هستند خط قرمز کشیده می‌شود.

```
namespace ConsoleApplication1
{
    class Program
    {
        static voidx Main(string[] args)
        {
            System.ConSolle.WriTTe()
            System.Console.WriteLine("Hello")
                Demonstrating Syntax Errors
        }
    }
}
```

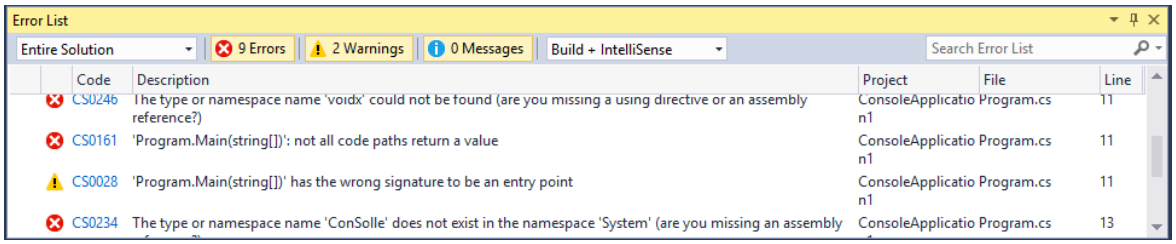
هنگامی که شما با ماوس روی این خطوط توقف کنید توضیحات خطا را مشاهده می‌کنید. شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می‌دهند. به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید (در درس‌های آینده توضیح خواهیم داد).

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int number;
        }
    }
}
```

در باره رفع خطاها در آینده توضیح بیشتری می‌دهیم. ErrorList (لیست خطاها) که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاها، هشدارها و رفع آنها را می‌دهد. برای باز کردن Error List می‌توانید به مسیر View > Other Windows > Error List بروید.



همانطور که در شکل زیر مشاهده می‌کنید هرگاه برنامه شما با خطا مواجه شود لیست خطاها در `Error List` نمایش داده می‌شود.

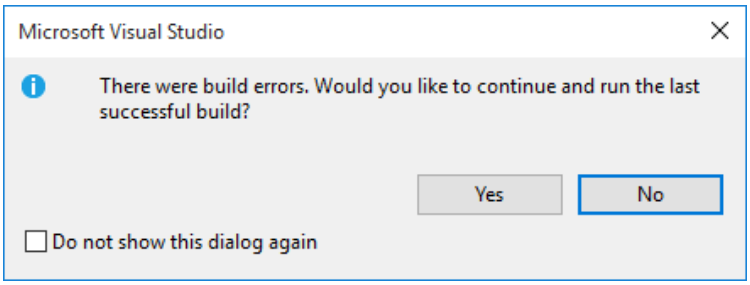


در شکل بالا تعدادی خطا همراه با راه حل رفع آنها در `Error List` نمایش داده شده است. `Error List` دارای چندین ستون است که به طور کامل جزییات خطاها را نمایش می‌دهند.

ستون	توضیحات
`Description`	توضیحی درباره خطا
`File`	فایلی که خطا در آن اتفاق افتاده است.
`Line`	شماره خطی از فایل که دارای خطاست.
`Column`	ستون یا موقعیت افقی خطا در داخل خط

Project	نام پروژه‌ای که دارای خطاست.

اگر برنامه شما دارای خطا باشد و آن را اجرا کنید با پنجره زیر روبه‌رو می‌شوید :



مربع کوچک داخل پنجره بالا را تیک بزنید، تا دفعات بعد که برنامه شما با خطا مواجه شد، دیگر این پنجره به عنوان هشدار نشان داده نشود. با کلیک بر روی دکمه Yes برنامه با وجود خطا نیز اجرا می‌شود. اما با کلیک بر روی دکمه NO اجرای برنامه متوقف می‌شود و شما باید خطاهای موجود در پنجره Error List را بر طرف نمایید. یکی دیگر از ویژگی‌های مهم پنجره Error List نشان دادن قسمتی از برنامه است که دارای خطاست. با یک کلیک ساده بر روی هر کدام خطاهای موجود در پنجره Error List، محل وقوع خطا نمایش داده می‌شود.

خطایابی و برطرف کردن آن

در جدول زیر لیست خطاهای معمول در پنجره Error List و نحوه برطرف کردن آنها آمده است. کلمه Sample، جانشین نام‌های وابسته به خطاهایی است که شما با آنها مواجه می‌شوید و در کل یک مثال است:

خطا	توضیح	راه حل
expected;	در پایان دستور علامت سیمیکان (;) قرار نداده‌اید	اضافه کردن یک سیمیکالن (;)
The name ‘sample’ does not exist in the current context.	کلمه sample در کد شما نه تعریف شده و نه وجود دارد	کلمه sample را حذف یا تعریف کنید.
Only assignment, call, increment, decrement, and new object expressions can be used as a statement.	کد جزء دستورات سی‌شارپ نیست	دستور را حذف کنید.
Use of unassigned local variable ‘sample’	متغیر sample مقدار دهی اولیه نشده	قبل از استفاده از متغیر آن را مقدار دهی اولیه کنید.
The type or namespace name ‘sample’ could not be found (are you missing	نوع یا فضای نام متغیر sample تعریف نشده است	باید یک کلاس یا فضای نام، به نام sample ایجاد کنید.

a using directive or an assembly reference?)		
'MyMethod()': not all code paths return a value	بدین معناست که متد MyMethod() که به عنوان متدی با مقدار برگشتی در نظر گرفته شده در همه قسمت‌های کد دارای مقدار برگشتی نیست.	مطمئن شوید که متد در همه قسمت‌های کد دارای مقدار برگشتی است.
Cannot implicitly convert type 'type1' to 'type2'	متغیر type2 نمی‌تواند به متغیر type1 تبدیل شود.	با استفاده از متدهای تبدیل انواع به هم، دو متغیر را یکسان کنید.

نگران یادگیری کلمات به کار رفته در جدول بالا نباشید چون توضیح آنها در درس‌های آینده آمده است.

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در سی‌شارپ (و بیشتر زبان‌های برنامه نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند. هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```
1  namespace CommentsDemo
2  {
3      class Program
4      {
5          public static void Main(string[] args)
6          {
7              // This line will print the message hello world
8              System.Console.WriteLine("Hello World!");
9          }
10     }
11 }
```

```
Hello World!
```

در کد بالا، خط ۷ یک توضیح درباره خط ۸ است که به کاربر اعلام می‌کند که وظیفه خط ۸ چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط ۷ در خروجی نمایش داده نمی‌شود چون کامپایلر توضیحات را نادیده می‌گیرد. توضیحات بر سه نوع‌اند:

```
// single line comment
```

```
/* multi
   line
```

```
   comment */
```

```
/// <summary>
/// This is XML comments
/// </summary>
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌روند. این توضیحات با علامت // شروع می‌شوند و هر نوشته ای که در سمت راست آن قرار بگیرد جزء توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح در باره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشته ای که بین این دو علامت قرار بگیرد جزء توضیحات محسوب می‌شود. نوع دیگری از توضیحات، توضیحات XML نامیده می‌شوند. این نوع با سه اسلش (///) نشان داده می‌شوند. از این نوع برای مستند سازی برنامه استفاده می‌شود و در درس‌های آینده در مورد آنها توضیح خواهیم داد.

کاراکترهای کنترلی

کاراکترهای کنترلی، کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

```
System.Console.WriteLine("Hello\nWorld!");
```

```
Hello
World!
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. متد WriteLine() هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند:

```
System.Console.WriteLine("Hello World!");
```

کد بالا و کد زیر هیچ فرقی با هم ندارند:

```
System.Console.Write("Hello World!\n");
```

متد Write() کارکردی شبیه به WriteLine() دارد با این تفاوت که نشان گر ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد نمی‌کند.

جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد:

کاراکتر کنترلی	عملکرد	کاراکتر کنترلی	عملکرد
\f	Form Feed	\'	چاپ کوتیشن

\n	خط جدید	\"	چاپ دابل کوتیشن
\r	سر سطر رفتن	\\	چاپ بک اسلش
\t	حرکت به صورت افقی	\0	چاپ فضای خالی
\v	حرکت به صورت عمودی	\a	صدای بیپ
\u	چاپ کاراکتر یونیکد	\b	حرکت به عقب

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه علامت \ معنای خاصی به رشته‌ها می‌دهد، برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم:

```
System.Console.WriteLine("We can print a \\ by using the \\\\ escape sequence.");
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در ویندوز است:

```
System.Console.WriteLine("C:\\Program Files\\Some Directory\\SomeFile.txt");
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (") برای نشان دادن رشته‌ها استفاده می‌کنیم، برای چاپ آن از "\ استفاده می‌کنیم:

```
System.Console.WriteLine("I said, \"Motivate yourself!\".");
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (') از '\ استفاده می‌کنیم:

```
System.Console.WriteLine("The programmer\'s heaven.");
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود:

```
System.Console.WriteLine("Left\tRight");
```

```
Left    Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی \r بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند:

```
System.Console.WriteLine("Mitten\rK");
```

```
Kitten
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی ۳\ آمده است. کاراکتر کنترلی، حرف K را به ابتدای سطر برده و جایگزین حرف M می‌کند. برای چاپ کاراکترهای یونیکد می‌توان از u\ استفاده کرد. برای استفاده از u\، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت u\ قرار می‌دهیم. برای مثال اگر بخواهیم علامت کپی رایت (©) را چاپ کنیم باید بعد از علامت u\ مقدار 00A9 را قرار دهیم مانند:

```
System.Console.WriteLine("\u00A9");
```

```
©
```

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید:

```
http://www.ascii.cl/htmlcodes.htm
```

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

علامت @

علامت @ به شما اجازه می‌دهد که کاراکترهای کنترلی را رد کرده و رشته‌ای خواناتر و طبیعی تر ایجاد کنید. وقتی از کاراکترهای کنترلی در یک رشته استفاده می‌شود، ممکن است برای تایپ مثلاً یک بک اسلش (\) به جای استفاده از دو علامت \\ از یک \ استفاده کرده و دچار اشتباه شوید. این کار باعث به وجود آمدن خطای کامپایلری شده و چون کامپایلر فکر می‌کند که شما می‌خواهید یک کاراکتر کنترلی را تایپ کنید، کاراکتر بعد از علامت \ را پردازش می‌کند و چون کاراکتر کنترلی وجود ندارد خطا به وجود می‌آید. به مثال زیر توجه کنید:

```
System.Console.WriteLine("I want to have a cat\dog as a birthday present."); //Error
```

با وجودیکه بهتر است در مثال بالا از اسلش (/) در cat/dog استفاده شود ولی عمداً از بک اسلش (\) برای اثبات گفته بالا استفاده کرده‌ایم. کامپایلر خطا ایجاد می‌کند و به شما می‌گوید که کاراکتر کنترلی d\ قابل تشخیص نیست، چون چنین کاراکتر کنترلی وجود ندارد. زمانی وضعیت بدتر خواهد شد که کاراکتر بعد از بک اسلش کاراکتری باشد که هم جزء یک کلمه باشد و هم جزء کاراکترهای کنترلی. به مثال زیر توجه کنید:

```
System.Console.WriteLine("Answer with yes\no");
```

```
Answer with yes
o
```

استفاده از علامت @ برای نادیده گرفتن کاراکترهای کنترلی

استفاده از علامت @ زمانی مناسب است که شما نمی‌خواهید از علامت بک اسلش برای نشان دادن یک کاراکتر کنترلی استفاده کنید. استفاده از این علامت بسیار ساده است و کافی است که قبل از رشته مورد نظر آن را قرار دهید.

```
System.Console.WriteLine(@"I want to have a cat\dog as a birthday present.");
```

```
I want to have a cat\dog as a birthday present.
```

از علامت @ معمولاً زمانی استفاده می‌شود که شما بخواهید مسیر یک دایرکتوری را به عنوان رشته داشته باشید. چون دایرکتوری‌ها دارای تعداد زیادی بک اسلش هستند و طبیعتاً استفاده از علامت @ به جای دابل بک اسلش (\\) بهتر است.

```
System.Console.WriteLine(@"C:\Some Directory\SomeFile.txt");
```

```
C:\Some Directory\SomeFile.txt
```

اگر بخواهید یک دابل کوتیشن چاپ کنید به سادگی می‌توانید از دو دابل کوتیشن استفاده کنید.

```
System.Console.WriteLine(@"Printing ""double quotations""...");
```

```
Printing "double quotations"...
```

از به کار بردن علامت @ و کاراکترهای کنترلی به طور همزمان خودداری کنید چون باعث چاپ کاراکتر کنترلی در خروجی می‌شود.

استفاده از علامت @ برای نگهداری از قالب بندی رشته‌ها

یکی دیگر از موارد استفاده از علامت @ چاپ رشته‌های چند خطی بدون استفاده از کاراکتر کنترلی \n است. به عنوان مثال برای چاپ پیغام زیر:

```
C# is a great programming language and
it allows you to create different
kinds of applications.
```

یکی از راه‌های چاپ جمله بالا به صورت زیر است:

```
Console.WriteLine("C# is a great programming language and\n" +
"it allows you to create different\n" +
"kinds of applications.");
```

به نحوه استفاده از \n در آخر هر جمله توجه کنید. این کاراکتر همانطور که قبلاً مشاهده کردید خط جدید ایجاد می‌کند و در مثال بالا باعث می‌شود که جمله به چند خط تقسیم شود. از علامت + هم برای ترکیب رشته‌ها استفاده می‌شود. راه دیگر برای نمایش مثال بالا در چندین خط، استفاده از علامت @ است:

```
Console.WriteLine(@"C# is a great programming language and
it allows you to create different
kinds of applications.");
```

در این حالت کافیست که در هر جا که می‌خواهید رشته در خط بعد نمایش داده شود دکمه Enter را فشار دهید.

متغیرها

متغیر، مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود

یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است.

ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. این مکان همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط، هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد:

- نام متغیر باید با یک از حروف الفبا (a-z or A-Z) شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند #، ?، ^ و $ باشد.
- نمی‌توان از کلمات رزرو شده در سی‌شارپ برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در سی‌شارپ دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نام‌های myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. شما نمی‌توانید دو متغیر را که دقیق شبیه هم هستند را در یک Scope (محدوده) تعریف کنید. Scope به معنای یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد Scope در فصل‌های آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست و نوع آن همان نوع داده‌ای است که در خود ذخیره می‌کند. معمول‌ترین انواع داده int، double، string، char، float، decimal می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع int استفاده کنید.

انواع ساده

انواع ساده، انواعی از داده‌ها هستند که شامل اعداد، کاراکترها، رشته‌ها و مقادیر بولی می‌باشند. به انواع ساده انواع اصلی نیز گفته می‌شود چون از آنها برای ساخت انواع پیچیده تری مانند کلاس‌ها و ساختارها استفاده می‌شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در جدول زیر انواع ساده و محدود آنها آمده است:

نوع	محدوده
sbyte	اعداد صحیح بین ۱۲۸- تا ۱۲۷
byte	اعداد صحیح بین ۰ تا ۲۵۵
short	اعداد صحیح بین ۳۲۷۶۸- تا ۳۲۷۶۷

ushort	اعداد صحیح بین ۰ تا ۶۵۵۳۵
int	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷
uint	اعداد صحیح بین ۰ تا ۴۲۹۴۹۶۷۲۹۵
long	اعداد صحیح بین ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸- تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۸۰۷
ulong	اعداد صحیح بین ۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵

به حرف u در ابتدای برخی از انواع داده‌ها مثلاً ushort توجه کنید. این بدان معناست که این نوع فقط شامل اعداد مثبت و صفر هستند. جدول زیر انواعی که مقادیر با ممیز اعشار را می‌توانند در خود ذخیره کنند، را نشان می‌دهد:

نوع	محدوده	دقت
float	۳٫۴۰۲۸۲۳E۳۸- تا ۳٫۴۰۲۸۲۳E۳۸	۷ رقم
double	۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸- تا ۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸	۱۵-۱۶ رقم
decimal	۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵- تا ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵	۲۸-۲۹ رقم

برای به خاطر سپردن آنها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده‌های غیر عددی به کار می‌روند و در جدول زیر نمایش داده شده‌اند:

نوع	مقادیر مجاز
char	کاراکترهای یونیکد
bool	مقدار true یا false
string	مجموعه‌ای از کاراکترهای

نوع char برای ذخیره کاراکترهای یونیکد استفاده می‌شود. کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند ('a'). نوع bool فقط می‌تواند مقادیر درست (true) یا نادرست (false) را در خود ذخیره کند و بیشتر در برنامه‌هایی که دارای ساختار تصمیم گیری هستند، مورد استفاده قرار می‌گیرد. نوع string برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود. مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کامپایلر به عنوان یک رشته در نظر گرفته شوند، مانند ("massage").

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است:

```
using System;

public class Program
{
    public static void Main()
    {
        //Declare variables
        int num1;
        int num2;
        double num3;
        double num4;
        bool boolVal;
        char myChar;
        string message;

        //Assign values to variables
        num1 = 1;
        num2 = 2;
        num3 = 3.54;
        num4 = 4.12;
        boolVal = true;
        myChar = 'R';
        message = "Hello World!";

        //Show the values of the variables
        Console.WriteLine("num1 = {0}", num1);
        Console.WriteLine("num2 = {0}", num2);
        Console.WriteLine("num3 = {0}", num3);
        Console.WriteLine("num4 = {0}", num4);
        Console.WriteLine("boolVal = {0}", boolVal);
        Console.WriteLine("myChar = {0}", myChar);
        Console.WriteLine("message = {0}", message);
    }
}
```

```
num1 = 1
num2 = 2
num3 = 3.54
num4 = 4.12
boolVal = true
myChar = R
message = Hello World!
```

تعریف متغیر

در خطوط ۸-۱۴ متغیرهایی با نوع و نام متفاوت تعریف شده‌اند. ابتدا باید نوع داده‌هایی را که این متغیرها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آنها در نظر بگیریم و در آخر سمیکالن بگذاریم. همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعریف کرد.

```
int num1;
int num2;
double num3;
double num4;
bool boolVal;
char myChar;
```

```
string message;
```

نحوه تعریف متغیر به صورت زیر است:

```
data_type identifier;
```

date_type همان نوع داده است مانند int، double و ... . identifier نیز نام متغیر است که به ما امکان استفاده و دسترسی به مقدار متغیر را می‌دهد. برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد:

```
data_type identifier1, identifier2, ... indentifierN;
```

مثال

```
int num1, num2, num3, num4, num5;
string message1, message2, message3;
```

در مثال بالا ۵ متغیر از نوع صحیح و ۳ متغیر از نوع رشته تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما (,) باشد.

نامگذاری متغیرها

نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود. نمی‌توان از کاراکترهای خاص مانند #، %، & یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers. نام متغیر نباید دارای فاصله باشد. برای نام‌های چند حرفی می‌توان به جای فاصله از علامت زیرخط یا _ استفاده کرد.

نام‌های مجاز:

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average       amountDue   last_name     _maximum
name  counter   sum           isLeapYear  color_of_car  _age
```

نام‌های غیر مجاز:

```
123         #numbers#  #ofstudents  1abc2
123abc      $money     first name   ty.np
my number   this&that  last name    1:00
```

اگر به نام‌های مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آنها خواهید شد. یکی از روش‌های نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه ای به کار می‌رود، اولین کلمه با حرف کوچک نوشته می‌شود و سایر کلمات با حرف بزرگ شروع می‌شوند. مانند myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید بعد از اولین کلمه، حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

در کد ابتدای درس، متغیرها در داخل متد Main() تعریف شده اند. در نتیجه، این متغیرها فقط در داخل متد Main() قابل دسترسی هستند. محدوده یک متغیر مشخص می‌کند که متغیر در کجای کد قابل دسترسی است. هنگامیکه برنامه به پایان متد Main() می‌رسد متغیرها از محدوده خارج و بدون استفاده می‌شوند. محدوده متغیرها انواعی دارد که در درس‌های بعدی با آنها آشنا می‌شوید. تشخیص محدوده متغیر بسیار مهم است. چون به وسیله آن می‌فهمید که در کجای کد می‌توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی‌توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطا می‌کند:

```
int num1;
int num1;
```

از آنجاییکه سی‌شارپ به بزرگی و کوچک بودن حروف حساس است می‌توان از این خاصیت برای تعریف چند متغیر همنام ولی با حروف متفاوت (از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند:

```
int num1;
int Num1;
int NUM1;
```

مقداردهی متغیرها

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آنها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقدار دهی متغیرها نشان داده شده است:

```
data_type identifier = value;
```

به عنوان مثال:

```
int myNumber = 7;
```

همچنین می‌توان چندین متغیر را فقط با گذاشتن کاما بین آنها به سادگی مقدار دهی کرد:

```
data_type variable1 = value1, varaible2 = value2, ... variableN, valueN;
```

به عنوان مثال:

```
int num1 = 1, num2 = 2, num3 = 3;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1;
num2 = 2;
num3 = 3.54;
num4 = 4.12;
boolVal = true;
myChar = 'R';
message = "Hello World!";
```

به این نکته توجه کنید که شما به متغیری که هنوز تعریف نشده نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده‌اند و مقادیری از نوع صحیح به آنها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد.

جانگهدار (Placeholders)

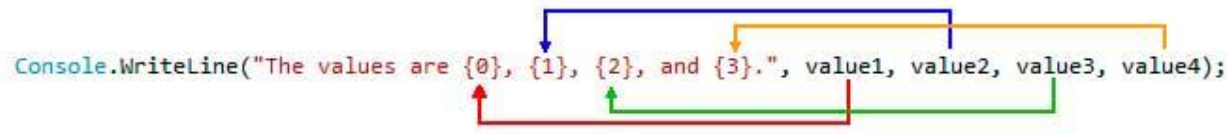
به متد WriteLine() در خطوط (۳۲-۲۶) توجه کنید. این متد دو آرگومان قبول می‌کند. آرگومانها اطلاعاتی هستند که متد با استفاده از آنها کاری انجام می‌دهد. آرگومانها به وسیله کاما از هم جدا می‌شوند. آرگومان اول، یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرد.

```
Console.WriteLine("num1 = {0}", num1);
Console.WriteLine("num2 = {0}", num2);
Console.WriteLine("num3 = {0}", num3);
Console.WriteLine("num4 = {0}", num4);
Console.WriteLine("boolVal = {0}", boolVal);
Console.WriteLine("myChar = {0}", myChar);
Console.WriteLine("message = {0}", message);
```

اگر به دقت نگاه کنید رشته قالب بندی شده دارای عدد صفری است که در داخل دو آکولاد محصور شده است. البته عدد داخل دو آکولاد می‌تواند از صفر تا n باشد. به این اعداد جانگهدار (Placeholder) می‌گویند. این اعداد بوسیله مقدار آرگومان بعد جایگزین می‌شوند. به عنوان مثال جانگهدار {۰} به این معناست که اولین آرگومان (مقدار) بعد از رشته قالب بندی شده در آن قرار می‌گیرد.

متد WriteLine() عملاً می‌تواند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جا نگهدار در آن قرار دارد و دومین آرگومان مقداری است که جایگزین جانگهدار می‌شود. در مثال زیر از چهار جا نگهدار استفاده شده است:

```
Console.WriteLine("The values are {0}, {1}, {2}, and {3}.", value1, value2, value3, value4);
```



جا نگهدارها از صفر شروع می‌شوند. تعداد جانگهدارها باید با تعداد آرگومانهای بعد از رشته قالب بندی شده برابر باشد. برای مثال اگر شما چهار جا نگهدار مثل بالا داشته باشید باید چهار مقدار هم برای آنها بعد از رشته قالب بندی شده در نظر بگیرید. اولین جا نگهدار با دومین آرگومان و

دومین جا نگهدار با سومین آرگومان جایگزین می‌شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه‌نویسی را شروع کرده‌اند سخت است، اما در درس‌های آینده مثال‌های زیادی در این مورد مشاهده خواهید کرد.

وارد کردن فضاهای نام

شاید به این نکته توجه کرده باشید که ما زمان فراخوانی متد ()WriteLine و قبل از Console، کلمه System را ننوشتیم چون در خط ۱ و در ابتدای برنامه این کلمه را در قسمت تعریف فضای نام وارد کردیم.

```
using System;
```

این دستور بدین معناست که ما از تمام چیزهایی که در داخل فضای نام System قرار دارند، استفاده می‌کنیم. پس به جای اینکه جمله زیر را به طور کامل بنویسیم:

```
System.Console.WriteLine("Hello World!");
```

می‌توانیم آن را ساده تر کرده و به صورت زیر بنویسیم:

```
Console.WriteLine("Hello World");
```

در مورد فضای نام در درس‌های آینده توضیح خواهیم داد.

ثابت‌ها

ثابت‌ها انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد، هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی const استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند تا تشخیص آنها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است:

```
const data_type identifier = initial_value;
```

مثال:

```
class Program
{
    public static void Main()
    {
        const int NUMBER = 1;

        NUMBER = 10; //ERROR, Cant modify a constant
    }
}
```

در این مثال می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می‌کند. نکته‌ی دیگری که نباید فراموش شود این است که، نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد. به مثال زیر توجه کنید:

```
int someVariable;
constint MY_CONST = someVariable
```

ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت‌ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی‌کنند، بهتر است که آنها را به صورت ثابت تعریف کنید. این کار هر چند کوچک، کیفیت برنامه شما را بالا می‌برد.

تبدیل ضمنی

تبدیل ضمنی متغیرها یک نوع تبدیل است که به طور خودکار توسط کامپایلر انجام می‌شود. یک متغیر از یک نوع داده می‌تواند به طور ضمنی به یک نوع دیگر تبدیل شود به شرطی که مقدار آن از مقدار داده‌ای که می‌خواهد به آن تبدیل شود کمتر باشد. به عنوان مثال نوع داده‌ای byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده‌ای int مقادیر ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید یک متغیر از نوع byte را به یک نوع int تبدیل کنید:

```
byte number1 = 5;
int number2 = number1;
```

در مثال بالا مقدار number1 برابر ۵ است در نتیجه متغیر number2 که یک متغیر از نوع صحیح است، می‌تواند مقدار number1 را در خود ذخیره کند. چون نوع int از نوع byte بزرگ‌تر است، پس متغیر number1 که یک متغیر از نوع byte است می‌تواند به طور ضمنی به number2 که یک متغیر از نوع صحیح است تبدیل شود. اما عکس مثال بالا صادق نیست:

```
int number1 = 5;
byte number2 = number1;
```

در این مورد ما با خطا مواجه می‌شویم. اگر چه مقدار ۵ متغیر number1 در محدوده مقادیر byte یعنی اعداد بین ۲۵۵-۰ قرار دارد اما متغیری از نوع byte حافظه کمتری نسبت به متغیری از نوع int اشغال می‌کند. نوع byte شامل ۸ بیت یا ۸ رقم دودویی است، در حالی که نوع int شامل ۳۲ بیت یا رقم دودویی است. یک عدد باینری عددی متشکل از اعداد ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بایت ذخیره می‌کنیم عددی به صورت زیر نمایش داده می‌شود:

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم به صورت زیر نمایش داده می‌شود:

```
00000000000000000000000000000101
```

بنابراین قرار دادن یک مقدار int در یک متغیر byte درست مانند این است که ما سعی کنیم که یک توپ فوتبال را در یک سوراخ کوچک گلف جای دهیم. برای قرار دادن یک مقدار int در یک متغیر از نوع byte می‌توان از تبدیل صریح استفاده کرد که در درس‌های آینده توضیح داده می‌شود. نکته دیگری که نباید فراموش شود این است که شما نمی‌توانید اعداد با ممیز اعشار را به یک نوع int تبدیل کنید چون این کار باعث از بین رفتن بخش اعشاری این اعداد می‌شود.

```
double number1 = 5.25;
```

```
int number2 = number1; //Error
```

می‌توان یک نوع کاراکتر را به نوع ushort تبدیل کرد، چون هر دو دارای طیف مشابهی از اعداد هستند. گرچه هر یک از آنها کاملاً متفاوت توسط کامپایلر ترجمه می‌شوند. نوع char به عنوان یک کاراکتر و نوع ushort به عنوان یک عدد ترجمه می‌شود.

```
char charVar = 'c';
ushort shortVar = charVar;

Console.WriteLine(charVar);
Console.WriteLine(shortVar);
```

```
c
99
```

تبدیلاتی که کامپایلر به صورت ضمنی می‌تواند انجام دهد در جدول زیر آمده است:

نوع	قابلیت تبدیل به انواع
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

نکته‌ای دیگر که معمولاً ابهام بر انگیز است تعین نوع داده است. برای مثال ما چطور بدانیم که مثلاً عدد ۷ از نوع int، uint، long یا ulong است؟ برای این کار باید کاراکترهایی را به انتهای اعداد اضافه کنیم.

```
uint number1 = 7U;
long number2 = 7L;
ulong number3 = 7UL;
```

در حالت پیش‌فرض و بدون قرار دادن کاراکتر در انتهای عدد، کامپایلر عدد را از نوع صحیح (int) در نظر می‌گیرد و در حالت پیش‌فرض کامپایلر اعداد دسیمال (decimal) را اعداد double در نظر می‌گیرد. شما می‌توانید برای نشان دادن اعداد اعشاری float از کاراکتر F و برای نشان دادن اعداد دسیمال از کاراکتر M استفاده کنید.

```
double number1 = 1.23;
float number2 = 1.23F;
```

```
decimal number3 = 1.23M
```

تبدیل صریح

تبدیل صریح نوعی تبدیل است که برنامه را مجبور می‌کند که یک نوع داده را به نوعی دیگر تبدیل کند، اگر این نوع تبدیل از طریق تبدیل ضمنی انجام نشود. در هنگام استفاده از این تبدیل باید دقت کرد. چون در این نوع تبدیل ممکن است مقادیر اصلاح یا حذف شوند. ما می‌توانیم این عملیات را با استفاده از Cast انجام دهیم. Cast فقط نام دیگر تبدیل صریح است و دستور آن به صورت زیر است:

```
datatypeA variableA = value;
datatypeB variableB = (datatypeB)variableA;
```

همانطور که قبلاً مشاهده کردید نوع int را نتوانستیم به نوع byte تبدیل کنیم، اما اکنون با استفاده از عمل Cast این تبدیل انجام خواهد شد:

```
int number1 = 5;

byte number2 = (byte)number1;
```

حال اگر برنامه را اجرا کنید با خطا مواجه نخواهید شد. همانطور که پیش‌تر اشاره شد ممکن است در هنگام تبدیلات مقادیر اصلی تغییر کنند. برای مثال وقتی که یک عدد با ممیز اعشار مثلاً از نوع double را به یک نوع int تبدیل می‌کنیم مقدار اعداد بعد از ممیز از بین می‌روند:

```
double number1 = 5.25;

int number2 = (int)number1;

Console.WriteLine(number2)
```

```
5
```

خروجی کد بالا عدد ۵ است چون نوع داده‌ای int نمی‌تواند مقدار اعشار بگیرد. حالت دیگر را تصور کنید. اگر شما بخواهید یک متغیر را که دارای مقداری بیشتر از محدوده متغیر مقصد هست تبدیل کنید چه اتفاقی می‌افتد؟ مانند تبدیل زیر که می‌خواهیم متغیر number1 را که دارای مقدار ۳۰۰ است را به نوع Byte که محدود اعداد بین ۲۵۵-۰ را پوشش می‌دهد، تبدیل کنیم.

```
int number1 = 300;

byte number2 = (byte)number1;

Console.WriteLine("Value of number2 is {0}.", number2);
```

```
Value of number2 is 44.
```

خروجی کد بالا عدد ۴۴ است. Byte فقط می‌تواند شامل اعداد ۰ تا ۲۵۵ باشد و نمی‌تواند مقدار ۳۰۰ را در خود ذخیره کند. حال می‌خواهیم ببینیم که چرا به جای عدد ۳۰۰، عدد ۴۴ را در خروجی نمایش داده می‌شود. این کار به تعداد بیت‌ها بستگی دارد. یک byte دارای ۸ بیت است درحالی که int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
 44 =                         00101100
```

خروجی بالا نشان می‌دهد که بیشترین مقدار byte که عدد ۲۵۵ است، می‌تواند فقط شامل ۸ بیت باشد (11111111) بنابراین فقط ۸ بیت اول مقدار int به متغیر byte انتقال می‌یابد که شامل (00101100) یا عدد ۴۴ در مبنای ۱۰ است. قرار ندادن یک مقدار مناسب در داخل یک متغیر باعث ایجاد یک سرریز (overflow) می‌شود. یک مورد آن سرریز ریاضی نام دارد که در مثال زیر مشاهده می‌کنید:

```
byte sum = (byte)(150 + 150);
```

گرچه در این تبدیل ما داده‌هایی را از دست می‌دهیم، اما کامپایلر کد ما را قبول می‌کند. برای اینکه برنامه هنگام وقوع سرریز پیغام خطا بدهد می‌توان از کلمه کلیدی checked استفاده کرد.

```
int number1 = 300;
byte number2 = checked((byte)number1);
Console.WriteLine("Value of number2 is {0}.", number2)
```

```
Unhandled Exception: System.OverflowException: Arithmetic operation resulted in an overflow ...
```

برنامه پیغام System.OverflowException که به زبان ساده نشان دهند وقوع خطاست. در نتیجه شما می‌توانید از اجرای برنامه جلوگیری کنید.

تبدیل با استفاده از کلاس Convert

.NET Framework دارای یک کلاس استاتیک است، که می‌توان از آن برای تبدیل مقادیر از نوعی به نوع دیگر استفاده کرد. این کلاس به نوبه خود دارای متدهایی برای تبدیل انواع داده به یکدیگر می‌باشد. در جدول زیر متدها ذکر شده‌اند:

دستور	نتیجه
Convert.ToBoolean(val)	مقدار val به نوع bool تبدیل می‌شود.
Convert.ToByte(val)	مقدار val به نوع byte تبدیل می‌شود.
Convert.ToChar(val)	مقدار val به نوع char تبدیل می‌شود.
Convert.ToDecimal(val)	مقدار val به نوع decimal تبدیل می‌شود.
Convert.ToDouble(val)	مقدار val به نوع double تبدیل می‌شود.
Convert.ToInt16(val)	مقدار val به نوع short تبدیل می‌شود.
Convert.ToInt32(val)	مقدار val به نوع int تبدیل می‌شود.
Convert.ToInt64(val)	مقدار val به نوع long تبدیل می‌شود.
Convert.ToSByte(val)	مقدار val به نوع ushort تبدیل می‌شود.

`Convert.ToSingle(val)`	مقدار `val` به نوع `float` تبدیل می‌شود.
`Convert.ToString(val)`	مقدار `val` به نوع `string` تبدیل می‌شود.
`Convert.ToUInt16(val)`	مقدار `val` به نوع `ushort` تبدیل می‌شود.
`Convert.ToUInt32(val)`	مقدار `val` به نوع `uint` تبدیل می‌شود.
`Convert.ToUInt64(val)`	مقدار `val` به نوع `ulong` تبدیل می‌شود.

در برنامه زیر یک نمونه از تبدیل متغیرها با استفاده از کلاس `Convert` و متدهای آن نمایش داده شده است:

```
double x = 9.99;
int convertedValue = Convert.ToInt32(x);

Console.WriteLine("Original value is: " + x);
Console.WriteLine("Converted value is: " + convertedValue);
```

```
Original value is: 9.99
Converted value is: 10
```

مقدار `val` هر نوع داده‌ای می‌تواند باشد، اما باید مطمئن شد که به نوع داده‌ای مورد نظر تبدیل شود.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید:

- عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.
- عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.

مثلاً X+Y یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند. زبان‌های برنامه‌نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. سی‌شارپ دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایس‌های، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. سه نوع عملگر در سی‌شارپ وجود دارد:

- یگانی (Unary) – به یک عملوند نیاز دارد
- دودویی (Binary) – به دو عملوند نیاز دارد
- سه تایی (Ternary) – به سه عملوند نیاز دارد

انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند عبارت‌اند از:

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه ای

- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

سی‌شارپ از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی سی‌شارپ را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
+	Binary	var1 = var2 + var3;	Var1 برابر است با حاصل جمع var2 و var3
-	Binary	var1 = var2 – var3;	Var1 برابر است با حاصل تفریق var2 و var3
*	Binary	var1 = var2 * var3;	Var1 برابر است با حاصلضرب var2 در var3
/	Binary	var1 = var2 / var3;	Var1 برابر است با حاصل تقسیم var2 بر var3
%	Binary	var1 = var2 % var3;	Var1 برابر است با باقیمانده تقسیم var2 و var3
+	Unary	var1 = +var2;	Var1 برابر است با مقدار var2
-	Unary	var1 = -var2	Var1 برابر است با مقدار var2 ضربدر ۱-

در مثال بالا از نوع عددی استفاده شده است. اما استفاده از عملگرهای ریاضی برای نوع رشته‌ای نتیجه متفاوتی دارد. همچنین در جمع دو کاراکتر کامپایلر معادل عددی آنها را نشان می‌دهد. اگر از عملگر + برای رشته‌ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می‌چسباند. دیگر عملگرهای سی‌شارپ عملگرهای کاهش و افزایش هستند. این عملگرها مقدار ۱ را از متغیرها کم یا به آنها اضافه می‌کنند. از این متغیرها اغلب در حلقه‌ها استفاده می‌شود:

عملگر	دسته	مثال	نتیجه
++	Unary	var1 = ++var2;	مقدار var1 برابر است با var2 بعلاوه ۱. از متغیر var2 یک واحد اضافه می‌شود.
--	Unary	var1 = --var2;	مقدار var1 برابر است با var2 منهای ۱. از متغیر var2 یک واحد کم می‌شود.
++	Unary	var1 = var2++;	مقدار var1 برابر است با var2. به متغیر var2 یک واحد اضافه می‌شود.
--	Unary	var1 = var2--;	مقدار var1 برابر است با var2.

			از متغیر var2 یک واحد کم می‌شود.

به این نکته توجه داشته باشید که محل قرارگیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بیاید افزایش یا کاهش var1 اتفاق می‌افتد و var2 تغییر نمی‌کند. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می‌شود و سپس متغیر var2 افزایش یا کاهش می‌یابد. به مثال‌های زیر توجه کنید:

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = ++y;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=2
y=2
```

```
using System;

using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = --y;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=0
y=0
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ قبل از عملوند y باعث می‌شود که ابتدا یک واحد از y کم و یا یک واحد به y اضافه شود و سپس نتیجه در عملوند x قرار بگیرد. حال به دو مثال زیر توجه کنید:

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = y--;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=1
y=0
```

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0;
            int y = 1;

            x = y++;

            Console.WriteLine("x= {0}",x);
            Console.WriteLine("y= {0}", y);
            Console.ReadLine();
        }
    }
}
```

```
x=1
y=2
```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای -- و ++ بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار بگیرد و سپس یک واحد از y کم و یا یک واحد به آن اضافه شود. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در سی‌شارپ را یاد بگیریم:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           //Variable declarations
8           int num1, num2;
9           string msg1, msg2;
10
```

```
11          //Assign test values
12          num1 = 5;
13          num2 = 3;
14
15          //Demonstrate use of mathematical operators
16          Console.WriteLine("The sum of {0} and {1} is {2}.",
17                            num1, num2, (num1 + num2));
18          Console.WriteLine("The difference of {0} and {1} is {2}.",
19                            num1, num2, (num1 - num2));
20          Console.WriteLine("The product of {0} and {1} is {2}.",
21                            num1, num2, (num1 * num2));
22          Console.WriteLine("The quotient of {0} and {1} is {2:F2}.",
23                            num1, num2, ((double)num1 / num2));
24          Console.WriteLine("The remainder of {0} divided by {1} is {2}",
25                            num1, num2, (num1 % num2));
26
27          //Demonstrate concatenation on strings using the + operator
28          msg1 = "Hello ";
29          msg2 = "World!";
30          Console.WriteLine(msg1 + msg2);
31      }
32  }
```

```
The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از متد Writeline() برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در این مثال با یک نکته عجیب مواجه می‌شویم و آن حاصل تقسیم دو عدد صحیح است. وقتی که دو عدد صحیح را بر هم تقسیم کنیم حاصل باید یک عدد صحیح و فاقد بخش کسری باشد. اما همانطور که مشاهده می‌کنید اگر فقط یکی از اعداد را به نوع اعشاری double تبدیل کنیم (در مثال می‌بینید) حاصل به صورت اعشار نشان داده می‌شود. برای اینکه ارقام کسری بعد از عدد حاصل دو رقم باشند از {2:F2} استفاده می‌کنیم. F به معنای فرمت بندی می‌باشد و در این جا بدین معناست که عدد را تا دو رقم اعشار نمایش بده. چون خطوط کد طولانی هستند آنها را در دو خط می‌نویسیم. سی‌شارپ خط جدید، فاصله و فضای خالی را نادیده می‌گیرد.

در خط ۲۹ مشاهده می‌کنید که دو رشته به وسیله عملگر + به هم متصل شده‌اند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه "Hello " و "World!" رشته "Hello World!" خواهد بود. به فاصله خالی بعد از کلمه Hello توجه کنید اگر آن را حذف کنید از خروجی برنامه نیز حذف می‌شود.

عملگرهای تخصیصی (جایگزینی)

نوع دیگر از عملگرهای سی‌شارپ عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در سی‌شارپ را نشان می‌دهد:

عملگر	مثال	نتیجه
=	var1 = var2;	مقدار var1 برابر است با مقدار var2.

+=	var1 += var2;	مقدار var1 برابر است با حاصل جمع var1 و var2.
-=	var1 -= var2;	مقدار var1 برابر است با حاصل تفریق var1 و var2.
*=	var1 *= var2;	مقدار var1 برابر است با حاصل ضرب var1 در var2.
/=	var1 /= var2;	مقدار var1 برابر است با حاصل تقسیم var1 بر var2.
%=	var1 %= var2;	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2.

از عملگر =+ برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می‌دهد.

```
using System;

public class Program
{
    public static void Main()
    {
        int number;

        Console.WriteLine("Assigning 10 to number...");
        number = 10;
        Console.WriteLine("Number = {0}", number);

        Console.WriteLine("Adding 10 to number...");
        number += 10;
        Console.WriteLine("Number = {0}", number);

        Console.WriteLine("Subtracting 10 from number...");
        number -= 10;
        Console.WriteLine("Number = {0}", number);
    }
}
```

```
Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از سه عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد، مقدار true و اگر نتیجه مقایسه اشتباه باشد، مقدار false را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند. به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه ای در سی‌شارپ را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
==	Binary	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است.
!=	Binary	var1 = var2 != var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است.
<	Binary	var1 = var2 < var3	var1 در صورتی true است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت false است.
>	Binary	var1 = var2 > var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر از مقدار var3 باشد در غیر اینصورت false است.
<=	Binary	var1 = var2 <= var3	var1 در صورتی true است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت false است.
>=	Binary	var1 = var2 >= var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت false است.

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد:

```
using System;

namespace ComparisonOperators
{
    class Program
    {
        static void Main()
        {
            int num1 = 10;
            int num2 = 5;

            Console.WriteLine("{0} == {1} : {2}", num1, num2, num1 == num2);
            Console.WriteLine("{0} != {1} : {2}", num1, num2, num1 != num2);
            Console.WriteLine("{0} <  {1} : {2}", num1, num2, num1 < num2);
            Console.WriteLine("{0} >  {1} : {2}", num1, num2, num1 > num2);
            Console.WriteLine("{0} <= {1} : {2}", num1, num2, num1 <= num2);
```

```
            Console.WriteLine("{0} >= {1} : {2}", num1, num2, num1 >= num2);
        }
    }
}
```

```
10 == 5 : False
10 != 5 : True
10 <  5 : False
10 >  5 : True
10 <= 5 : False
10 >= 5 : True
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به x اختصاص می‌دهد. عملگر == عملگر مقایسه ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y.

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرط‌های پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید مقادیر بولی می‌توانند false یا true باشند. فرض کنید که var2 و var3 دو مقدار بولی هستند.

عملگر	نام	دسته	مثال
&&	منطقی AND	Binary	var1 = var2 && var3;
||	منطقی OR	Binary	var1 = var2 || var3;
!	منطقی NOT	Unary	var1 = !var1;

عملگر منطقی (&&)AND

اگر مقادیر دو طرف این عملگر، true باشند، عملگر AND مقدار true را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها false باشند، مقدار false را بر می‌گرداند. در زیر جدول درستی عملگر AND نشان داده شده است:

X && Y	Y	X
true	true	true
false	false	true
false	true	false
false	false	false

برای درک بهتر تأثیر عملگر AND یادآوری می‌کنیم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عملوند مقدارشان true باشد. در غیر اینصورت نتیجه تمام ترکیب‌های بعدی، false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارآمد است که ما با محدوده خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

عملگر منطقی OR(||)

اگر یکی یا هر دو مقدار دو طرف عملگر OR، درست (true) باشد، عملگر OR مقدار true را بر می‌گرداند. جدول درستی عملگر OR در زیر نشان داده شده است:

X	Y	X || Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می‌کنید که عملگر OR در صورتی مقدار false را بر می‌گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true) است، که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

عملگر منطقی NOT(!)

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا عبارت بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

```
isMinor = !(age >= 18);
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد اعشاری به باینری را از لینک زیر یاد بگیرید:

```
http://www.w3-farsi.com/?p=5698
```

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می‌گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع int به ۳۲ بیت یا ۴ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود:

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت چپ برای پر کردن بیت‌هایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی سی‌شارپ در جدول زیر نشان داده شده‌اند:

عملگر	نام	دسته	مثال
&	بیتی AND	Binary	x = y & z;
|	بیتی OR	Binary	x = y | z;
^	بیتی XOR	Binary	x = y ^ z;
~	بیتی NOT	Unary	x = ~y;
&=	بیتی – تخصیصی AND	Binary	x &= y;
|=	بیتی – تخصیصی OR	Binary	x |= y;
^=	بیتی – تخصیصی XOR	Binary	x ^= y;

عملگر بیتی AND(&)

عملگر بیتی AND مانند کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیت‌ها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد، مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عمگر بیتی AND در زیر آمده است:

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است:

```
int result = 5 & 3;

Console.WriteLine(result);
```

```
1
```

همانطور که در مثال بالا مشاهده می‌کنید، نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد ۱ می‌شود. اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
-----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (int) ۳۲ بیت است از صفر برای پر کردن بیت‌های خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی OR(|)

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه ۰ در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است:

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی OR در صورتی ۰ است که عملوندهای دو طرف آن ۰ باشند. اگر فقط یکی از دو عملوند ۱ باشد، نتیجه ۱ خواهد شد. به مثال زیر توجه کنید:

```
int result = 7 | 9;

Console.WriteLine(result);
```

```
15
```

وقتی که از عملگر بیتی OR برای دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم، نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

```
7: 00000000000000000000000000000111
9: 00000000000000000000000000001001
---------------------------------
15:00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

عملگر بیتی XOR (^)

جدول درستی این عملگر در زیر آمده است:

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو ۰ یا هر دو ۱ باشند نتیجه ۰، در غیر اینصورت نتیجه ۱ می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید:

```
int result = 5 ^ 7;

Console.WriteLine(result);
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
---------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر بیتی NOT (~)

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:

X	NOT X
1	0
0	1

عملگر بیتی NOT مقادیر بیت‌ها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است:

```
int result = ~7;

Console.WriteLine(result);
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید.

```
7: 00000000000000000000000000000111
-----------------------------------
-8: 11111111111111111111111111111000
```

مثال‌هایی از عملگرهای بیتی

فرض کنید که از یک سبک خاص فونت در برنامه‌تان استفاده کنید. کدهای مربوط به هر سبک هم در جدول زیر آمده است:

کد	سبک
0	Regular
1	Bold
2	Italic
4	Underline
8	Strikeout

توجه کنید که مقدار اولیه ۰ بدین معنی است که می‌خواهید از سبک regular (عادی) استفاده کنید.

```
int fontStyle = 0;
```

برای نشان دادن فونت‌ها به صورت کلفت (Bold) از عملگر بیتی OR استفاده می‌شود. توجه کنید که برای فونت Bold باید کد ۱ را به کار برید.

```
fontStyle = fontStyle | 1;
```

برای استفاده از سبک Italic باید از عملگر بیتی OR و کد ۲ استفاده شود.

```
fontStyle |= 2;
```

برای استفاده از سایر سبک‌ها می‌توان به روش‌های ذکر شده در بالا عمل کرد و فقط کدها را جایگزین کنید. اگر بخواهید یک سبک جدید ایجاد کنید که ترکیبی از چند سبک باشد، می‌توانید به سادگی عملگر بیتی OR را در بین هر سبک فونت قرار دهید مانند مثال زیر:

```
fontStyle = 1 | 2 | 4 | 8;
```

عملگر بیتی تغییر مکان (shift)

این نوع عملگرها به شما اجازه می‌دهند که بیت‌ها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

عملگر	نام	دسته	مثال
<<	تغییر مکان به سمت چپ	Binary	x = y << 2;
>>	تغییر مکان به سمت راست	Binary	x = y >> 2;

عملگر تغییر مکان به سمت چپ

این عملگر، بیت‌های عملوند سمت چپ را به تعداد n مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می‌کند. به عنوان مثال:

```
int result = 10 << 2;

Console.WriteLine(result);
```

```
40
```

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم:

```
10: 00000000000000000000000000001010
----------------------------------
40: 00000000000000000000000000101000
```

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عملگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال:

```
int result = 100 >> 4;

Console.WriteLine(result);
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم:

```
100: 00000000000000000000000001100100
----------------------------------
  6: 00000000000000000000000000000110
```

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

تقدم عملگرها

تقدم عملگرها مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند، ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در سی‌شارپ در محاسبات دارای حق تقدم هستند. به عنوان مثال:

```
number = 1 + 2 * 3 / 1;
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه ۹ خواهد شد (۳=۲+۱ سپس ۹=۳×۳ و در آخر ۹/۱=۹). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آنها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید. در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای سی‌شارپ آمده است:

عملگرها	تقدم
++, --, (used as prefixes); +, - (unary)	بالاترین
*, /, %	
+, -	
<<, >>	
<, >, <=, >=	
==, !=	
&	
^	
|	
&&	
||	
=, *=, /=, %=, +=, -=	
++, -- (used as suffixes)	پایین‌ترین

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. به این نکته توجه کنید که تقدم عملگرها ++ و -- به مکان قرارگیری آنها بستگی دارد (در سمت چپ یا راست عملوند باشند). به عنوان مثال:

```
int number = 3;

number1 = 3 + ++number; //results to 7
number2 = 3 + number++; //results to 6
```

در عبارت اول ابتدا به مقدار number یک واحد اضافه شده و ۴ می‌شود و سپس مقدار جدید با عدد ۳ جمع می‌شود و در نهایت عدد ۷ به دست می‌آید. در عبارت دوم مقدار عددی ۳ به مقدار number اضافه می‌شود و عدد ۶ به دست می‌آید. سپس این مقدار در متغیر number2 قرار می‌گیرد. و در نهایت مقدار number به ۴ افزایش می‌یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم:

```
number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ));
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال:

```
number = 3 * 2 + 8 / 4;
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابراین شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

گرفتن ورودی از کاربر

چارچوب دات‌نت تعدادی متد برای گرفتن ورودی از کاربر در اختیار شما قرار می‌دهد. حال می‌خواهیم در باره متد ReadLine() یکی دیگر از متدهای کلاس Console بحث کنیم که یک مقدار رشته‌ای را از کاربر دریافت می‌کند. متد ReadLine() فقط مقدار رشته‌ای را که توسط کاربر نوشته می‌شود را بر می‌گرداند. همانطور که از نام این متد پیداست، تمام کاراکترهایی را که شما در محیط کنسول تایپ می‌کنید تا زمانی که دکمه Enter را می‌زنید می‌خواند. هر چه که در محیط کنسول تایپ می‌شود از نوع رشته است. برای تبدیل نوع رشته به انواع دیگر می‌توانید از کلاس Convert و متدهای آن استفاده کنید. به برنامه زیر توجه کنید:

```
1  using System;
2
3  public class Program
4  {
5      public static void Main()
6      {
7          string name;
8          int age;
9          double height;
10
11         Console.Write("Enter your name: ");
12         name = Console.ReadLine();
13         Console.Write("Enter your age: ");
14         age = Convert.ToInt32(Console.ReadLine());
15         Console.Write("Enter your height: ");
16         height = Convert.ToDouble(Console.ReadLine());
17
18         //Print a blank line
19         Console.WriteLine();
20
21         //Show the details you typed
22         Console.WriteLine("Name is {0}.", name);
23         Console.WriteLine("Age is {0}.", age);
```

```
24          Console.WriteLine("Height is {0}.", height);
25      }
26  }
```

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John.
Age is 18.
Height is 160.5.
```

ابتدا ۳ متغیر را برای ذخیره داده در برنامه تعریف می‌کنیم (خطوط ۷ و ۸ و ۹). برنامه از کاربر می‌خواهد که نام خود را وارد کند (خط ۱۱). در خط ۱۲ شما به عنوان کاربر نام خود را وارد می‌کنید. مقدار متغیر نام، برابر مقداری است که توسط متد ReadLine() خوانده می‌شود. از آنجاییکه نام از نوع رشته است و مقداری که از متد ReadLine() خوانده می‌شود هم از نوع رشته است در نتیجه نیازی به تبدیل انواع نداریم.

سپس برنامه از ما سن را سؤال می‌کند (خط ۱۳). سن، متغیری از نوع صحیح (int) است، پس نیاز است که ما تبدیل از نوع رشته به صحیح را انجام دهیم. بنابراین از کلاس و متد Convert.ToInt32() برای این تبدیل استفاده می‌کنیم (خط ۱۴). مقدار بازگشتی از این متد در متغیر سن قرار می‌گیرد. چون متغیر قد (height) را از نوع double تعریف کرده‌ایم برای تبدیل رشته دریافتی از محیط کنسول به نوع double باید از متد ToDouble() مربوط به کلاس Convert استفاده کنیم (خط ۱۶). علاوه بر آنچه گفته شد شما می‌توانید از متد Parse() برای تبدیل‌های بالا استفاده کنید، مانند:

```
age = int.Parse(Console.ReadLine());
height = double.Parse(Console.ReadLine());
```

توجه داشته باشد که این متد برای تبدیل رشته به رقم استفاده می‌شود، یعنی رشته‌ای که توسط کاربر تایپ می‌شود، باید فقط عدد باشد.

ساختارهای تصمیم

تقریباً همه زبان‌های برنامه‌نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید در صورتیکه مقدار یک متغیر با یک عدد برابر باشد، سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. سی‌شارپ راه‌های مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالب زیر آشنا خواهید شد:

- دستور if
- دستور if...else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو

- عملگرهای منطقی
- دستور switch

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است، کد معینی را انجام بده. ساختار دستور if به صورت زیر است:

```
if(condition)
{
  code to execute;
}
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه ای است. می‌توان از عملگرهای مقایسه ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           //Declare a variable and set it a value less than 10
8           int number = 5;
9
10          //If the value of number is less than 10
11          if (number < 10)
12              Console.WriteLine("Hello World.");
13
14          //Change the value of a number to a value which
15          // is greater than 10
16          number = 15;
17
18          //If the value of number is greater than 10
19          if (number > 10)
20              Console.WriteLine("Goodbye World.");
21      }
22  }
```

```
Hello World.
Goodbye World.
```

در خط ۸ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۱۱ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است؟ یعنی ۵ کوچک‌تر از ۱۰ است؟ منطقی است که نتیجه مقایسه درست می‌باشد. بنابراین خط ۱۲ اجرا و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۶). وقتی به دومین دستور if در خط ۱۹ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۲۰). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت:

```
if (number > 10) Console.WriteLine("Goodbye World.");
```

شما می‌توانید چندین دستور را در داخل دستور if بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جزء بدنه دستور if هستند. نحوه تعریف چند دستور در داخل بدنه if به صورت زیر است:

```
if(condition)
{
    statement1;
    statement2;
        .
        .
        .
    statementN;
}
```

این هم یک مثال ساده:

```
if (x > 10)
{
  Console.WriteLine("x is greater than 10.");
  Console.WriteLine("This is still part of the if statement.");
}
```

در مثال بالا اگر مقدار x از ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از ۱۰ بزرگ‌تر نباشد مانند کد زیر:

```
if (x > 10)
Console.WriteLine("x is greater than 10.");
Console.WriteLine("This is still part of the if statement. (Really?)");
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم.

```
if (x > 10)
Console.WriteLine("x is greater than 10.");

Console.WriteLine("This is still part of the if statement. (Really?)");
```

می‌بینید که دستور دوم (خط ۳) در مثال بالا جزء دستور if نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچک‌تر است پس خط This is still part of the if statement (Really?) چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه if داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند. یکی از خطاهای معمول کسانی که برنامه‌نویسی را تازه شروع کرده‌اند قرار دادن سمیکالن در سمت راست پرانتز if است. به عنوان مثال:

```
if (x > 10);
Console.WriteLine("x is greater than 10");
```

به یاد داشته باشید که if یک مقایسه را انجام می‌دهد و دستور اجرایی نیست. بنابراین برنامه شما با یک خطای منطقی مواجه می‌شود. همیشه به یاد داشته باشید که قرار گرفتن سمیکالن در سمت راست پرانتز if به منزله این است که بلوک کد در اینجا به پایان رسیده است. مثالی دیگر در مورد دستور if:

```
using System;

public class Program
{
    public static void Main()
    {
        int firstNumber;
        int secondNumber;

        Console.Write("Enter a number: ");
        firstNumber = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter another number: ");
        secondNumber = Convert.ToInt32(Console.ReadLine());

        if (firstNumber == secondNumber)
        {
            Console.WriteLine("{0} == {1}", firstNumber, secondNumber);
        }
        if (firstNumber != secondNumber)
        {
            Console.WriteLine("{0} != {1}", firstNumber, secondNumber);
        }
        if (firstNumber < secondNumber)
        {
            Console.WriteLine("{0} < {1}", firstNumber, secondNumber);
        }
        if (firstNumber > secondNumber)
        {
            Console.WriteLine("{0} > {1}", firstNumber, secondNumber);
        }
        if (firstNumber <= secondNumber)
        {
            Console.WriteLine("{0} <= {1}", firstNumber, secondNumber);
        }
        if (firstNumber >= secondNumber)
        {
            Console.WriteLine("{0} >= {1}", firstNumber, secondNumber);
        }
    }
}
```

```
Enter a number: 2
Enter another number: 5
2 != 5
2 < 5
2 <= 5
Enter a number: 10
Enter another number: 3
10 != 3
10 > 3
10 >= 3
Enter a number: 5
Enter another number: 5
5 == 5
5 <= 5
5 >= 5
```

ما از عملگرهای مقایسه ای در دستور if استفاده کرده‌ایم. ابتدا دو عدد که قرار است با هم مقایسه شوند را به عنوان ورودی از کاربر می‌گیریم. اعداد با هم مقایسه می‌شوند و اگر شرط درست بود پیغامی چاپ می‌شود. به این نکته توجه داشته باشید که شرط‌ها مقادیر بولی هستند، بنابراین شما می‌توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور if استفاده کنید.

```
bool isNewMillenium = year == 2000;

if (isNewMillenium)
{
Console.WriteLine("Happy New Millenium!");
}
```

اگر مقدار year برابر ۲۰۰۰ باشد سپس حاصل عبارت در متغیر isNewMillenium ذخیره می‌شود. می‌توان از متغیر برای تشخیص کد اجرایی بدنه دستور if استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

دستور if...else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود یعنی اگر حالتی برقرار بود کارخاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود دستور دیگر اجرا شود باید از دستور if…else استفاده کنید. ساختار دستور if…else در زیر آمده است:

```
if(condition)
{
  code to execute if condition is true;
}
else
{
  code to execute if condition is false;
}
```

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if…else آمده است.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int number = 5;
8
9           //Test the condition
10          if (number < 10)
11          {
12              Console.WriteLine("The number is less than 10.");
13          }
14          else
15          {
16              Console.WriteLine("The number is either greater than or equal to 10.");
17          }
18
19          //Modify value of number
```

```
20          number = 15;
21
22          //Repeat the test to yield a different result
23          if (number < 10)
24          {
25              Console.WriteLine("The number is less than 10.");
26          }
27          else
28          {
29              Console.WriteLine("The number is either greater than or equal to 10.");
30          }
31      }
32  }
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

وقتی مقدار number از ۱۰ کمتر باشد کد داخل بلوک if اجرا می‌شود و اگر مقدار number را تغییر دهیم و به مقداری بزرگ‌تر از ۱۰ تغییر دهیم شرط نادرست می‌شود و کد داخل بلوک else اجرا می‌شود. مانند بلوک if نباید به آخر کلمه کلیدی else سمیکالن اضافه شود.

عملگر شرطی

عملگر شرطی (:?) در سی‌شارپ مانند دستور شرطی if…else عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی سی‌شارپ است که نیاز به سه عملوند دارد: شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
public class Program
{
    public static void Main()
    {
        string pet1 = "puppy";
        string pet2 = "kitten";
        string type1;
        string type2;

        type1 = (pet1 == "puppy")  ? "dog" : "cat";
        type2 = (pet2 == "kitten") ? "cat" : "dog";
    }
}
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می‌دهد. خط یک به صورت زیر ترجمه می‌شود: اگر مقدار pet1 برابر با puppy بود، مقدار dog را در type1 قرار بده در غیر اینصورت مقدار cat را type1 قرار بده. خط دو به صورت زیر ترجمه می‌شود: اگر مقدار pet2 برابر با kitten بود، مقدار cat را در type2 قرار بده در غیر اینصورت مقدار dog. حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```
if (pet1 == "puppy")
    type1 = "dog";
else
    type1 = "cat";
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید، چون خوانایی برنامه را پایین می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید:

```
if (condition)
{
   code to execute;
}
else
{
   if (condition)
   {
      code to execute;
   }
   else
   {
      if (condition)
      {
         code to execute;
      }
      else
      {
         code to execute;
      }
   }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده تر کنید:

```
if(condition)
{
  code to execute;
}
else if(condition)
{
  code to execute;
}
else if(condition)
{
  code to execute;
}
else
{
   code to execute;
}
```

حال که نحوه استفاده از دستور else if را یاد گرفتید باید بدانید که مانند else، else if نیز به دستور if وابسته است. دستور else if وقتی اجرا می‌شود که اولین دستور if اشتباه باشد. حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور else if را نشان می‌دهد:

```
using System;

public class Program
{
    public static void Main()
    {
        int choice;
```

```
        Console.WriteLine("What's your favorite color?");
        Console.WriteLine("[1] Black");
        Console.WriteLine("[2] White");
        Console.WriteLine("[3] Blue");
        Console.WriteLine("[4] Red");
        Console.WriteLine("[5] Yellow\n");

        Console.Write("Enter your choice: ");
        choice = Convert.ToInt32(Console.ReadLine());

        if (choice == 1)
        {
            Console.WriteLine("You might like my black t-shirt.");
        }
        else if (choice == 2)
        {
            Console.WriteLine("You might be a clean and tidy person.");
        }
        else if (choice == 3)
        {
            Console.WriteLine("You might be sad today.");
        }
        else if (choice == 4)
        {
            Console.WriteLine("You might be inlove right now.");
        }
        else if (choice == 5)
        {
            Console.WriteLine("Lemon might be your favorite fruit.");
        }
        else
        {
            Console.WriteLine("Sorry, your favorite color is not in the choices above.");
        }
    }
}
```

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 1
You might like my black t-shirt.
```

```
What's your favorite color?
[1] Black
[2] White
[3] Blue
[4] Red
[5] Yellow

Enter your choice: 999
Sorry, your favorite color is not in the choices above.
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغام‌های مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد، کد مربوط به بلوک else اجرا می‌شود.

دستور if تو در تو

می‌توان از دستور if تو در تو در سی‌شارپ استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if (condition)
{
    code to execute;

    if (condition)
    {
         code to execute;
    }
    else if (condition)
    {
        if (condition)
        {
           code to execute;
        }
    }
}
else
{
    if (condition)
    {
       code to execute;
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم:

```
using System;
public class Program
{
    public static void Main()
    {
        int age;
        string gender;

        Console.Write("Enter your age: ");
        age = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter your gender (male/female): ");

        gender = Console.ReadLine();

        if (age > 12)
        {
            if (age < 20)
            {
                if (gender == "male")
                {
                    Console.WriteLine("You are a teenage boy.");
                }
                else
                {
                    Console.WriteLine("You are a teenage girl.");
                }
            }
            else
            {
                Console.WriteLine("You are already an adult.");
            }
        }
```

```
34          else
35          {
36              Console.WriteLine("You are still too young.");
37          }
38      }
39  }
```

```
Enter your age: 18
Enter your gender: male
You are a teenage boy.
```

```
Enter your age: 12
Enter your gender: female
You are still too young.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا برنامه از شما درباره سنتان سؤال می‌کند (خط ۹). در خط ۱۲ درباره جنستان از شما سؤال می‌کند. سپس به اولین دستور `if` می‌رسد (خط ۱۶).

در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور `if` می‌شود در غیر اینصورت وارد بلوک `else` (خط ۳۴) مربوط به همین دستور `if` می‌شود. حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین `if` شده‌اید. در بدنه اولین `if` دو دستور `if` دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد شما وارد بدنه `if` دوم می‌شوید و اگر نباشد به قسمت `else` متناظر با آن می‌روید (خط ۲۹). دوباره فرض می‌کنیم که سن شما کمتر از ۲۰ باشد، در اینصورت وارد بدنه `if` دوم شده و با یک `if` دیگر مواجه می‌شوید (خط ۲۰). در اینجا جنسیت شما مورد بررسی قرار می‌گیرد که اگر برابر `"male"` باشد کدهای داخل بدنه سومین `if` اجرا می‌شود در غیر اینصورت قسمت `else` مربوط به این `if` اجرا می‌شود (خط ۲۴). پیشنهاد می‌شود که از `if` تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است:

عملگر	تلفظ	مثال	تأثیر
&&	And	`z = (x > 2) && (y < 10)`	مقدار Z در صورتی `true` است که هر دو شرط دو طرف عملگر مقدارشان `true` باشد. اگر فقط مقدار یکی از شروط `false` باشد مقدار z، `false` خواهد شد.
||	Or	`z = (x > 2) || (y < 10)`	مقدار Z در صورتی `true` است که یکی از دو شرط دو طرف عملگر مقدارشان `true` باشد. اگر هر دو شرط مقدارشان `false` باشد مقدار z، `false` خواهد شد.
!	Not	`z = !(x > 2)`	مقدار Z در صورتی `true` است که مقدار شرط `false` باشد و در صورتی `false` است که مقدار شرط `true` باشد.

به عنوان مثال جمله `z = (x > 2) && (y < 10)` را به این صورت بخوانید: "در صورتی مقدار z برابر true است که مقدار x بزرگ‌تر از ۲ و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی OR (||) تأثیر متفاوتی نسبت به عملگر منطقی AND (&&) دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند:

```
if ( (x == 1) && ( (y > 3) || z < 10) ) )
{
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرط‌ها استفاده می‌کنیم. در اینجا ابتدا عبارت `((z < 10) || (y > 3)` مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر AND با نتیجه `(x == 1)` مقایسه می‌شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int age;
8           string gender;
9
10          Console.Write("Enter your age: ");
11          age = Convert.ToInt32(Console.ReadLine());
12
13          Console.Write("Enter your gender (male/female): ");
14          gender = Console.ReadLine();
15
16          if (age > 12 && age < 20)
17          {
18              if (gender == "male")
19              {
20                  Console.WriteLine("You are a teenage boy.");
21              }
22              else
23              {
24                  Console.WriteLine("You are a teenage girl.");
25              }
26          }
27          else
28          {
29              Console.WriteLine("You are not a teenager.");
30          }
31      }
32  }
```

```
Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
```

```
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی AND را نشان می‌دهد (خط ۱۶). وقتی به دستور if می‌رسید (خط ۱۶) برنامه سن شما را چک می‌کند. اگر سن شما بزرگ‌تر از ۱۲ و کوچک‌تر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود. عملگر AND عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند. نکته مهم اینجاست که شما می‌توانید از عملگرهای & و | به عنوان عملگر بیتی استفاده کنید.

```
if (x == 2 & y == 3)
{
   //Some code here
}

if (x == 2 | y == 3)
{
   //Some code here
}
```

تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد، عملوند سمت چپ به وسیله عملگر بیتی AND(&) ارزیابی می‌شود. اگر شرط‌ها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی AND(&&) و OR (||) به جای عملگرهای بیتی AND(&) و OR(|) بهتر خواهد بود. یکی دیگر از عملگرهای منطقی عملگر NOT(!) است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (!(x == 2))
{
   Console.WriteLine("x is not equal to 2.");
}
```

اگر نتیجه عبارت x == 2 برابر false باشد عملگر! آن را True می‌کند.

دستور Switch

در سی‌شارپ ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور switch معادل دستور if تو در تو است با این تفاوت که در دستور switch متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغیر نیستند. در زیر نحوه استفاده از دستور switch آمده است:

```
switch (testVar)
{
   case compareVal1:
       code to execute if testVar == compareVa11;
       break;
   case compareVa12:
       code to execute if testVar == compareVa12;
       break;
   .
   .
   .
   case compareVa1N:
       code to execute if testVer == compareVa1N;
```

```
        break;
    default:
        code to execute if none of the values above match the testVar;
        break;
}
```

ابتدا یک مقدار در متغیر switch که در مثال بالا testVar است قرار می‌دهید. این مقدار با هر یک از عبارت‌های case داخل بلوک switch مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات case برابر بود، کد مربوط به آن case اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور case با کلمه کلیدی break تشخیص داده می‌شود که باعث می‌شود برنامه از دستور switch خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوفتد برنامه با خطا مواجه می‌شود. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات case برابر نباشد. دستور default اختیاری است و اگر از بدنه switch حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int choice;
8
9           Console.WriteLine("What's your favorite pet?");
10          Console.WriteLine("[1] Dog");
11          Console.WriteLine("[2] Cat");
12          Console.WriteLine("[3] Rabbit");
13          Console.WriteLine("[4] Turtle");
14          Console.WriteLine("[5] Fish");
15          Console.WriteLine("[6] Not in the choices");
16          Console.Write("\nEnter your choice: ");
17
18          choice = Convert.ToInt32(Console.ReadLine());
19
20          switch (choice)
21          {
22              case 1:
23                  Console.WriteLine("Your favorite pet is Dog.");
24                  break;
25              case 2:
26                  Console.WriteLine("Your favorite pet is Cat.");
27                  break;
28              case 3:
29                  Console.WriteLine("Your favorite pet is Rabbit.");
30                  break;
31              case 4:
32                  Console.WriteLine("Your favorite pet is Turtle.");
33                  break;
34              case 5:
35                  Console.WriteLine("Your favorite pet is Fish.");
36                  break;
37              case 6:
38                  Console.WriteLine("Your favorite pet is not in the choices.");
39                  break;
40              default:
41                  Console.WriteLine("You don't have a favorite pet.");
42                  break;
43          }
44      }
```

```
45  }
```

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.



What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر caseها برابر نبود دستور default اجرا می‌شود. یکی دیگر از ویژگی‌های دستور switch این است که شما می‌توانید از دو یا چند case برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار number، ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که caseها باید پشت سر هم نوشته شوند.

```
switch (number)
{
   case 1:
   case 2:
   case 3:
       Console.WriteLine("This code is shared by three values.");
       break;
}
```

همانطور که قبلاً ذکر شد دستور switch معادل دستور if تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان نوشت:

```
if (choice == 1)
    Console.WriteLine("Your favorite pet is Dog.");
else if (choice == 2)
    Console.WriteLine("Your favorite pet is Cat.");
else if (choice == 3)
    Console.WriteLine("Your favorite pet is Rabbit.");
else if (choice == 4)
   Console.WriteLine("Your favorite pet is Turtle.");
else if (choice == 5)
    Console.WriteLine("Your favorite pet is Fish.");
else if (choice == 6)
    Console.WriteLine("Your favorite pet is not in the choices.");
else
    Console.WriteLine("You don't have a favorite pet.");
```

کد بالا دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. حال از بین این دو دستور (switch و if else) کدامیک را انتخاب کنیم. از دستور switch موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلاً در مثال زیر هیچگاه از switch استفاده نکنید.

```
int myNumber = 5;
int x = 5;

switch (myNumber)
{
   case x:
       Console.WriteLine("Error, you can't use variables as a value" +
                                  " to be compared in a case statment.");
       break;
}
```

مشاهده می‌کنید که با اینکه مقدار x عدد ۵ است و به طور واضح با متغیر myNumber مقایسه شده است برنامه خطا می‌دهد چون x یک ثابت نیست بلکه یک متغیر است یا به زبان ساده تر، قابلیت تغییر را دارد. اگر بخواهید از x استفاده کنید و برنامه خطا ندهد باید از کلمه کلیدی const به صورت زیر استفاده کنید.

```
int myNumber = 5;
const int x = 5;

switch (myNumber)
{
 case x:
      Console.WriteLine("Error has been fixed!");
 break;
}
```

از کلمه کلیدی const برای ایجاد ثابت‌ها استفاده می‌شود. توجه کنید که بعد از تعریف یک ثابت نمی‌توان مقدار آن را در طول برنامه تغییر داد. به یاد داشته باشید که باید ثابت‌ها را حتماً مقداردهی کنید. دستور switch یک مقدار را با مقادیر Caseها مقایسه می‌کند و شما لازم نیست که به شکل زیر مقادیر را با هم مقایسه کنید:

```
switch (myNumber)
{
 case x > myNumber:
      Console.WriteLine("switch staments can't test if a value is less than " +
            "or greater than the other value.");
 break;
}
```

تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر:

```
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
Console.WriteLine("Hello World.");
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. برای نوشتن کدهای بالا استفاده از حلقه‌ها بهتر است. ساختارهای تکرار در سی‌شارپ عبارت‌اند از:

- while
- do while
- for

حلقه While

ابتدایی‌ترین ساختار تکرار در سی‌شارپ حلقه While است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار حلقه while به صورت زیر است:

```
while(condition)
{
   code to loop;
}
```

می‌بینید که ساختار while مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است می‌نویسیم اگر نتیجه درست یا true باشد سپس کدهای داخل بلوک while اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه while برسد هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه while اصلاح شوند. به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه while آمده است:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int counter = 1;
8
9           while (counter <= 10)
10          {
11              Console.WriteLine("Hello World!");
12              counter++;
13          }
14
15      }
16  }
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۷ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد. در خط ۹ حلقه While را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است.

همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۱۲). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد. اگر مقدار شمارنده ۱ بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت > از => استفاده شده است. اگر از علامت > استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون 10 > 10 نیست. اگر می‌خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
   //code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

حلقه do while

حلقه do while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه do while به صورت زیر است:

```
do
{
   code to repeat;
}
while(condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد، دستورات داخل بدنه اجرا نمی‌شوند. یکی از موارد برتری استفاده از حلقه do while نسبت به حلقه while زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید:

استفاده از while

```
//while version
Console.WriteLine("Enter a number greater than 10: ");
number = Convert.ToInt32(Console.ReadLine());

while (number < 10)
{
   Console.WriteLine("Enter a number greater than 10: ");
   number = Convert.ToInt32(Console.ReadLine());
}
```

استفاده از do while

```
//do while version

do
{
   Console.WriteLine("Enter a number greater than 10: ");
   number = Convert.ToInt32(Console.ReadLine());
} while (number < 10);
```

مشاهده می‌کنید که از کدهای کمتری در بدنه do while نسبت به while استفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد و فقط دارای چند خصوصیت اضافی است.

ساختار حلقه for به صورت زیر است:

```
for(initialization; condition; operation)
{
   code to repeat;
}
```

مقدار اولیه (initialization) اولین مقداری است که به شمارنده حلقه می‌دهیم. شمارنده فقط در داخل حلقه for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقه ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد. در زیر یک مثال از حلقه for آمده است:

```
using System;

namespace ForLoopDemo
{
    public class Program
    {
        public static void Main()
        {
```

```
            for (int i = 1; i <= 10; i++)
            {
                Console.WriteLine("Number " + i);
            }
        }
    }
}
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (i++) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته Number و سپس مقدار جاری i یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار i اضافه شده و مقدار i برابر ۲ می‌شود و بار دیگر i با عدد ۱۰ مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط true شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید:

```
for (int i = 10; i > 0; i--)
{
  //code omitted
}
```

کد بالا اعداد را از ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (--) برنامه‌ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید از چندین متغیر در ساختار حلقه for استفاده کنید.

```
for (int i = 1, y = 2; i < 10 && y > 20; i++, y -= 2)
{
//some code here
}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می‌کنید باید آنها را با استفاده از کاما از هم جدا کنید.

حلقه‌های تو در تو (Nested Loops)

سی‌شارپ به شما اجازه می‌دهد که از حلقه‌ها به صورت تو در تو استفاده کنید. اگر یک حلقه در داخل حلقه دیگر قرار بگیرد، به آن حلقه تو در تو گفته می‌شود. در این نوع حلقه‌ها، به ازای اجرای یک بار حلقه بیرونی، حلقه داخلی به طور کامل اجرا می‌شود. در زیر نحوه ایجاد حلقه تو در تو آمده است:

```
for (init; condition; increment)
{
   for (init; condition; increment)
   {
      //statement(s);
   }
   //statement(s);
}
```

```
while(condition)
{
   while(condition)
   {
      //statement(s);
   }
   //statement(s);
}
```

```
do
{
   //statement(s);
   do
   {
      //statement(s);
   }
   while(condition);

}
while(condition);
```

نکته ای که در مورد حلقه‌های تو در تو وجود دارد این است که می‌توان از یک نوع حلقه در داخل نوع دیگر استفاده کرد. مثلاً می‌توان از حلقه for در داخل حلقه while استفاده نمود. در مثال زیر نحوه استفاده از این حلقه‌ها ذکر شده است. فرض کنید که می‌خواهید یک مستطیل با ۳ سطر و ۵ ستون ایجاد کنید:

```
1   using System;
2
3   namespace NestedLoopsDemo
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               for (int i = 1; i <= 4; i++)
10              {
11                  for (int j = 1; j <= 5; j++)
```

```
                {
                    Console.Write(" * ");
                }
                Console.Write("\n");
            }
        }
    }
}
```

```
 * * * * *
 * * * * *
 * * * * *
 * * * * *
```

در کد بالا به ازای یک بار اجرای حلقه for اول (خط ۹)، حلقه for دوم (۱۴-۱۱) به طور کامل اجرا می‌شود. یعنی وقتی مقدار i برابر عدد ۱ می‌شود، علامت * توسط حلقه دوم ۵ بار چاپ می‌شود، وقتی i برابر ۲ می‌شود، دوباره علامت * پنج بار چاپ می‌شود و .... در کل منظور از دو حلقه for این است که در ۴ سطر علامت * در ۵ ستون چاپ شود یا ۴ سطر ایجاد شود و در هر سطر ۵ بار علامت * چاپ شود. خط ۱۵ هم برای ایجاد خط جدید است. یعنی وقتی حلقه داخلی به طور کامل اجرا شد، یک خط جدید ایجاد می‌شود و علامت‌های * در خطوط جدید چاپ می‌شوند. البته به جای این خط می‌توان Console.WriteLine(); را هم نوشت.

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد:

```
using System;

namespace BreakContinueDemo
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Demonstrating the use of break.\n");

            for (int x = 1; x < 10; x++)
            {
                if (x == 5)
                    break;

                Console.WriteLine("Number " + x);
            }

            Console.WriteLine("\nDemonstrating the use of continue.\n");

            for (int x = 1; x < 10; x++)
            {
                if (x == 5)
                    continue;

                Console.WriteLine("Number " + x);
            }
        }
    }
}
```

```
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
Number 9
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است. اگر به جای for از حلقه‌های while و do…while استفاده می‌شد، نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۱۱) آمده است وقتی که مقدار x به عدد ۵ رسید سپس دستور break اجرا می‌شود (خط ۱۲). حلقه بلافاصله متوقف می‌شود حتی اگر شرط 10 > x برقرار باشد. از طرف دیگر در خط ۲۲ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد (وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه‌ها

آرایه نوعی متغیر است که لیستی از آدرس‌های مجموعه‌ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است:

```
datatype[] arrayName = new datatype[length];
```

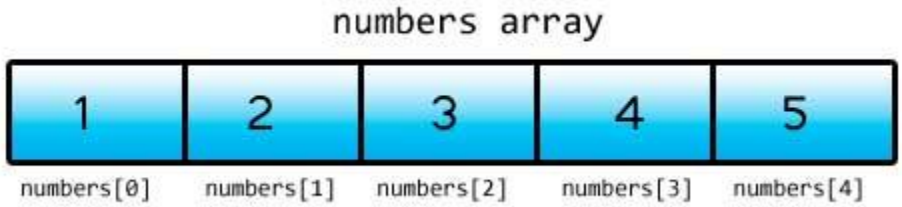
Datatype نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. کروشه که بعد از نوع داده قرار می‌گیرد و نشان دهنده استفاده از آرایه است. arrayName که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌هایی که اعداد را در خود ذخیره می‌کند از کلمه number استفاده کنید. طول آرایه که به کامپایلر می‌گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید. از کلمه کلیدی new هم برای اختصاص فضای حافظه به اندازه طول آرایه استفاده می‌شود. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم:

```
int[] numbers = new int[5];
```

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرس‌ها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آنها استفاده می‌شود.

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است، پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید:



به هر یک از اجزاء آرایه و اندیس‌های داخل کروشه توجه کنید. کسانی که تازه شروع به برنامه‌نویسی کرده‌اند، معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیس‌ها را از ۱ شروع کنند. اگر بخواهید به یکی از اجزای آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیس‌های آرایه شما نباشد با پیغام خطای IndexOutOfRangeException مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید که وجود ندارد. یکی دیگر از راه‌های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است:

```
datatype[] arrayName = new datatype[length] { val1, val2, ... valN };
```

در این روش شما می‌توانید فوراً بعد از تعریف اندازه آرایه مقادیر را در داخل آکولاد قرار دهید. به یاد داشته باشید که هر کدام از مقادیر را با استفاده از کاما از هم جدا کنید. همچنین تعداد مقادیر داخل آکولاد باید با اندازه آرایه تعریف شده برابر باشد. به مثال زیر توجه کنید:

```
int[] numbers = new int[5] { 1, 2, 3, 4, 5 };
```

این مثال با مثال قبل هیچ تفاوتی ندارد و تعداد خط‌های کدنویسی را کاهش می‌دهد. شما می‌توانید با استفاده از اندیس به مقدار هر یک از اجزاء آرایه دسترسی یابید و آنها را به دلخواه تغییر دهید. تعداد اجزاء آرایه در مثال بالا ۵ است و ما ۵ مقدار را در آن قرار می‌دهیم. اگر تعداد مقادیری که در آرایه قرار می‌دهیم کمتر یا بیشتر از طول آرایه باشد با خطا مواجه می‌شویم. یکی دیگر از راه‌های تعریف آرایه در زیر آمده است. شما می‌توانید هر تعداد عنصر را که خواستید در آرایه قرار دهید بدون اینکه اندازه آرایه را مشخص کنید. به عنوان مثال:

```
int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

در این مثال ما ۱۰ مقدار را به آرایه اختصاص داده‌ایم. نکته اینجاست که طول آرایه را تعریف نکرده‌ایم. در این حالت کامپایلر بعد از شمردن تعداد مقادیر داخل آکولاد طول آرایه را تشخیص می‌دهد. به این نکته توجه کنید که اگر برای آرایه طولی در نظر نگیرید، باید برای آن مقدار تعریف کنید، در غیر این صورت با خطا مواجه می‌شوید:

```
int[] numbers = new int[]; //not allowed
```

یک راه بسیار ساده تر برای تعریف آرایه به صورت زیر است:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

به سادگی و بدون احتیاج به کلمه کلیدی new می‌توان مقادیر را در داخل آکولاد قرار داد. کامپایلر به صورت اتوماتیک با شمارش مقادیر، طول آرایه را تشخیص می‌دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[] numbers = new int[5];
8           int total = 0;
9           double average;
10
11          for (int i = 0; i < numbers.Length; i++)
12          {
13              Console.Write("Enter a number: ");
14              numbers[i] = Convert.ToInt32(Console.ReadLine());
15          }
16
17          for (int i = 0; i < numbers.Length; i++)
18          {
19              total += numbers[i];
20          }
21
22          average = total / (double)numbers.Length;
23
24          Console.WriteLine("Average = {0}", average);
25      }
26  }
```

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط ۷ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۸ و ۹ متغیرهایی تعریف شده‌اند که از آنها برای محاسبه میانگین و جمع کل استفاده می‌شود. توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۱ تا ۱۵ حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از خاصیت طول (Length) آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه for مقدار ۵ را برای شرط قرار دهیم، ولی استفاده از خاصیت Length آرایه کار راحت تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ شود. در خط ۱۴ ورودی دریافت شده از کاربر به نوع int تبدیل و در آرایه ذخیره می‌شود. اندیس استفاده شده در number (خط ۱۴) مقدار i جاری در حلقه است. برای مثال در ابتدای حلقه

مقدار i صفر است، بنابراین وقتی در خط ۱۴ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر صفر می‌شود. در تکرار بعدی i یک واحد اضافه می‌شود و در نتیجه در خط ۱۴ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌یابد. در خطوط ۲۰-۱۷ از حلقه for دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۲۲). مقدار total یا جمع کل را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت length آرایه استفاده کرد. توجه کنید که در اینجا ما مقدار خاصیت length را به نوع double تبدیل کرده‌ایم، بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع double تبدیل نکنیم، نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۲۴ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغییر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جزء است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جزء تغییر اندازه دهید. البته تعداد خاصی از کلاس‌ها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید، بسیار مهم است.

حلقه foreach

حلقه foreach یکی دیگر از ساختارهای تکرار در سی‌شارپ می‌باشد که مخصوصاً برای آرایه‌ها، لیست‌ها و مجموعه‌ها طراحی شده است. حلقه foreach با هر بار گردش در بین اجزاء، مقادیر هر یک از آنها را در داخل یک متغیر موقتی قرار می‌دهد و شما می‌توانید بواسطه این متغیر به مقادیر دسترسی پیدا کنید. در زیر نحوه استفاده از حلقه foreach آمده است:

```
foreach (datatype temporaryVar in array)
{
   code to execute;
}
```

temporaryVar متغیری است که مقادیر اجزای آرایه را در خود نگهداری می‌کند.temporaryVar باید دارای نوع باشد تا بتواند مقادیر آرایه را در خود ذخیره کند. به عنوان مثال اگر آرایه شما دارای اعدادی از نوع صحیح باشد باید نوع متغیر موقتی از نوع اعداد صحیح باشد یا هر نوع دیگری که بتواند اعداد صحیح را در خود ذخیره کند مانند double یا long. سپس کلمه کلیدی in و بعد از آن نام آرایه را می‌نویسیم. در زیر نحوه استفاده از حلقه foreach آمده است:

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[] numbers = { 1, 2, 3, 4, 5 };
8
9           foreach (int n in numbers)
10          {
11              Console.WriteLine("Number {0}", n);
```

```
12          }
13      }
14  }
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
```

در برنامه بالا آرایه ای با ۵ جزء تعریف شده و مقادیر ۱ تا ۵ در آنها قرار داده شده است (خط ۷). در خط ۹ حلقه foreach شروع می‌شود. ما یک متغیر موقتی تعریف کرده‌ایم که اعداد آرایه را در خود ذخیره می‌کند. در هر بار تکرار از حلقه foreach متغیر موقتی n، مقادیر عددی را از آرایه استخراج می‌کند. حلقه foreach مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می‌دهد.

حلقه foreach برای دریافت هر یک از مقادیر آرایه کاربرد دارد. بعد از گرفتن مقدار یکی از اجزای آرایه، مقدار متغیر موقتی را چاپ می‌کنیم (خط ۱۱). حلقه foreach یک ضعف دارد و آن این است که این حلقه ما را قادر می‌سازد که به داده‌ها دسترسی یابیم و یا آنها را بخوانیم ولی اجازه اصلاح اجزاء آرایه را نمی‌دهد. برای درک این مطلب در مثال زیر سعی شده است که مقدار هر یک از اجزا آرایه یک واحد افزایش یابد:

```
int[] numbers = { 1, 2, 3 };

foreach(int number in numbers)
{
   number++;
}
```

اگر برنامه را اجرا کنید با خطا مواجه می‌شوید. برای اصلاح هر یک از اجزا آرایه می‌توان از حلقه for استفاده کرد.

```
int[] numbers = { 1, 2, 3 };

for (int i = 0; i < number.Length; i++)
{
   numbers[i]++;
}
```

آرایه‌های چند بعدی

آرایه‌های چند بعدی آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدادی ستون و ردیف تصور کنید. با افزایش اندیس‌ها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است:

```
datatype[,] arrayName = new datatype[lengthX, lengthY];
```

و یک آرایه سه بعدی به صورت زیر ایجاد می‌شود:

```
datatype[, ,] arrayName = new datatype[lengthX, lengthY, lengthZ];
```

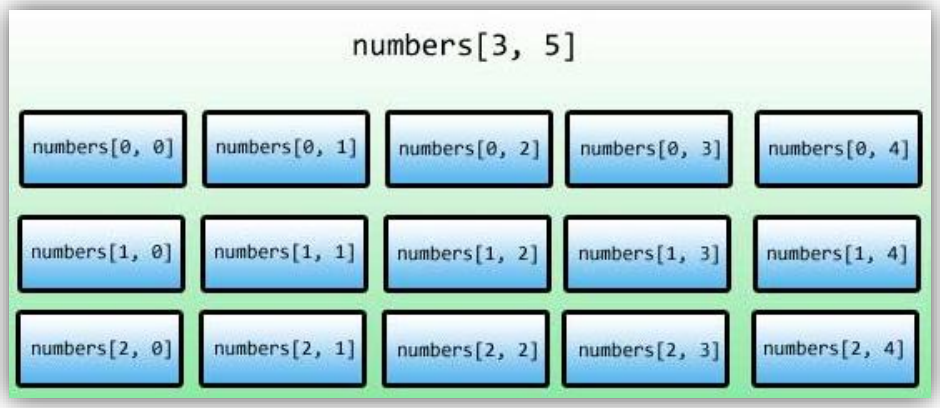
می‌توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه‌های سه بعدی یا آرایه‌های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می‌گیرند اجازه بدهید که در این درس بر روی آرایه‌های دو بعدی تمرکز کنیم. در تعریف این نوع

آرایه، ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می‌کند را مشخص می‌کنیم. سپس یک جفت کروشه و در داخل کروشه‌ها یک کاما قرار می‌دهیم.

به تعداد کاماهایی که در داخل کروشه می‌گذارید، توجه کنید. اگر آرایه ما دو بعدی است باید ۱ کاما و اگر سه بعدی است باید ۲ کاما قرار دهیم. سپس یک نام برای آرایه انتخاب کرده و بعد تعریف آنرا با گذاشتن کلمه new، نوع داده و طول آن کامل می‌کنیم. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار X نشان دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. یک آرایه سه بعدی را می‌توان به صورت یک مکعب تصور کرد که دارای سه بعد است و X طول، Y عرض و Z ارتفاع آن است. یک مثال از آرایه دو بعدی در زیر آمده است:

```
int[,] numbers = new int[3, 5];
```

کد بالا به کامپایلر می‌گوید که فضای کافی به عناصر آرایه اختصاص بده (در این مثال ۱۵ خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به x، چون ۳ سطر و مقدار ۵ را به Y چون ۵ ستون داریم، اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه‌ها وجود دارد.

```
datatype[,] arrayName = new datatype[x, y] { { r0c0, r0c1, ... r0cY },
                                             { r1c0, r1c1, ... r1cY },
                                                       .
                                                       .
                                                       .
                                             { rXc0, rXc1, ... rXcY } };
```

برای راحتی کار می‌توان از نوشتن قسمت new datatype[,] صرف نظر کرد.

```
datatype[,] arrayName = { { r0c0, r0c1, ... r0cY },
                          { r1c0, r1c1, ... r1cY },
                                   .
                                   .
                                   .
                          { rXc0, rXc1, ... rXcY } };
```

به عنوان مثال:

```
int[,] numbers = { { 1, 2, 3, 4, 5 },
                   { 6, 7, 8, 9, 10 },
                   { 11, 12, 13, 14, 15 } };
```

و یا می‌توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند:

```
array[0, 0] = value;
array[0, 1] = value;
array[0, 2] = value;
array[1, 0] = value;
array[1, 1] = value;
array[1, 2] = value;
array[2, 0] = value;
array[2, 1] = value;
array[2, 2] = value;
```

همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیس‌های X و Y و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه foreach و یا حلقه for تو در تو است.

اجازه دهید ابتدا از حلقه foreach استفاده کنیم.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
7           int[,] numbers = { { 1, 2, 3, 4, 5 },
8                              { 6, 7, 8, 9, 10 },
9                              { 11, 12, 13, 14, 15 }
10                           };
11
12          foreach (int number in numbers)
13          {
14              Console.Write(number + " ");
15          }
16      }
17  }
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

مشاهده کردید که گردش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. به وسیله حلقه foreach نمی‌توانیم انتهای ردیف‌ها را مشخص کنیم. برنامه زیر نشان می‌دهد که چطور از حلقه for برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید.

```
1   using System;
2
3   public class Program
4   {
5       public static void Main()
6       {
```

```
7           int[,] numbers = { { 1, 2, 3, 4, 5 },
8                              { 6, 7, 8, 9, 10 },
9                              { 11, 12, 13, 14, 15 }
10                           };
11
12          for (int row = 0; row < numbers.GetLength(0); row++)
13          {
14              for (int col = 0; col < numbers.GetLength(1); col++)
15              {
16                  Console.Write(numbers[row, col] + " ");
17              }
18
19              //Go to the next line
20              Console.WriteLine();
21          }
22      }
23  }
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه ساده for نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تو نیاز داریم. در اولین حلقه for (خط ۱۲) یک متغیر تعریف شده است که در میان ردیف‌های آرایه (row) گردش می‌کند. این حلقه تا زمانی ادامه می‌یابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از متد GetLength() کلاس Array استفاده کرده‌ایم. این متد طول آرایه را در یک بعد خاص نشان می‌دهد و دارای یک پارامتر است که همان بعد آرایه می‌باشد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه مقدار صفر را به این متد ارسال می‌کنیم چون شمارش ابعاد یک آرایه از صفر تا یک واحد کمتر از تعداد ابعاد انجام می‌شود.

در داخل اولین حلقه for حلقه for دیگری تعریف شده است (خط ۱۴). در این حلقه یک شمارنده برای شمارش تعداد ستون‌های (columns) هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از متد GetLength() استفاده شده است، ولی این بار مقدار ۱ را به آن ارسال می‌کنیم تا طول بعد دوم آرایه را به دست آوریم.

پس به عنوان مثال وقتی که مقدار ردیف (row) صفر باشد، حلقه دوم از [0, 0] تا [0, 4] اجرا می‌شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (col) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0, 0]) قرار دارد نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور Console.WriteLine() که به برنامه اطلاع می‌دهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند.

سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. این فرایند تا زمانی اجرا می‌شود که مقدار row کمتر از طول اولین بعد باشد. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```
1   using System;
2
```

```
3   public class Program
4   {
5       public static void Main()
6       {
7           double[,] studentGrades = new double[3, 4];
8           double total;
9
10          for (int student = 0; student < studentGrades.GetLength(0); student++)
11          {
12              total = 0;
13
14              Console.WriteLine("Enter grades for Student {0}", student + 1);
15
16              for (int grade = 0; grade < studentGrades.GetLength(1); grade++)
17              {
18                  Console.Write("Enter Grade #{0}: ", grade + 1);
19                  studentGrades[student, grade] = Convert.ToDouble(Console.ReadLine());
20                  total += studentGrades[student, grade];
21              }
22
23              Console.WriteLine("Average is {0:F2}", (total / studentGrades.GetLength(1)));
24              Console.WriteLine();
25          }
26      }
27  }
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
Average is 90.00
```

در برنامه بالا یک آرایه چند بعدی از نوع double تعریف شده است (خط ۷). همچنین یک متغیر به نام total تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه for تو در تو می‌شویم (خط ۱۰). در اولین حلقه for یک متغیر به نام sudent برای تشخیص پایه درسی هر دانش آموز تعریف کرده‌ایم. از متد GetLength() هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدنه حلقه for می‌شویم. در خط ۱۲ مقدار متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (student + 1). عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش Student 0، با Student 1 شروع شود، تا طبیعی تر به نظر برسد.

سپس به دومین حلقه for در خط ۱۶ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم که طول دومین بعد آرایه را با استفاده از فراخوانی متد GetLength(1) به دست می‌آورد. این طول تعداد نمراتی را که برنامه از سؤال می‌کند را نشان می‌دهد. برنامه چهار نمره

مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود. وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خطوط ۲۴-۲۳ معدل دانش آموز نشان داده می‌شود. به فرمت {0:F2} توجه کنید. این فرمت معدل را تا دو رقم اعشار نشان می‌دهد. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید. از متد GetLength(1) هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

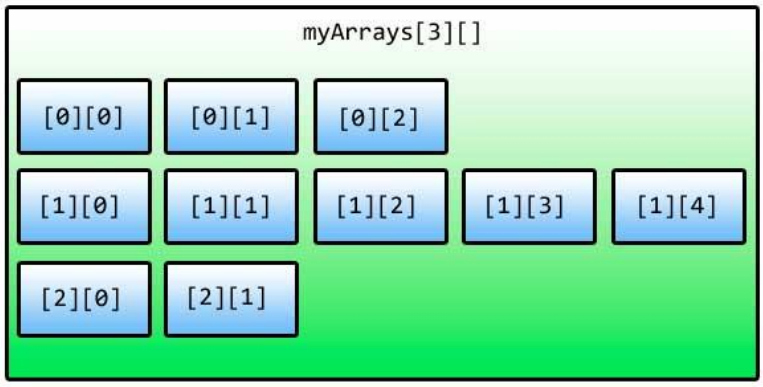
آرایه‌های دندانه دار

آرایه‌های دندانه دار نوعی از آرایه‌های چند بعدی هستند که شامل ردیف‌هایی با تعداد ستون‌های مختلف‌اند. آرایه چند بعدی ساده، آرایه ای به شکل مستطیل است، چون تعداد ستون‌های آن یکسان است ولی آرایه دندانه دار دارای سطرهایی با تعداد ستون‌های متفاوت است. بنابراین می‌توان یک آرایه دندانه دار را آرایه ای از آرایه‌ها فرض کرد. در زیر نحوه تعریف آرایه‌های چند بعدی آمده است.

```
datatype[][] arrayName;
```

مقدار دهی به آرایه‌های دندانه دار بسیار گیج کننده است. در زیر نحوه مقدار دهی به یک آرایه دندانه دار نشان داده شده است:

```
int[][] myArrays = new int[3][];

myArrays[0] = new int[3];
myArrays[1] = new int[5];
myArrays[2] = new int[2];
```



ابتدا تعداد ردیف‌های آرایه را به وسیله کلمه کلیدی new تعریف می‌کنیم، و بعد نوع داده‌ای آرایه و سپس دو جفت کروشه که در جفت کروشه اول تعداد ردیف‌ها قرار دارد را تعریف می‌کنیم. حال تعداد ستون‌های هر ردیف را با استفاده از سه ردیفی که در دسترس است و با استفاده از اندیس‌های آنها مانند یک آرایه ساده مقدار دهی می‌کنیم. می‌توان به ستون‌های هر ردیف مجموعه‌ای از مقادیر اختصاص داد:

```
int[][] myArrays = new int[3][];

myArrays[0] = new int[3] { 1, 2, 3 };
myArrays[1] = new int[5] { 5, 4, 3, 2, 1 };
myArrays[2] = new int[2] { 11, 22 };
```

یک راه بهتر برای مقداردهی آرایه‌های دندانه دار به شکل زیر است:

```
int[][] myArrays = new int[3][] { new int[3] { 1, 2, 3 },
                                  new int[5] { 5, 4, 3, 2, 1 },
                                  new int[2] { 11, 22 } };
```

همچنین می‌توان از ذکر طول ردیف‌های آرایه صرف نظر کرد:

```
int[][] myArrays = new int[][] { new int[] { 1, 2, 3 },
                                 new int[] { 5, 4, 3, 2, 1 },
                                 new int[] { 11, 22 } };
```

کد بالا را باز هم می‌توان ساده تر نوشت:

```
int[][] myArrays = { new int[] { 1, 2, 3 },
                     new int[] { 5, 4, 3, 2, 1 },
                     new int[] { 11, 22 } };
```

برای دسترسی به عناصر یک آرایه دندانه دار می‌توان از ستون‌ها و ردیف‌های آن استفاده کرد:

```
array[row][column]
```

```
Console.WriteLine(myArrays[1][2]);
```

از یک حلقه foreach ساده نمی‌توان برای دسترسی به اجزای این آرایه‌ها استفاده کرد.

```
foreach (int array in myArrays)
{
      Console.WriteLine(array);
}
```

اگر از حلقه foreach استفاده کنیم با خطا مواجه می‌شویم، چون عناصر این نوع آرایه‌ها، آرایه هستند نه عدد یا رشته یا... . برای حل این مشکل باید نوع متغیر موقتی (array) را تغییر داده و از حلقه foreach دیگری برای دسترسی به مقادیر استفاده کرد. مثال:

```
foreach (int[] array in myArrays)
{
    foreach (int number in array)
    {
         Console.WriteLine(number);
    }
}
```

این کار با استفاده از یک حلقه for تو در تو قابل اجراست:

```
for (int row = 0; row < myArray.Length; row++)
{
    for (int col = 0; col < myArray[row].Length; col++)
    {
        Console.WriteLine(myArray[row][col]);
    }
}
```

در اولین حلقه for با استفاده از خاصیت Length، myArray تعداد ردیف‌های آرایه را به دست می‌آوریم. در حلقه for دوم نیز با استفاده از خاصیت Length عنصر ردیف جاری تعداد ستون‌ها را به دست می‌آوریم. سپس با استفاده از اندیس، عناصر آرایه را چاپ می‌کنیم.

متدها

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می‌کنند. متد در داخل کلاس تعریف می‌شود. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در سی‌شارپ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند باید از کجا شروع شوند، این متد `Main()` نام دارد. پارامترها همان چیزهایی هستند که متد منتظر دریافت آنها است. آرگومانها مقادیری هستند که به پارامترها ارسال می‌شوند. گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک متد به صورت زیر است:

```
returnType MethodName()
{
   code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
1  using System;
2
3  public class Program
4  {
5      static void PrintMessage()
6      {
7          Console.WriteLine("Hello World!");
8      }
9
10     public static void Main()
11     {
12         PrintMessage();
13     }
14 }
```

```
Hello World!
```

در خطوط ۸-۵ یک متد تعریف کرده‌ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می‌توانید آن را زیر متد `Main()` تعریف کنید. می‌توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد `Main()` است که می‌توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده‌ایم (یعنی متد `PrintMessage()`) را صدا بزنیم. متد `Main()` به صورت `static` تعریف شده است. برای اینکه بتوان از متد `PrintMessage()` در داخل متد `Main()` استفاده کنیم، باید آن را به صورت `static` تعریف کنیم.

کلمه `static` به طور ساده به این معناست که می‌توان از متد استفاده کرد بدون اینکه از کلاس نمونه‌ای ساخته شود. متد `Main()` همواره باید به صورت `static` تعریف شود چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می‌کند. وقتی به مبحث برنامه‌نویسی شیء گرا رسیدید به طور دقیق کلمه `static` مورد بحث قرار می‌گیرد. برنامه `class` (مثال بالا) زمانی اجرا می‌شود که برنامه دو متدی را که تعریف کرده‌ایم را اجرا کند و متد `Main()` به صورت `static` تعریف شود. درباره این کلمه کلیدی در درس‌های آینده مطالب بیشتری می‌آموزیم. در تعریف متد بالا بعد از کلمه `static` کلمه کلیدی `void` آمده است که نشان دهنده آن است که متد مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک

متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد. نام متد ما PrintMessage() است. به این نکته توجه کنید که در نامگذاری متد از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کار آن متد را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor.

همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد، می‌توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید، مانند IsLeapyear یا IsTeenager ... . ولی از گذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درس‌های آینده در مورد متدها بیشتر توضیح می‌دهیم. بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متد را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم.

در داخل متد Main()، متدی که در خط ۱۲ ایجاد کرده‌ایم را صدا می‌زنیم. برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام، پرانتزها را قرار دهیم. اگر متد دارای پارامتر باشد باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درس‌های آینده توضیح بیشتری می‌دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متد PrintMessage() برنامه از متد Main() به محل تعریف متد PrintMessage() می‌رود. مثلاً وقتی ما متد PrintMessage() را در خط ۱۲ صدا می‌زنیم، برنامه از خط ۱۲ به خط ۷، یعنی جایی که متد تعریف شده می‌رود. اکنون ما یک متد در کلاس Program داریم که همه متدهای این کلاس می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش، سند را به شما تحویل دهد. سند همان مقدار برگشتی متد است. نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متد آسان است. کافیست در تعریف متد به روش زیر عمل کنید:

```
returnType MethodName()
{
   return value;
}
```

returnType در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (int، bool، ...). در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است، را می‌نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود. اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه void قبل از نام متد استفاده کنیم. مثال زیر یک متد که دارای مقدار برگشتی است را نشان می‌دهد.

```
1  using System;
2
3  public class Program
```

```
{
    static int CalculateSum()
    {
        int firstNumber = 10;
        int secondNumber = 5;

        int sum = firstNumber + secondNumber;

        return sum;
    }

    public static void Main()
    {
        int result = CalculateSum();

        Console.WriteLine("Sum is {0}.", result);
    }
}
```

```
Sum is 15.
```

همانطور که در خط ۵ مثال فوق مشاهده می‌کنید هنگام تعریف متد از کلمه int به جای void استفاده کرده‌ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۷ و ۸ دو متغیر تعریف و مقدار دهی شده‌اند. توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد Main() قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده‌اند قابل استفاده هستند. در خط ۱۰ جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط ۱۲ مقدار برگشتی sum توسط دستور return فراخوانی می‌شود.

در داخل متد Main() یک متغیر به نام result در خط ۱۷ تعریف می‌کنیم و متد CalculateSum() را فراخوانی می‌کنیم. متد CalculateSum() مقدار ۱۵ را بر می‌گرداند که این مقدار در داخل متغیر result ذخیره می‌شود. در خط ۱۹ مقدار ذخیره شده در متغیر result چاپ می‌شود. متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متد در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درس‌های آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک متد از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```
using System;

public class Program
{
    static int GetNumber()
    {
        int number;

        Console.Write("Enter a number greater than 10: ");
        number = Convert.ToInt32(Console.ReadLine());

        if (number > 10)
        {
            return number;
        }
        else
        {
            return 0;
```

```
        }
    }

    public static void Main()
    {
        int result = GetNumber();

        Console.WriteLine("Result = {0}.", result);
    }
}
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط 5-20 یک متد با نام GetNumber() تعریف شده است که از کاربر یک عدد بزرگ‌تر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متد مقدار صفر را بر می‌گرداند و اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم، در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم. چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار صفر را برگرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود، چون برنامه نیاز به مقدار برگشتی دارد، پیغام خطا می‌دهد. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک متد که مقدار برگشتی ندارد، خارج شوید. حتی اگر از نوع داده‌ای void در یک متد استفاده می‌کنید، باز هم می‌توانید کلمه کلیدی return را در آن به کار ببرید. استفاده از return باعث خروج از بدنه متد و اجرای کدهای بعد از آن می‌شود.

```
using System;

public class Program
{
    static void TestReturnExit()
    {
        Console.WriteLine("Line 1 inside the method TestReturnExit()");
        Console.WriteLine("Line 2 inside the method TestReturnExit()");

        return;

        //The following lines will not execute
        Console.WriteLine("Line 3 inside the method TestReturnExit()");
        Console.WriteLine("Line 4 inside the method TestReturnExit()");
    }

    public static void Main()
    {
        TestReturnExit();
        Console.WriteLine("Hello World!");
    }
}
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!
```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه، متد تعریف شده (TestReturnExit()) در داخل متد Main() فراخوانی و اجرا می‌شود.

پارامترها و آرگومانها

پارامترها داده‌های خامی هستند که متد آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید، که بر طبق آنها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است:

```
returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{
   code to execute;
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می‌گیرند. بر اساس کاری که متد انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تأمین کنید. آرگومانها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدهید که یک مثال بزنیم:

```
1   using System;
2
3   public class Program
4   {
5       static int CalculateSum(int number1, int number2)
6       {
7           return number1 + number2;
8       }
9
10      public static void Main()
11      {
12          int num1, num2;
13
14          Console.Write("Enter the first number: ");
15          num1 = Convert.ToInt32(Console.ReadLine());
16          Console.Write("Enter the second number: ");
17          num2 = Convert.ToInt32(Console.ReadLine());
18
19          Console.WriteLine("Sum = {0}", CalculateSum(num1, num2));
20      }
21  }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط ۸-۵) تعریف شده است، که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید int باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می‌کنند. در بدنه متد دستور return نتیجه جمع دو عدد را بر می‌گرداند. در داخل متد Main() برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال متد را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می‌شود.

حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد، چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی متد دقیقاً با ترتیب قرارگیری پارامترهای تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (بجز کلمه اول بقیه کلمات با حرف بزرگ شروع می‌شوند) نوشته می‌شود. در داخل بدنه متد (خط ۷) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدی که متد ( ) CalculateSum را فراخوانی می‌کند) ارسال می‌شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم، ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۷). در داخل متد ()Main از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود. در خط ۱۹ متد ()CalculateSum را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از متد به وسیله متد ()WriteLine از کلاس Console نمایش داده می‌شود (خط ۱۹). در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```
1   using System;
2
3   public class Program
4   {
5       static void ShowMessageAndNumber(string message, int number)
6       {
7           Console.WriteLine(message);
8           Console.WriteLine("Number = {0}", number);
9       }
10
11      public static void Main()
12      {
13          ShowMessageAndNumber("Hello World!", 100);
14      }
15  }
```

```
Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع int دریافت می‌کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۱۳ متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متد به صورت زیر فراخوانی می‌شد:

```
ShowMessageAndNumber(100, "Welcome to Gimme C#!");
```

در برنامه خطا به وجود می‌آمد، چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته‌ی Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شود. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است. به مثال ۱ توجه کنید. در آن مثال دو عدد از نوع int به پارامترها ارسال کردیم، که ترتیب ارسال آنها چون هر دو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متد دارای اهداف خاصی باشند، ترتیب ارسال آرگومانها مهم است.

```
void ShowPersonStats(int age, int height)
{
    Console.WriteLine("Age = {0}", age);
    Console.WriteLine("Height = {0}", height);
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);
```

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده‌ای قبول کند، باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید، چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم، سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود، که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
int MyMethod()
{
   return 5;
}

void AnotherMethod(int number)
{
   Console.WriteLine(number);
}

// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متد `MyMethod()` عدد ۵ است و به عنوان آرگومان به متد `AnotherMethod()` ارسال می‌شود خروجی کد بالا هم عدد ۵ است.

نامیدن آرگومانها

یکی دیگر از راه‌های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یادآوری و رعایت ترتیب پارامترها هنگام ارسال آرگومانها راحت می‌کند. در عوض شما باید نام پارامترهای متد را به خاطر بسپارید (ولی از آن جاییکه ویژوال استودیو `Intellisense` دارد نیازی به این کار نیست).

استفاده از نام آرگومانها خوانایی برنامه را بالا می‌برد، چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است. نامیدن آرگومانها در سی‌شارپ ۲۰۱۰ مطرح شده است و اگر شما از نسخه‌های قبلی مانند سی‌شارپ ۲۰۰۸ استفاده می‌کنید، نمی‌توانید از این خاصیت استفاده کنید. در زیر نحوه استفاده از نام آرگومانها وقتی که متد فراخوانی می‌شود، نشان داده شده است:

```
MethodToCall(paramName1: value, paramName2: value, ... paramNameN: value);
```

حال به مثال زیر توجه کنید:

```
using System;

public class Program
{
    static void SetSalaries(decimal jack, decimal andy, decimal mark)
    {
        Console.WriteLine("Jack's salary is {0:C}.", jack);
        Console.WriteLine("Andy's salary is {0:C}.", andy);
        Console.WriteLine("Mark's salary is {0:C}.", mark);
    }

    public static void Main()
    {
        SetSalaries(jack: 120, andy: 30, mark: 75);

        //Print a newline
        Console.WriteLine();

        SetSalaries(andy: 60, mark: 150, jack: 50);

        Console.WriteLine();

        SetSalaries(mark: 35, jack: 80, andy: 150);
    }
}
```

```
Jack' salary is $120.
Andy's salary is $30.
Mark's salary is $75.

Jack's salary is $50.
Andy's salary is $60.
Mark's salary is $150.

Jack's salary is $80.
Andy's salary is $150.
Mark's salary is $35.
```

متد `WriteLine()` در خطوط ۹-۷ از فرمت پول رایج، که با `{0:C}` نشان داده می‌شود، استفاده کرده است که یک داده عددی را به نوع پولی تبدیل می‌کند. خروجی نشان می‌دهد که حتی اگر ما ترتیب آرگومانها در سه متد فراخوانی شده را تغییر دهیم، مقادیر مناسب به پارامترهای مربوطه‌شان اختصاص داده می‌شود. همچنین می‌توان از آرگومانهای دارای نام و آرگومانهای ثابت (مقداری) به طور همزمان استفاده کرد، به شرطی که آرگومانهای ثابت قبل از آرگومانهای دارای نام قرار بگیرند.

```
SetSalary(30, andy: 50, mark: 60);


SetSalary(30, mark: 60, andy: 50);


SetSalary(mark: 60, andy: 50, 30);


SetSalary(mark: 60, 30, andy: 50);
```

همانطور که مشاهده می‌کنید ابتدا باید آرگومانهای ثابت هنگام فراخوانی متد ذکر شوند. در اولین و دومین فراخوانی در کد بالا، مقدار ۳۰ را به عنوان اولین آرگومان به اولین پارامتر متد یعنی jack اختصاص می‌دهیم. سومین و چهارمین خط کد بالا اشتباه هستند، چون آرگومانهای دارای نام قبل از آرگومانهای ثابت قرار گرفته‌اند. قرار گرفتن آرگومانهای دارای نام بعد از آرگومانها ثابت، از بروز خطا جلوگیری می‌کند.

ارسال آرگومانها به روش ارجاع

آرگومانها را می‌توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیر را ارسال می‌کنید، نه مقدار آن را. ارسال با ارجاع زمانی مفید است که شما بخواهید یک آرگومان که دارای مقدار بزرگی است (مانند یک آبجکت) را ارسال کنید. در این حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می‌کنیم، مقدار اصلی آرگومان در خارج از متد هم تغییر می‌کند. در زیر دستورالعمل پایه ای تعریف پارامترها که در آنها به جای مقدار از آدرس استفاده شده است نشان داده شده:

```
returnType MethodName(ref datatype param1)
{
   code to execute;
}
```

فراموش نشود که باید از کلمه کلیدی ref استفاده کنید. وقتی یک متد فراخوانی می‌شود و آرگومانها به آنها ارسال می‌شود هم باید از کلمه کلیدی ref استفاده شود.

```
MethodName(ref argument);
```

اجازه دهید که تفاوت بین ارسال با ارجاع و ارسال با مقدار آرگومان را با یک مثال توضیح دهیم.

```
1   using System;
2
3   public class Program
4   {
5       static void ModifyNumberVal(int number)
6       {
7           number += 10;
8           Console.WriteLine("Value of number inside method is {0}.", number);
9       }
10
11      static void ModifyNumberRef(ref int number)
12      {
13          number += 10;
14          Console.WriteLine("Value of number inside method is {0}.", number);
15      }
16
17      public static void Main()
18      {
19          int num = 5;
20
21          Console.WriteLine("num = {0}\n", num);
22
23          Console.WriteLine("Passing num by value to method ModifyNumberVal() ...");
24          ModifyNumberVal(num);
25          Console.WriteLine("Value of num after exiting the method is {0}.\n", num);
26
27          Console.WriteLine("Passing num by ref to method ModifyNumberRef() ...");
28          ModifyNumberRef(ref num);
29          Console.WriteLine("Value of num after exiting the method is {0}.\n", num);
30      }
```

```
31  }
```

```
num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.

Passing num by ref to method ModifyNumberRef() ...
Value of number inside method is 15.
Value of num after exiting the method is 15.
```

در برنامه بالا دو متد که دارای یک هدف یکسان هستند، تعریف شده‌اند و آن اضافه کردن عدد ۱۰ به مقداری است که به آنها ارسال می‌شود. اولین متد (خطوط ۹-۵) دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متد را صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط ۲۴)، کپی آرگومان به پارامتر متد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار ۱۰ را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم.

برای اثبات اینکه متغیر num هیچ تغییری نکرده است، مقدار آن را یکبار دیگر چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است. دومین متد (خطوط ۱۵-۱۱) نیاز به یک مقدار با ارجاع دارد. در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به آن ارسال شود، آدرس متغیر به آن ارسال می‌شود. حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می‌شود، دسترسی دارد. وقتی که ما مقدار پارامتری که شامل آدرس متغیر اصلی است را تغییر می‌دهیم (خط ۱۳)، در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده‌ایم. در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می‌دهیم و مشاهده می‌شود که مقدار آن واقعاً تغییر کرده است.

پارامترهای out

پارامترهای out، پارامترهایی هستند که متغیرهایی که مقدار دهی اولیه نشده‌اند، را قبول می‌کنند. کلمه کلیدی out زمانی مورد استفاده قرار می‌گیرد که، بخواهیم یک متغیر بدون مقدار را به متد ارسال کنیم. متغیر بدون مقدار اولیه، متغیری است که مقداری به آن اختصاص داده نشده است. در این حالت متد یک مقدار به متغیر می‌دهد. ارسال متغیر مقداردهی نشده به متد زمانی مفید است که شما بخواهید از طریق متد، متغیر را مقداردهی کنید. استفاده از کلمه کلیدی out باعث ارسال آرگومان به روش ارجاع می‌شود نه مقدار. به مثال زیر توجه کنید:

```
1   using System;
2
3   public class Program
4   {
5       static void GiveValue(out int number)
6       {
7           number = 10;
8       }
9
10      public static void Main()
11      {
12          //Uninitialized variable
13          int myNumber;
14
15          GiveValue(out myNumber);
16
17          Console.WriteLine("myNumber = {0}", myNumber);
18      }
```

```
19  }
```

```
myNumber = 10
```

از کلمه کلیدی out برای پارامترهای متد استفاده شده است، بنابراین می‌توانند متغیرهای مقداردهی نشده را قبول کنند. در متد Main()، خط ۱۵ متد را فراخوانی می‌کنیم و قبل از آرگومان کلمه کلیدی out را قرار می‌دهیم. متغیر مقداردهی نشده (myNumber) به متد ارسال می‌شود و در آنجا مقدار ۱۰ به آن اختصاص داده می‌شود (خط ۷). مقدار myNumber در خط ۱۷ نمایش داده می‌شود و مشاهده می‌کنید که مقدارش برابر مقداری است که در داخل متد به آن اختصاص داده شده است (یعنی ۱۰). استفاده از پارامترهای out بدین معنا نیست که شما همیشه نیاز دارید که آرگومانهای مقداردهی نشده را به متد ارسال کنید، بلکه آرگومانهایی که شامل مقدار هستند را هم می‌توان به متد ارسال کرد. این کار در حکم استفاده از کلمه کلیدی ref است. تفاوت ref با out این است که کلمه کلیدی ref به کامپایلر می‌گوید که متغیر مقدار دهی اولیه و بعد به متد ارسال شده است ولی out به کامپایلر می‌گوید که متغیر مقدار دهی اولیه نشده و باید در داخل متد مقدار دهی اولیه شود. زمانی که لازم باشد یک متد دارای چندین خروجی باشد از out استفاده می‌کنیم.

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```
1   using System;
2
3   namespace ArraysAsArgumentsDemo1
4   {
5       public class Program
6       {
7           static void TestArray(int[] numbers)
8           {
9               foreach (int number in numbers)
10              {
11                  Console.WriteLine(number);
12              }
13          }
14
15          public static void Main()
16          {
17              int[] array = { 1, 2, 3, 4, 5 };
18
19              TestArray(array);
20          }
21      }
22  }
```

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می‌توان با گذاشتن کروشه بعد از نوع داده‌ای پارامتر، یک متد ایجاد کرد که پارامتر آن، آرایه دریافت می‌کند. وقتی متد در خط ۱۹ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها هم به روش ارجاع به

متدها ارسال می‌شوند. در خطوط ۱۲-۹ از حلقه foreach برای دسترسی به اجزای اصلی آرایه که به عنوان آرگومان به متد ارسال کرده‌ایم، استفاده می‌کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```
using System;

namespace ArraysAsArgumentsDemo2
{
    public class Program
    {
        static void IncrementElements(int[] numbers)
        {
            for (int i = 0; i < numbers.Length; i++)
            {
                numbers[i]++;
            }
        }

        public static void Main()
        {
            int[] array = { 1, 2, 3, 4, 5 };

            IncrementElements(array);

            foreach (int num in array)
            {
                Console.WriteLine(num);
            }
        }
    }
}
```

```
2
3
4
5
6
```

برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. به این نکته توجه کنید که از حلقه foreach نمی‌توان برای افزایش مقادیر آرایه استفاده کنیم، چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آنها. در داخل متد، مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی متد هم اصلاح شده‌اند. راه دیگر برای ارسال آرایه به متد، مقداردهی مستقیم به متد فراخوانی شده است. به عنوان مثال:

```
IncrementElements(new int[] { 1, 2, 3, 4, 5 });
```

در این روش ما آرایه ای تعریف نمی‌کنیم، بلکه مجموعه‌ای از مقادیر را به پارامتر ارسال می‌کنیم، که آنها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه ای تعریف نکرده‌ایم، نمی‌توانیم در متد Main() نتیجه را چاپ کنیم. اگر از چندین پارامتر در متد استفاده می‌کنید، همیشه برای هر یک از پارامترهایی که آرایه قبول می‌کنند از یک جفت کروشه استفاده کنید. به عنوان مثال:

```
void MyMethod(int[] param1, int param2)
{
   //code here
}
```

به پارامترهای متد بالا توجه کنید. پارامتر اول (param1) آرگومانی از جنس آرایه قبول می‌کند ولی پارامتر دوم (param2) یک عدد صحیح. حال اگر پارامتر دوم (param2) هم آرایه قبول می‌کرد، باید برای آن هم از کروشه استفاده می‌کردیم:

```
void MyMethod(int[] param1, int[] param2)
{
  //code here
}
```

کلمه کلیدی params

کلمه کلیدی params امکان ارسال تعداد دلخواه پارامترهاي همنوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. کد زیر طریقه استفاده از کلمه کلیدی params را نشان می‌دهد:

```
using System;

public class Program
{
    static int CalculateSum(params int[] numbers)
    {
        int total = 0;

        foreach (int number in numbers)
        {
            total += number;
        }

        return total;
    }

    public static void Main()
    {
        Console.WriteLine("1 + 2 + 3 = {0}", CalculateSum(1, 2, 3));

        Console.WriteLine("1 + 2 + 3 + 4 = {0}", CalculateSum(1, 2, 3, 4));

        Console.WriteLine("1 + 2 + 3 + 4 + 5 = {0}", CalculateSum(1, 2, 3, 4, 5));

    }
}
```

```
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
```

از کلمه کلیدی params قبل از نوع داده‌ای آرایه پارامتر استفاده می‌شود (مثال بالا). حال متد را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم. این آرگومانها در داخل یک پارامتر از نوع آرایه ذخیره می‌شوند. با استفاده از حلقه foreach این آرگومانها را جمع و به متد فراخوان برگشت می‌دهیم.

وقتی از چندین پارامتر در یک متد استفاده می‌کنید، فقط یکی از آنها باید دارای کلمه کلیدی params بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر (پارامتری که دارای کلمه کلیدی params است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر params دار استفاده کنید با خطا مواجه می‌شوید. به مثال‌های اشتباه و درست زیر توجه کنید:

```
void SomeFunction(params int[] x, params int[] y) //ERROR
```

```
void SomeFunction(params int[] x, int y, int z)   //ERROR

void SomeFunction(int x, int y, params int[] z)   //Correct
```

محدوده متغیر

متدها در سی‌شارپ دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود، فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```
using System;

public class Program
{
    static void DemonstrateScope()
    {
        int number = 5;

        Console.WriteLine("number inside method DemonstrateScope() = {0}", number);
    }

    public static void Main()
    {
        int number = 10;

        DemonstrateScope();

        Console.WriteLine("number inside the Main method = {0}", number);
    }
}
```

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد Main() هیچ ارتباطی به متغیر داخل متد DemonstrateScope() ندارد. وقتی به مبحث کلاس‌ها رسیدیم در این باره بیشتر توضیح خواهیم داد.

پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست، اختیاری هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها، آرگومانی ارسال نشود از مقادیر پیش‌فرض استفاده می‌کنند. به مثال زیر توجه کنید:

```
1  using System;
2
3  public class Program
4  {
5      static void PrintMessage(string message = "Welcome to Visual C# Tutorials!")
6      {
7          Console.WriteLine(message);
8      }
9
```

```
10      public static void Main()
11      {
12          PrintMessage();
13
14          PrintMessage("Learn C# Today!");
15      }
16  }
```

```
Welcome to Visual C# Tutorials!
Learn C# Today!
```

متد PrintMessage() (خطوط ۸-۵) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (خط ۵). دو بار متد را فراخوانی می‌کنیم. در اولین فراخوانی (خط ۱۲) ما آرگومانی به متد ارسال نمی‌کنیم، بنابراین متد از مقدار پیش‌فرض (Welcome to Visual C# Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط ۱۴) یک پیغام (آرگومان) به متد ارسال می‌کنیم، که جایگزین مقدار پیش‌فرض پارامتر می‌شود. اگر از چندین پارامتر در متد استفاده می‌کنید همه پارامترهای اختیاری باید در آخر بقیه پارامترها ذکر شوند. به مثال‌های زیر توجه کنید.

```
void SomeMethod(int opt1 = 10, int opt2 = 20, int req1, int req2) //ERROR

void SomeMethod(int req1, int opt1 = 10, int req2, int opt2 = 20) //ERROR

void SomeMethod(int req1, int req2, int opt1 = 10, int opt2 = 20) //Correct
```

وقتی متدهای با چندین پارامتر اختیاری فراخوانی می‌شوند، باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد:

```
void SomeMethod(int required1, int optional1 = 10, int optional2 = 20)
{
   //Some Code
}

// ... Code omitted for demonstration

SomeMethod(10, , 100); //Error
```

اگر بخواهید از یک پارامتر اختیاری که در آخر پارامترهای دیگر نیست رد شوید و آن را نادیده بگیرید باید از نام پارامترها استفاده کنید.

```
SomeMethod(10, optional2: 100);
```

برای استفاده از نام پارامتر، شما به راحتی می‌توانید نام مخصوص پارامتر و بعد از نام علامت کالن (:) و بعد مقدار اختصاص شده به آن را بنویسید، مانند (optional2: 100). متد بالا هیچ آرگومانی برای پارامتر اختیاری optional1 ندارد، بنابراین این پارامتر از مقدار پیشفرضی که در زمان تعریف متد به آن اختصاص داده شده است، استفاده می‌کند.

سربارگذاری متدها

سربارگذاری متدها به شما اجازه می‌دهد که چندین متد با نام یکسان تعریف کنید که دارای امضاء و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومانهایی که شما به متد ارسال می‌کنید، به صورت خودکار تشخیص می‌دهد که کدام متد را فراخوانی کرده‌اید یا کدام متد مد نظر شماست. امضای یک متد نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید:

```
void MyMethod(int x, double y, string z)
```

که امضای متد بالا

```
MyMethod(int, double, string)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای متد نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری متدها آمده است.

```
1  using System;
2
3  namespace MethodOverloadingDemo
4  {
5      public class Program
6      {
7          static void ShowMessage(double number)
8          {
9              Console.WriteLine("Double version of the method was called.");
10         }
11
12         static void ShowMessage(int number)
13         {
14             Console.WriteLine("Integer version of the method was called.");
15         }
16
17         static void Main()
18         {
19             ShowMessage(9.99);
20             ShowMessage(9);
21         }
22     }
23 }
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده‌اند. اگر سربارگذاری متد توسط سی‌شارپ پشتیبانی نمی‌شد، برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می‌شوند، لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد همنام در صورتی فرق می‌گذارد، که پارامترهای متفاوتی داشته باشند. وقتی یک متد را فراخوانی می‌کنیم، متد نوع آرگومانها را تشخیص می‌دهد. در فراخوانی اول (خط ۱۹) ما یک مقدار double را به متد ShowMessage() ارسال کرده‌ایم، در نتیجه متد ShowMessage() (خطوط ۱۰-۷) که دارای پارامتری از نوع double است، اجرا می‌شود.

در بار دوم که متد فراخوانی می‌شود (خط ۲۰)، ما یک مقدار int را به متد ShowMessage() ارسال می‌کنیم. متد ShowMessage() (خطوط ۱۵-۱۲) که دارای پارامتری از نوع int است، اجرا می‌شود. معنای اصلی سربارگذاری متد همین است که توضیح داده شد. هدف اصلی از سربارگذاري

متدها این است، که بتوان چندین متد که وظیفه یکسانی انجام می‌دهند را تعریف کرد. تعداد زیادی از متدها در کلاس‌های دات‌نت سربارگذاری می‌شوند، مانند متد WriteLine() از کلاس Console. قبلاً مشاهده کردید که این متد می‌تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می‌تواند دو یا چند آرگومان قبول کند.

بازگشت

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه‌نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید، که بازگشت باید در یک نقطه متوقف شود، در غیر اینصورت برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچک‌تر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

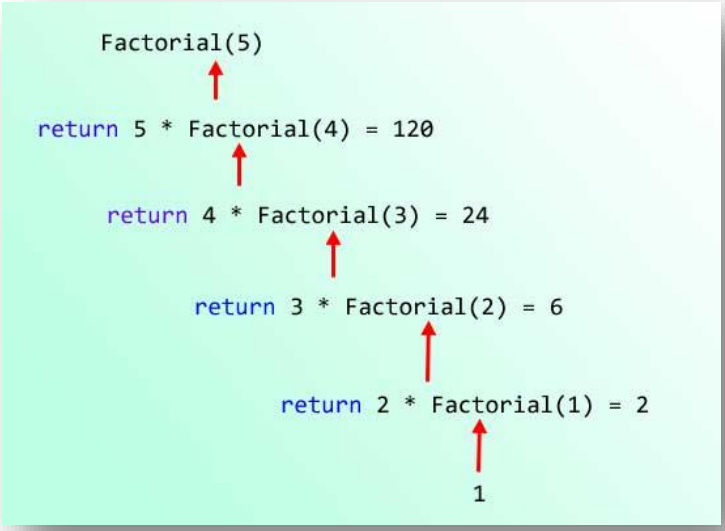
```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
1   using System;
2
3   public class Program
4   {
5       static long Factorial(int number)
6       {
7           if (number == 1)
8               return 1;
9
10          return number * Factorial(number - 1);
11      }
12
13      public static void Main()
14      {
15          Console.WriteLine(Factorial(5));
16      }
17  }
```

```
120
```

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور if می‌نویسیم و در خط ۷ می‌گوییم که اگر آرگومان ارسال شده برابر ۱ باشد، سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود. در خط ۱۰ مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می‌شود. در این خط متد Factorial()، خود را فراخوانی می‌کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number، ۱۰ باشد، یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم، آرگومان متد Factorial() در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان نوشت.

```
factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter;
```

این کد از کد معادل بازگشتی آن آسان تر است. از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت زمانی طبیعی تر به نظر می‌رسد که ما از غیر بازگشتی (Iteration) استفاده کنیم. استفاده از بازگشت حافظه زیادی اشغال می‌کند، پس اگر سرعت برای شما مهم است، از آن استفاده نکنید.

نماینده‌ها (Delegates)

Delegate ها انواعی هستند که مرجع یک متد را در خود ذخیره می‌کنند. همچنین می‌توانند رفتار هر متدی را کپی برداری کنند. برای تعریف یک delegate از کلمه کلیدی delegate استفاده می‌شود. تعریف یک delegate بسیار شبیه به تعریف یک متد است، با این تفاوت که متد بدنه دارد ولی delegate ندارد. Delegate دقیقاً مانند متدها دارای نوع برگشتی و مجموعه‌ای از پارامترها هستند. Delegate ها، می‌گویند که چه نوع متدی را می‌توانند در خود ذخیره کنند. در زیر نحوه تعریف delegate نشان داده شده است:

```
delegate returnType DelegateName(dt param1, dt param2, ... dt paramN);
```

در زیر نحوه استفاده از یک delegate و فواید آن نشان داده شده است:

```
1  using System;
2
3  public class Program
4  {
5      delegate void ArithmeticDelegate(int num1, int num2);
6
```

```
7       static void Add(int x, int y)
8       {
9           Console.WriteLine("Sum is {0}.", x + y);
10      }
11
12      static void Subtract(int x, int y)
13      {
14          Console.WriteLine("Difference is {0}.", x - y);
15      }
16
17      static void Main()
18      {
19          ArithmeticDelegate Operation;
20
21          int num1, num2;
22
23          Console.Write("Enter first number: ");
24          num1 = Convert.ToInt32(Console.ReadLine());
25
26          Console.Write("Enter second number: ");
27          num2 = Convert.ToInt32(Console.ReadLine());
28
29          if (num1 < num2)
30          {
31              Operation = new ArithmeticDelegate(Add);
32          }
33          else
34          {
35              Operation = new ArithmeticDelegate(Subtract);
36          }
37
38          Operation(num1, num2);
39      }
40  }
```

```
Enter first number: 3
Enter second number: 5
Sum is 8
Enter first number: 5
Enter second number: 3
Difference is 2
```

در خط ۵، delegate تعریف شده است. از کلمه کلیدی delegate برای نشان داده آن استفاده شده است. به دنبال آن نوع برگشتی متدی که قبول می‌کند، هم آمده است. برای نامگذاری delegate مانند متدها از روش Pascal استفاده می‌کنیم. همچنین برای تشخیص بهتر، بهتر است از کلمه delegate در نامگذاری آنها استفاده شود. پارامترهایی که برای delegate تعریف می‌کنیم، باید از نظر نوع و تعداد با پارامترهای متدها برابر باشد.

Delegate ی که در خط ۵ تعریف شده است، فقط مرجع متدهایی را قبول می‌کند که دارای مقدار برگشتی نیستند و دو پارامتر از نوع int دارند. بعد از تعریف delegate دو متد با امضای دقیقاً مشابه به عنوان نماینده تعریف می‌کنیم. هر دو متد هیچ مقدار برگشتی ندارند و هر دو، دو آرگومان از نوع int قبول می‌کنند. در داخل متد Main() یک متغیر از نوع delegate ی که قبلاً تعریف کرده‌ایم، تعریف می‌کنیم (خط ۱۹). این متغیر اشاره به متدی دارد که امضای آن با امضای Delegate مطابقت دارد. برنامه از کاربر می‌خواهد دو مقدار از نوع int را وارد کند. بعد از وارد کردن مقادیر، وارد اولین دستور if می‌شویم، چنانچه مقدار اولین عددی که کاربر وارد کرده از دومین عدد وارد شده کمتر باشد، دو عدد با هم جمع

می‌شوند، در غیر اینصورت اگر مقدار اولین عدد بزرگ‌تر یا مساوی دومین عدد باشد، از هم کم می‌شوند. برای ارجاع یک متد به یک delegate به صورت زیر عمل می‌کنیم:

```
variable = new DelegateName(MethodName);
```

وقتی یک delegate را با مرجع یک متد برابر قرار می‌دهیم، باید قبل از نام delegate از کلمه کلیدی new استفاده کنیم (مثال بالا). در داخل پرانتز نام متدی که delegate به آن مراجعه می‌کند، نشان داده شده است. یک راه بسیار ساده تر برابر قرار دادن نام متد با متغیر delegate است:

```
Operation = Add;
Operation = Subtract;
```

به دستور if بر می‌گردیم وقتی شرط درست باشد، delegate را به متد add() و هنگامی که شرط نادرست باشد آن را به متد Subtract() ارجاع می‌دهیم. اجرای delegate باعث اجرای متدی می‌شود که delegate به آن مراجعه می‌کند. اگر قصد داشته باشید که بیش از یک متد را به delegate اضافه کنید باید از عملگر += استفاده نمایید:

```
MyDelegate del = Method1;
del += Method2;
del += Method3;
...
```

کاربرد اصلی delegate ها هنگام کار با رویدادها می‌باشد که در درس‌های آینده توضیح می‌دهیم.

آرگومانهای خط فرمان (Command Line Arguments)

برای اجرای موفق یک برنامه سی‌شارپی باید یک متد مهم به نام متد Main() وجود داشته باشد، که نقطه آغاز برنامه است. این متد باید به صورت public static تعریف شود. همه ما می دانیم که به متدها می‌توان آرگومان ارسال کرد، اما برای متد Main(string[] args) چطور؟ جواب مثبت است. شما می‌توانید از طریق دستور خط فرمان ویندوز یا همان CMD آرگومانهایی را برای این متد ارسال کنید. برای روشن شدن مطلب یک برنامه کنسول به نام Sample ایجاد کنید، سپس کدهای برنامه را به صورت زیر بنویسید:

```
using System;

namespace Sample
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("First Name is " + args[0]);
            Console.WriteLine("Last  Name is " + args[1]);
            Console.ReadLine();
        }
    }
}
```

برنامه را یک بار اجرا و ذخیره کنید (ممکن است با پیغام خطا مواجه شوید ولی مهم نیست). به پارامتر args توجه کنید. در حقیقت این پارامتر یک آرایه رشته ای است که می‌تواند چندین آرگومان از نوع رشته قبول کند. اگر برنامه‌تان را ایجاد کرده و به فایل با پسوند .exe دسترسی داشته

باشید می‌توانید پارامترهای رشته ای را به متد Main() ارسال کنید. فایلSample.exe را که در پوشه Debug برنامه‌تان است را به یک درایو یا پوشه مشخص که مسیر گیج کننده ای نداشته باشد انتقال دهید. در این مثال ما فایل Sample.exe را مستقیماً در درایو C قرار می‌دهیم. حال CMD ویندوز را اجرا کنید، سپس کدهای زیر را در داخل CMD نوشته و دکمه Enter را بزنید:

```
Microsoft Windows[Version 6.1.7601]
Copyright(c) 2009 Microsoft Corporation.All rights reserved.

C:\Users\VisualCsharp>cd/

C:\>Sample Steven Clark
First Name is Steven
Last  Name is Clark
```

با نوشتن نام فایل، باعث اجرای آن می‌شویم. بعد از نوشتن نام فایل کلمه Steven و سپس Clark را می‌نویسیم. همانطور که در کد مشاهده می‌کنید ما دو متغیر به نام‌های args[0] و args[1] تعریف کرده‌ایم. این دو متغیر به ترتیب خانه‌های اول و دوم آرایه هستند. کلمه Stevenدر متغیر رشته ای args[0] که اولین عنصر آرایه و کلمه Clark را در متغیر رشته ای args[1] که دومین عنصر آرایه است ذخیره و سپس با استفاده از متد WriteLine() آنها را چاپ می‌کنیم. در حقیقت بسیاری از برنامه‌ها از این تکنیک استفاده می‌کنند. شما می‌توانید با ارسال آرگومانهایی به متد Main() نحوه اجرای برنامه را تغییر دهید.

شمارش (Enumeration)

شمارش راهی برای تعریف داده‌هایی است که می‌توانند مقادیر محدودی که شما از قبل تعریف کرده‌اید را بپذیرند. به عنوان مثال شما می‌خواهید یک متغیر تعریف کنید که فقط مقادیر جهت (جغرافیایی) مانند east، west، north و south را در خود ذخیره کند. ابتدا یک enumeration تعریف می‌کنید و برای آن یک اسم انتخاب کرده و بعد از آن تمام مقادیر ممکن که می‌توانند در داخل بدنه آن قرار بگیرند تعریف می‌کنید. به نحوه تعریف یک enumeration توجه کنید:

```
enum enumName
{
   value1,
   value2,
   value3,
   .
   .
   .
   valueN
}
```

ابتدا کلمه کلیدی enum و سپس نام آن را به کار می‌بریم. در سی‌شارپ برای نامگذاری enumeration از روش پاسکال استفاده کنید. در بدنه enum مقادیری وجود دارند که برای هر کدام یک نام در نظر گرفته شده است. به یک مثال توجه کنید:

```
enum Direction
{
    North,
    East,
    South,
    West
}
```

در حالت پیش‌فرض مقادیری که یک enumeration می‌تواند ذخیره کند از نوع int هستند. به عنوان مثال مقدار پیش‌فرض North صفر و مقدار بقیه مقادیر یک واحد بیشتر از مقدار قبلی خودشان است. بنابراین مقدار East برابر ۱، مقدار South برابر ۲ و مقدار West برابر ۳ است. می‌توانید این مقادیر پیش‌فرض را به دلخواه تغییر دهید، مانند:

```
enum Direction
{
    North = 3,
    East = 5,
    South = 7,
    West = 9
}
```

اگر به عنوان مثال هیچ مقداری به یک عنصر اختصاص ندهید، آن عنصر به صورت خودکار مقدار می‌گیرد.

```
enum Direction
{
    North = 3,
    East = 5,
    South,
    West
}
```

در مثال بالا مشاهده می‌کنید که ما هیچ مقداری برای South در نظر نگرفته‌ایم، بنابراین به صورت خودکار یک واحد بیشتر از East یعنی ۶ و به West یک واحد بیشتر از South یعنی ۷ اختصاص داده می‌شود. همچنین می‌توان مقادیر یکسانی برای عناصر enumeration در نظر گرفت. مثال:

```
enum Direction
{
    North = 3,
    East,
    South = North,
    West
}
```

می‌توانید مقادیر بالا را حدس بزنید؟ مقادیر North، East، South، West به ترتیب ۳، ۴، ۳، ۴ است. وقتی مقدار ۳ را به North می‌دهیم مقدار East برابر ۴ می‌شود. سپس وقتی مقدار South را برابر ۳ قرار دهیم، به صورت اتوماتیک مقدار West برابر ۴ می‌شود. اگر نمی‌خواهید که مقادیر آیتم‌های enumeration شما پیش‌فرض (از نوع int) باشد می‌توانید از نوع مثلاً byte به عنوان نوع داده‌ای آیتم‌های آن اسفاده کنید.

```
enum Direction : byte
{
    North,
    East,
    South,
    West
}
```

نوع داده‌ای byte فقط شامل مقادیر بین ۰ تا ۲۵۵ می‌شود بنابراین تعداد مقادیر که شما می‌توانید به enumeration اضافه کنید، محدود می‌باشد. به نحوه استفاده از enumeration در یک برنامه سی‌شارپ توجه کنید.

```
1  using System;
2
```

```
3   enum Direction
4   {
5       North = 1,
6       East,
7       South,
8       West
9   }
10
11  public class Program
12  {
13      public static void Main()
14      {
15          Direction myDirection;
16
17          myDirection = Direction.North;
18
19          Console.WriteLine("Direction: {0}", myDirection.ToString());
20      }
21  }
```

```
Direction: North
```

ابتدا enumeration را در خطوط ۹-۳ تعریف می‌کنیم. توجه کنید که enumeration را خارج از کلاس قرار داده‌ایم. این کار باعث می‌شود که enumeration در سراسر برنامه در دسترس باشد. می‌توان enumeration را در داخل کلاس هم تعریف کرد ولی در این صورت فقط در داخل کلاس قابل دسترس است.

```
class Program
{
    enum Direction
    {
        //Code omitted
    }

    static void Main(string[] args)
    {
        //Code omitted
    }
}
```

برنامه را ادامه می‌دهیم. در داخل بدنه enumeration نام چهار جهت جغرافیایی وجود دارد که هر یک از آنها با ۱ تا ۴ مقدار دهی شده‌اند. در خط ۱۵ یک متغیر تعریف شده است که مقدار یک جهت را در خود ذخیره می‌کند. نحوه تعریف آن به صورت زیر است:

```
enumType variableName ;
```

در اینجا enumType نوع داده شمارشی (مثلاً Direction یا مسیر) می‌باشد و variableName نیز نامی است که برای آن انتخاب کرده‌ایم که در مثال قبل myDirection است. سپس یک مقدار به متغیر myDirection اختصاص می‌دهیم (خط ۱۷). برای اختصاص یک مقدار به صورت زیر عمل می‌کنیم:

```
variable = enumType.value;
```

ابتدا نوع Enumeration سپس علامت نقطه و بعد مقدار آن (مثلاً North) را می‌نویسیم. می‌توان یک متغیر را فوراً، به روش زیر مقدار دهی کرد:

```
Direction myDirection = Direction.North;
```

حال در خط ۱۹ با استفاده از Console.WriteLine() مقدار myDirection را چاپ می‌کنیم. توجه کنید که با استفاده از متد ToString() مقدار عددی myDirection را به رشته، جهت چاپ تبدیل می‌کنیم. تصور کنید که اگر enumeration نبود شما مجبور بودید که به جای کلمات، اعداد را حفظ کنید چون مقادیر enumeration در واقع اعدادی هستند که با نام مستعار توسط شما یا هر کس دیگر تعریف می‌شوند. متغیرهای شمارشی می‌توانند به انواع دیگری مانند int یا string تبدیل شوند. همچنین یک مقدار رشته‌ای می‌تواند به نوع شمارشی معادلش تبدیل شود.

تبدیل انواع شمارشی

می‌توان انواع شمارشی را به دیگر مقادیر تبدیل کرد و بالعکس. مقادیر شمارشی در واقع مقادیر عددی هستند که برای درک بهتر آنها، به هر عدد یک نام اختصاص داده شده است. به مثال زیر توجه کنید:

```
1  using System;
2
3  enum Direction
4  {
5      North,
6      East,
7      South,
8      West
9  }
10 public class Program
11 {
12     public static void Main()
13     {
14         Direction myDirection = Direction.East;
15         int myDirectionCode = (int)myDirection;
16
17         Console.WriteLine("Value of East is {0}", myDirectionCode);
18
19         myDirection = (Direction)3;
20         Console.WriteLine("\nDirection: {0}", myDirection.ToString());
21     }
22 }
```

```
Value of East is 1

Direction: West
```

در خط ۱۴ مقدار East نوع شمارشی Direction را به متغیر myDirection را به اختصاص داده‌ایم. در حالت پیش‌فرض مقدار East در داخل آیتم‌های این داده شمارشی، ۱ می‌باشد. در خط ۱۵ نحوه تبدیل یک آیتم از نوع شمارشی به عدد صحیح معادل آن به روش تبدیل صریح نشان داده شده است. نحوه این تبدیل به صورت زیر است:

```
variable = (DestinationDataType)enumerationVariable;
```

از آنجاییکه متغیر myDirectionCode (خط ۱۵) از نوع int است در نتیجه یک مقدار int باید در آن قرار بگیرد. می‌توان به سادگی نوع داده مقصد را در داخل یک جفت پرانتز قرار داد و آن را کنار نوع شمارشی بگذارید (خط ۱۵). نتیجه یک مقدار تبدیل شده را برگشت می‌دهد. در خط ۱۹ معکوس این کار را انجام می‌دهیم. در این خط یک مقدار صحیح را به یک مقدار شمارشی تبدیل می‌کنیم. مقدار ۳ را برابر آیتم West قرار می‌دهیم.

برای تبدیل آن از روشی شبیه به تبدیل یک نوع شمارشی به صحیح استفاده می‌کنیم (تبدیل صریح). به این نکته توجه کنید که اگر عددی را که می‌خواهید تبدیل کنید در محدوده انواع شمارشی نباشد، تبدیل انجام می‌شود ولی آن آیتم شمارشی و عدد برابر هم نیستند. به عنوان مثال:

```
myDirection = (Direction)10;
Console.WriteLine("Direction: {0}", myDirection.ToString());
```

```
Direction: 10
```

از آنجاییکه عدد ۱۰ مقدار هیچ کدام از آیتم‌های نوع شمارشی مثال بالا نیست (مقدار آیتم‌های نوع شمارشی مثال بالا به ترتیب ۰ و ۱ و ۲ و ۳ می‌باشد) خروجی Console خود عدد را نشان می‌دهد. ولی اگر به جای عدد ۱۰ هر کدام از مقادیر عددی ذکر شده را قرار دهید، آیتم معادل با آن نمایش داده خواهد شد.

تبدیل یک نوع رشته‌ای به یک نوع شمارشی

می‌توان یک نوع رشته‌ای را به نوع شمارشی تبدیل کرد. مثلاً می‌خواهید رشته "West" را به نوع شمارشی Direction.West مثال بالا تبدیل کنید. برای این کار باید از کلاس Enum و فضای نام System به صورت زیر استفاده کنید:

```
Direction myDirection = (Direction)Enum.Parse(typeof(Direction), "West");
Console.WriteLine("Direction: {0}", myDirection.ToString());
```

```
Direction: West
```

متد Enum.Parse() دارای دو پارامتر است. اولین پارامتر نوع شمارشی است. با استفاده از عملگر typeof نوع شمارشی را برگشت می‌دهیم. دومین پارامتر، رشته‌ای است که قرار است به نوع شمارشی تبدیل شود. چون مقدار برگشتی از نوع شیء (object) است، بنابراین یک تبدیل مناسب نوع شمارشی لازم است. با این جزییات الان می‌دانیم که چگونه یک رشته را به نوع شمارشی تبدیل کنیم.

```
enumType name = (enumType)Enum.Parse(typeof(enumType), string);
```

اگر رشته‌ای که به متد ارسال می‌کنید جزء آیتم‌های داده شمارشی نباشد، با خطا مواجه می‌شوید.

ساختارها

ساختارها یا struct، انواعی از داده‌ها هستند که، توسط کاربر تعریف می‌شوند (user-define) و می‌توانند دارای فیلد و متد باشند. با ساختارها می‌توان نوع داده‌ای خیلی سفارشی ایجاد کرد. فرض کنید می‌خواهیم داده‌ای ایجاد کنیم که نه تنها نام شخص را ذخیره کند بلکه سن و حقوق ماهیانه او را نیز در خود جای دهد. برای تعریف یک ساختار به صورت زیر عمل می‌کنیم:

```
struct StructName
{
   member1;
   member2;
   member3;
   ...
```

```
    member4;
}
```

برای تعریف ساختار از کلمه کلیدی struct استفاده می‌شود. برای نامگذاری ساختارها از روش نامگذاری Pascal استفاده می‌شود. اعضاء در مثال بالا (member1-4) می‌توانند متغیر باشند یا متد. در زیر مثالی از یک ساختار آمده است:

```
using System;

public struct Employee
{
    public string name;
    public int age;
    public decimal salary;
}

public class Program
{
    public static void Main()
    {
        Employee employee1;
        Employee employee2;

        employee1.name = "Jack";
        employee1.age = 21;
        employee1.salary = 1000;

        employee2.name = "Mark";
        employee2.age = 23;
        employee2.salary = 800;

        Console.WriteLine("Employee 1 Details");
        Console.WriteLine("Name: {0}", employee1.name);
        Console.WriteLine("Age: {0}", employee1.age);
        Console.WriteLine("Salary: {0:C}", employee1.salary);

        Console.WriteLine(); //Seperator

        Console.WriteLine("Employee 2 Details");
        Console.WriteLine("Name: {0}", employee2.name);
        Console.WriteLine("Age: {0}", employee2.age);
        Console.WriteLine("Salary: {0:C}", employee2.salary);
    }
}
```

```
Employee 1 Details
Name: Jack
Age: 21
Salary: $1000.00

Employee 2 Datails
Name: Mike
Age: 23
Salary: $800.00
```

برای درک بهتر، کد بالا را شرح می‌دهیم. در خطوط ۸-۳ یک ساختار تعریف شده است. به کلمه Public در هنگام تعریف توجه کنید. این کلمه کلیدی نشان می‌دهد که Employee می‌تواند در هر جای برنامه قابل دسترسی و استفاده باشد، حتی خارج از برنامه. Public یکی از سطوح دسترسی است، که توضیحات بیشتر در مورد آن در درس‌های آینده آمده است. قبل از نام ساختار از کلمه کلیدی struct استفاده می‌کنیم. نام

ساختار نیز از روش نامگذاری Pascal پیروی می‌کند. در داخل بدنه ساختار سه فیلد تعریف کرده‌ایم (خطوط ۷-۵). این سه فیلد مشخصات Employee (کارمند) مان را نشان می‌دهند.

مثلاً یک کارمند دارای نام، سن و حقوق ماهانه می‌باشد. همچنین هر سه فیلد به صورت Public تعریف شده‌اند، بنابراین در خارج از ساختار نیز می‌توان آنها را فراخوانی کرد. در خطوط ۱۴ و ۱۵ دو نمونه از ساختار Employee تعریف شده است. تعریف یک نمونه از ساختارها بسیار شبیه به تعریف یک متغیر معمولی است. ابتدا نوع ساختار و سپس نام آن را مشخص می‌کنید. در خطوط ۱۷ تا ۲۳ به فیلدهای مربوط به هر employee مقادیری اختصاص می‌دهید. برای دسترسی به فیلدها در خارج از ساختار باید آنها را به صورت Public تعریف کنید. ابتدا نام متغیر را تایپ کرده و سپس علامت دات (.) و در آخر نام فیلد را می‌نویسیم. وقتی که از عملگر دات استفاده می‌کنیم، این عملگر اجازه دسترسی به اعضای مخصوص آن ساختار یا کلاس را به شما می‌دهد. در خطوط ۲۵ تا ۳۵ نشان داده شده که شما چطور می‌توانید به مقادیر ذخیره شده در هر فیلد دسترسی یابید.

ساختارها انواع مقداری هستند. این بدین معنی است که اگر مثلاً در مثال بالا employee2 را برابر employee1 قرار دهید، employee2 همه مقادیر صفات employee1 را به جای اینکه به آنها مراجعه کند، کپی برداری می‌کند. کلاس یک ساختار ساده است ولی از انواع مرجع به حساب می‌آید. در مورد کلاس در درس‌های آینده توضیح خواهیم داد. می‌توان به ساختار، متد هم اضافه کرد. مثال زیر اصلاح شده مثال قبل است.

```
1   using System;
2
3   public struct Employee
4   {
5       public string name;
6       public int age;
7       public decimal salary;
8
9       public void SayThanks()
10      {
11          Console.WriteLine("{0} thanked you!", name);
12      }
13  }
14
15  public class Program
16  {
17      public static void Main()
18      {
19          Employee employee1;
20          Employee employee2;
21
22          employee1.name = "Jack";
23          employee1.age = 21;
24          employee1.salary = 1000;
25
26          employee2.name = "Mark";
27          employee2.age = 23;
28          employee2.salary = 800;
29
30          Console.WriteLine("Employee 1 Details");
31          Console.WriteLine("Name: {0}", employee1.name);
32          Console.WriteLine("Age: {0}", employee1.age);
33          Console.WriteLine("Salary: {0:C}", employee1.salary);
34
35          employee1.SayThanks();
36
37          Console.WriteLine(); //Seperator
38
```

```
39          Console.WriteLine("Employee 2 Details");
40          Console.WriteLine("Name: {0}", employee2.name);
41          Console.WriteLine("Age: {0}", employee2.age);
42          Console.WriteLine("Salary: {0:C}", employee2.salary);
43
44          employee2.SayThanks();
45      }
46  }
```

```
Employee 1 Details
Name: Jack
Age: 21
Salary: $1000.00
Jack thanked you!

Employee 2 Details
Name: Mike
Age: 23
Salary: $800.00
Mike thanked you!
```

در خطوط ۹ تا ۱۲ یک متد در داخل ساختار تعریف شده است. این متد یک پیام را در صفحه نمایش نشان می‌دهد و مقدار فیلد name را گرفته و یک پیام منحصر به فرد برای هر نمونه نشان می‌دهد. برای فراخوانی متد، به جای اینکه بعد از علامت دات، نام فیلد را بنویسیم، نام متد را نوشته و بعد از آن همانطور که در مثال بالا مشاهده می‌کنید (خطوط ۳۵ و ۴۴) پرانتزها را قرار می‌دهیم و در صورتی که متد به آرگومان هم نیاز داشت در داخل پرانتز آنها را می‌نویسیم.

برنامه نویسی شیء گرا (Object Oriented Programming)

برنامه نویسی شیء گرا (OOP) شامل تعریف کلاس‌ها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است. برای مثال یک ماشین را در نظر بگیرید. این ماشین دارای خواصی مانند رنگ، سرعت، مدل، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است. اشیاء در سی‌شارپ تقلیدی از یک شیء مانند ماشین در دنیای واقعی هستند. برنامه‌نویسی شیء گرا با استفاده از کدهای دسته بندی شده کلاس‌ها و اشیاء را بیشتر قابل کنترل می‌کند. در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاء مان داریم. شیء در برنامه‌نویسی شیء گرا از روی کلاسی که شما تعریف کرده‌اید، ایجاد می‌شود. برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است. کلاس شامل خواص یک ساختمان مانند مساحت، بلندی و مواد مورد استفاده در ساخت خانه می‌باشد. در دنیای واقعی ساختمآنها نیز بر اساس یک نقشه (کلاس) پایه گذاری (تعریف) شده‌اند. برنامه‌نویسی شیء گرا یک روش جدید در برنامه‌نویسی است، که بوسیله برنامه نویسان مورد استفاده قرار می‌گیرد و به آنها کمک می‌کند که برنامه‌هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند. سی‌شارپ نیز یک برنامه شیء گراست و هر چیز در سی‌شارپ یک شیء است. در درس زیر به شما نحوه تعریف کلاس و استفاده از اشیاء آموزش داده خواهد شد. همچنین شما با دو مفهوم وراثت و چند ریختی که از مباحث مهم در برنامه‌نویسی شیء گرا هستند، در آینده آشنا می‌شوید.

کلاس

کلاس به شما اجازه می‌دهد یک نوع داده‌ای که توسط کاربر تعریف می‌شود و شامل فیلدها و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شیء می‌باشد. شیء یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می‌کند. وقتی یک شیء می‌سازید یعنی اینکه یک نمونه از کلاس ساخته‌اید (در درس ممکن است از کلمات شیء و نمونه به جای هم استفاده شود). ابتدا ممکن است فکر کنید که کلاس‌ها و ساختارها شبیه هم هستند. تفاوت مهم بین این دو این است که کلاس‌ها از نوع مرجع و ساختارها از نوع مقداری هستند.

در درس‌های آینده این موضوع شرح داده خواهد شد. اگر یادتان باشد در بخشهای اولیه این آموزش کلاسی به نام Program تعریف کردیم که شامل متد Main() بود و ذکر شد که این متد نقطه آغاز هر برنامه است. تعریف یک کلاس مشابه تعریف یک ساختار است. اما به جای استفاده از کلمه کلیدی struct باید از کلمه کلیدی class استفاده شود.

```
class ClassName
{
    field1;
    field2;
    ...
    fieldN;

    method1;
    method2;
    ...
```

```
    methodN;
}
```

این کلمه کلیدی را قبل از نامی که برای کلاسمان انتخاب می‌کنیم، می‌نویسیم. در نامگذاری کلاس‌ها هم از روش نامگذاری Pascal استفاده می‌کنیم. در بدنه کلاس فیلدها و متدهای آن قرار داده می‌شوند. فیلدها اعضای داده‌ای خصوصی هستند که کلاس از آنها برای رفتارها و ذخیره مقادیر خاصیت‌هایش (property) استفاده می‌کند. متدها، رفتارها یا کارهایی هستند که یک کلاس می‌تواند انجام دهد. در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام person نشان داده شده است.

```
1   using System;
2
3   public class Person
4   {
5       public string name;
6       public int age;
7       public double height;
8
9       public void TellInformation()
10      {
11          Console.WriteLine("Name: {0}", name);
12          Console.WriteLine("Age: {0} years old", age);
13          Console.WriteLine("Height: {0}cm", height);
14      }
15  }
16
17  public class Program
18  {
19      public static void Main()
20      {
21          Person firstPerson = new Person();
22          Person secondPerson = new Person();
23
24          firstPerson.name = "Jack";
25          firstPerson.age = 21;
26          firstPerson.height = 160;
27          firstPerson.TellInformation();
28
29          Console.WriteLine(); //Separator
30
31          secondPerson.name = "Mike";
32          secondPerson.age = 23;
33          secondPerson.height = 158;
34          secondPerson.TellInformation();
35      }
36  }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

برنامه بالا شامل دو کلاس Person و Program می‌باشد. می‌دانیم که کلاس Program شامل متد Main() است که برنامه برای اجرا به آن احتیاج دارد ولی اجازه دهید که بر روی کلاس Person تمرکز کنیم. در خطوط ۱۵-۳ کلاس Person تعریف شده است. در خط ۳ یک نام به کلاس اختصاص داده‌ایم تا به وسیله آن قابل دسترسی باشد. همچنین سطح دسترسی آن را public تعریف کرده‌ایم تا در دیگر کلاس‌ها قابل شناسایی باشد. درباره سطوح دسترسی در یک درس جداگانه بحث خواهیم کرد.

در داخل بدنه کلاس فیلدهای آن تعریف شده‌اند (خطوط ۷-۵). این سه فیلد، خصوصیات واقعی یک فرد در دنیای واقعی را در خود ذخیره می‌کنند. یک فرد در دنیای واقعی دارای نام، سن، و قد می‌باشد. در خطوط ۱۴-۹ یک متد هم در داخل کلاس به نام TellInformation() تعریف شده است که رفتار کلاسمان است و مثلاً اگر از فرد سوالی بپرسیم، در مورد خودش چیزهایی می‌گوید. در داخل متد کدهایی برای نشان دادن مقادیر موجود در فیلدها نوشته شده است. نکته‌ای درباره فیلدها وجود دارد و آن این است که، چون فیلدها در داخل کلاس تعریف و به عنوان اعضای کلاس در نظر گرفته شده‌اند، محدوده آنها یک کلاس است. این بدین معناست که فیلدها فقط می‌توانند در داخل کلاس یعنی جایی که به آن تعلق دارند و یا به وسیله نمونه ایجاد شده از کلاس، مورد استفاده قرار بگیرند. در داخل متد Main()، خطوط ۲۲-۲۱ دو نمونه یا دو شیء از کلاس Person ایجاد می‌کنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی new و به دنبال آن نام کلاس و یک جفت پرانتز قرار دهیم. وقتی نمونه کلاس ایجاد شد، سازنده را صدا می‌زنیم.

یک سازنده، متد خاصی است که برای مقداردهی اولیه به فیلدهای یک شیء به کار می‌رود. وقتی هیچ آرگومانی در داخل پرانتزها قرار ندهید، کلاس یک سازنده پیش‌فرض بدون پارامتر را فراخوانی می‌کند. درباره سازنده‌ها در درس‌های آینده توضیح خواهیم داد. در خطوط ۲۶-۲۴ مقادیری به فیلدهای اولین شیء ایجاد شده از کلاس Person (first Person) اختصاص داده شده است. برای دسترسی به فیلدها یا متدهای یک شیء، از علامت نقطه (دات) استفاده می‌شود. به عنوان مثال کد firstPerson.name نشان دهنده فیلد name از شیء firstPerson می‌باشد. برای چاپ مقادیر فیلدها باید متد TellInformation() شیء firstPerson را فراخوانی می‌کنیم.

در خطوط ۳۴-۳۱ نیز مقادیری به شیء دومی که قبلاً از کلاس ایجاد شده تخصیص می‌دهیم و سپس متد TellInformation() را فراخوانی می‌کنیم. به این نکته توجه کنید که firstPerson و secondPerson نسخه‌های متفاوتی از هر فیلد دارند، بنابراین تعیین یک نام برای secondPerson هیچ تاثیری بر نام firstPerson ندارد. در مورد اعضای کلاس در درس‌های آینده توضیح خواهیم داد.

سازنده (Constructor)

سازنده‌ها، متدهای خاصی هستند که وجود آنها برای ساخت اشیاء لازم است. آنها به شما اجازه می‌دهند که مقادیری را به هر یک از اعضای داده‌ای (فیلدها) اختصاص دهید و کدهایی که را که می‌خواهید هنگام ایجاد یک شیء اجرا شوند را به برنامه اضافه کنید. اگر از هیچ سازنده‌ای در کلاستان استفاده نکنید، کامپایلر از سازنده پیش‌فرض که یک سازنده بدون پارامتر است استفاده می‌کند. می‌توانید در برنامه‌تان از تعداد زیادی سازنده استفاده کنید که دارای پارامترهای متفاوتی باشند. در مثال زیر یک کلاس که شامل سازنده است را مشاهده می‌کنید.

```
1   using System;
2
3   namespace ConstructorsDemo
4   {
5       public class Person
6       {
7           public string name;
8           public int age;
9           public double height;
10
11          //Explicitly declare a default constructor
12          public Person()
13          {
14          }
```

```
15
16          //Constructor that has 3 parameters
17          public Person(string n, int a, double h)
18          {
19              name = n;
20              age = a;
21              height = h;
22          }
23
24          public void ShowInformation()
25          {
26              Console.WriteLine("Name: {0}", name);
27              Console.WriteLine("Age: {0} years old", age);
28              Console.WriteLine("Height: {0}cm", height);
29          }
30      }
31
32      public class Program
33      {
34          public static void Main()
35          {
36              Person firstPerson = new Person();
37              Person secondPerson = new Person("Mike", 23, 158);
38
39              firstPerson.name = "Jack";
40              firstPerson.age = 21;
41              firstPerson.height = 160;
42              firstPerson.ShowInformation();
43
44              Console.WriteLine(); //Seperator
45
46              secondPerson.ShowInformation();
47          }
48      }
49  }
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm
```

همانطور که مشاهده می‌کنید در مثال بالا دو سازنده را به کلاس Person اضافه کرده‌ایم. یکی از آنها سازنده پیش‌فرض (خطوط ۱۲-۱۰) و دیگری سازنده‌ای است که سه آرگومان قبول می‌کند (خطوط ۲۰- ۱۵). به این نکته توجه کنید که سازنده درست شبیه به یک متد است با این تفاوت که نه مقدار برگشتی دارد و نه از نوع void است. نام سازنده باید دقیقاً شبیه نام کلاس باشد. سازنده پیش‌فرض در داخل بدنه‌اش هیچ چیزی ندارد و وقتی فراخوانی می‌شود که ما از هیچ سازنده‌ای در کلاس مان استفاده نکنیم. در آینده متوجه می‌شوید که چطور می‌توان مقادیر پیشفرضی به اعضای داده‌ای اختصاص داد، وقتی که از یک سازنده پیش‌فرض استفاده می‌کنید. به دومین سازنده توجه کنید. نام آن شبیه نام سازنده اول است. سازنده‌ها نیز مانند متدها می‌توانند سربارگذاری شوند. حال اجازه دهید که یک سازنده خاص را هنگام تعریف یک نمونه از کلاس فراخوانی کنیم.

```
Person firstPerson  = new Person();
Person secondPerson = new Person("Mike", 23, 158);
```

در اولین نمونه ایجاد شده از کلاس Person از سازنده پیش‌فرض استفاده کرده‌ایم، چون پارامتری برای دریافت آرگومان ندارد. در دومین نمونه ایجاد شده، از سازنده‌ای استفاده می‌کنیم که دارای سه پارامتر است. کد زیر تأثیر استفاده از دو سازنده مختلف را نشان می‌دهد:

```
firstPerson.name = "Jack";
firstPerson.age = 21;
firstPerson.height = 160;
firstPerson.ShowInformation();

Console.WriteLine(); //Seperator

secondPerson.ShowInformation();
```

همانطور که مشاهده می‌کنید لازم است که به فیلدهای شیء ای که از سازنده پیش‌فرض استفاده می‌کند، مقادیری اختصاص داده شود تا این شیء نیز با فراخوانی متد ShowInformation() آنها را نمایش دهد. حال به شیء دوم که از سازنده دارای پارامتر استفاده می‌کند توجه کنید، مشاهده می‌کنید که با فراخوانی متد ShowInformation() همه چیز همانطور که انتظار می‌رود اجرا می‌شود. این بدین دلیل است که شما هنگام تعریف نمونه و از قبل، مقادیری به هر یک از فیلدها اختصاص داده‌اید بنابراین آنها نیاز به مقدار دهی مجدد ندارند، مگر اینکه بخواهید این مقادیر را اصلاح کنید.

اختصاص مقادیر پیش‌فرض به سازنده پیشفرض

در مثال‌های قبلی یک سازنده پیش‌فرض با بدنه خالی نشان داده شد. شما می‌توانید به بدنه این سازنده پیش‌فرض کدهایی اضافه کنید. همچنین می‌توانید مقادیر پیشفرضی به فیلدهای آن اختصاص دهید.

```
public Person()
{
   name   = "No Name";
   age    = 0;
   height = 0;
}
```

همانطور که در مثال بالا می‌بینید سازنده پیش‌فرض ما چیزی برای اجرا دارد. اگر نمونه‌ای ایجاد کنیم که از این سازنده پیش‌فرض استفاده کند، نمونه ایجاد شده مقادیر پیش‌فرض سازنده پیش‌فرض را نشان می‌دهد.

```
Person person1 = new Person();

person1.ShowInformation();
```

```
Name: No Name
Age: 0 years old
Height: 0cm
```

استفاده از کلمه کلیدی this

راهی دیگر برای ایجاد مقادیر پیش‌فرض، استفاده از کلمه کلیدی this است. مثال زیر اصلاح شده مثال قبل است و نحوه استفاده از چهار سازنده با تعداد پارامترهای مختلف را نشان می‌دهد.

```
1  using System;
2
3  public class Person
4  {
5      public string name;
6      public int age;
```

```
    public double height;

    public Person()
        : this("No Name", 0, 0)
    {
    }

    public Person(string n)
        : this(n, 0, 0)
    {
    }

    public Person(string n, int a)
        : this(n, a, 0)
    {
    }

    public Person(string n, int a, double h)
    {
        name = n;
        age = a;
        height = h;
    }

    public void ShowInformation()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("Age: {0} years old", age);
        Console.WriteLine("Height: {0}cm\n", height);
    }
}

public class Program
{
    public static void Main()
    {
        Person firstPerson  = new Person();
        Person secondPerson = new Person("Jack");
        Person thirdPerson  = new Person("Mike", 23);
        Person fourthPerson = new Person("Chris", 18, 152);

        firstPerson.ShowInformation();
        secondPerson.ShowInformation();
        thirdPerson.ShowInformation();
        fourthPerson.ShowInformation();
    }
}
```

```
Name: No Name
Age: 0 years old
Height: 0cm

Name: Jack
Age: 0 years old
Height: 0cm

Name: Mike
Age: 23 years old
Height: 0cm

Name: Chris
Age: 18 years old
Height: 152cm
```

ما چهار سازنده برای اصلاح کلاسمان تعریف کرده‌ایم. شما می‌توانید تعداد زیادی سازنده برای مواقع لزوم در کلاس داشته باشید. اولین سازنده یک سازنده پیش‌فرض است (خطوط ۱۲-۹). دومین سازنده یک پارامتر از نوع رشته دریافت می‌کند (خطوط ۱۷-۱۴). سومین سازنده دو پارامتر و چهارمین سازنده سه پارامتر می‌گیرد. به چهارمین سازنده در خطوط ۲۹-۲۴ توجه کنید. سه سازنده دیگر به این سازنده وابسته هستند. در خطوط ۱۲-۹ یک سازنده پیش‌فرض بدون پارامتر تعریف شده است. توجه کنید که از کلمه کلیدی this باید بعد از علامت کالن (:) استفاده کنید. این کلمه کلیدی به شما اجازه می‌دهد که یک سازنده دیگر موجود در داخل کلاس را فراخوانی کنید. مقادیر پیشفرضی به فیلدها، از طریق سازنده پیش‌فرض اختصاص می‌دهیم. چون ما سه آرگومان برای پرانتزهای بعد از کلمه کلیدی this سازنده پیش‌فرض در نظر گرفته‌ایم در نتیجه سازنده‌ای که دارای سه پارامتر است، فراخوانی شده و سه آرگومان به پارامترهای آن ارسال می‌شود.

کدهای داخل بدنه چهارمین سازنده اجرا می‌شوند و مقادیر پارامترهای آن به هر یک از اعضای داده‌ای مربوط اختصاص داده می‌شود. نمی‌توانیم هیچ کدی را در داخل بدنه اولین سازنده بنویسیم، چون توسط سازنده چهارم اجرا می‌شوند. اگر کدی در داخل بدنه سازنده پیش‌فرض بنویسیم، قبل از بقیه کدها اجرا می‌شود. دومین سازنده (خطوط ۱۷-۱۴) به یک آرگومان نیاز دارد که همان فیلد name کلاس Person است. وقتی این پارامتر با یک مقدار رشته‌ای پر شد، سپس به پارامترهای سازنده چهارم ارسال شده و در کنار دو مقدار پیش‌فرض دیگر (۰ برای age و ۰ برای height) قرار می‌گیرد. پس در داخل بدنه دومین سازنده هم نباید کدی بنویسیم، چون این کد نیز توسط سازنده چهارم اجرا می‌شود. در خط ۲۲-۹ سومین سازنده تعریف شده است که بسیار شبیه دومین سازنده است با این تفاوت که دو پارامتر دارد. مقدار دو پارامتر سومین سازنده به اضافه‌ی یک مقدار پیش‌فرض صفر برای سومین آرگومان، به چهارمین سازنده با استفاده از کلمه کلیدی this ارسال می‌شود.

```
Person firstPerson  = new Person();
Person secondPerson = new Person("Jack");
Person thirdPerson  = new Person("Mike", 23);
Person fourthPerson = new Person("Chris", 18, 152);
```

همانطور که مشاهده می‌کنید با ایجاد چندین سازنده برای یک کلاس، چندین راه برای ایجاد یک شیء بر اساس داده‌هایی که نیاز داریم به وجود می‌آید. در مثال بالا ۴ نمونه از کلاس Person ایجاد کرده‌ایم و چهار تغییر در سازنده آن به وجود آورده‌ایم. سپس مقادیر مربوط به فیلدهای هر نمونه را نمایش می‌دهیم. یکی از موارد استفاده از کلمه کلیدی this به صورت زیر است. فرض کنید نام پارامترهای متد کلاس شما یا سازنده، شبیه نام یکی از فیلدها باشد.

```
public Person(string name, int age, double height)
{
   name = name;
   age = age;
   height = height;
}
```

این نوع کدنویسی ابهام بر انگیز است و کامپایلر نمی‌تواند متغیر را تشخیص داده و مقداری به آن اختصاص دهد. اینجاست که از کلمه کلیدی this استفاده می‌کنیم.

```
public Person(string name, int age, double height)
{
   this.name = name;
   this.age = age;
   this.height = height;
```

```
}
```

قبل از هر فیلدی کلمه کلیدی this را می‌نویسیم و نشان می‌دهیم که این همان چیزی است که می‌خواهیم به آن مقداری اختصاص دهیم. کلمه کلیدی this ارجاع یک شیء به خودش را نشان می‌دهد.

مخرب (Destructor)

مخرب‌ها، نقطه مقابل سازنده‌ها هستند. مخرب‌ها، متدهای خاصی هستند که هنگام تخریب یک شیء فراخوانی می‌شوند. اشیاء از حافظه کامپیوتر استفاده می‌کنند و اگر پاک نشوند ممکن است با کمبود حافظه مواجه شوید. می‌توان از مخرب‌ها برای پاک کردن منابعی که در برنامه مورد استفاده قرار نمی‌گیرند، استفاده کرد. معمولاً دات‌نت فریم ورک به صورت اتوماتیک از زباله روب (garbage collection) برای پاک کردن حافظه استفاده می‌کند و لازم نیست شما به صورت دستی اشیاء را از حافظه پاک کنید. بعضی اوقات زباله روب کارش را به خوبی انجام نمی‌دهد. به عنوان مثال وقتی یک کانکشن برای پایگاه داده می‌سازید و یا یک فایل متنی را برای خواندن باز می‌کنید. با استفاده از مخرب‌ها، کدی را تعریف می‌کنید که وقتی اجرا می‌شود که یک شیء تخریب شده باشد. معمولاً شیء وقتی تخریب می‌شود که از محدوده خارج شود. دستور نوشتن مخرب کمی با سازنده متفاوت است:

```
~ClassName()
{
   code to execute;
}
```

مانند سازنده‌ها، مخرب‌ها باید همنام کلاسی باشند که در آن تعریف شده‌اند. به این نکته توجه کنید که قبل از نام مخرب علامت (~) را درج کنید. یک مخرب نمی‌تواند دارای سطح دسترسی (مانند public) باشد. برنامه زیر نحوه فراخوانی سازنده و مخرب را نشان می‌دهد:

```
using System;

public class Test
{
    public Test()
    {
        Console.WriteLine("Constructor was called.");
    }
    ~Test()
    {
        Console.WriteLine("Destructor was called.");
    }
}

public class Program
{
    public static void Main()
    {
        Test x1 = new Test();
    }
}
```

```
Constructor was called.
Destructor was called.
```

در کلاس Test یک سازنده و یک مخرب تعریف شده است. سپس در داخل متد Main() یک نمونه از کلاس ایجاد کرده‌ایم. وقتی یک نمونه از کلاس ایجاد می‌کنیم سازنده فراخوانی شده و پیغام مناسب نمایش داده می‌شود. وقتی از متد Main() خارج می‌شویم نمونه ایجاد شده نابود و مخرب فراخوانی می‌شود. دات نت فریم ورک در حقیقت مخرب را به عنوان یک متد که override شده متد Finalize() است، به صورت زیر تفسیر می‌کند :

```
protected override void Finalize()
{
    try
    {
        // Cleanup statements...
    }
    finally
    {
        base.Finalize();
    }
}
```

نگران کلماتی مانند Override، protect و ... نباشید. در درس‌های آینده در مورد آنها توضیح می‌دهیم.

فیلدهای فقط – خواندنی

از کلمه کلیدی readonly برای فیلدها استفاده می‌شود و اجازه تغییر مقادیر آنها را نمی‌دهد. فیلدهای فقط – خواندنی از لحاظ ساختاری بسیار شبیه به ثابت‌ها هستند با این تفاوت که در هنگام تعریف این نوع فیلدها می‌توانید به آنها مقداری اختصاص ندهید. هر چند باید به آنها در داخل سازنده کلاس مقادیری اختصاص دهید. بعد از تخصیص مقدار به یک فیلد فقط - خواندنی نمی‌توان آن را تغییر داد چون باعث بروز خطا می‌شود.

```
public class Sample
{
    readonly int y = 10;
    readonly int x;

    public Sample(int number)
    {
        x = number;
    }
}
```

در کلاس بالا دو فیلد فقط – خواندنی تعریف شده‌اند. اولین فیلد در هنگام تعریف مقدار گرفته است و دومین فیلد در سازنده کلاس.

سطح دسترسی (Scope)

سطح دسترسی مشخص می‌کند که متدها و فیلدهای یک کلاس در چه جای برنامه قابل دسترسی هستند. در این درس می‌خواهیم به سطح دسترسی Private و Public نگاهی بیندازیم. سطح دسترسی Public زمانی مورد استفاده قرار می‌گیرد که شما بخواهید به یک متد یا فیلد در خارج از کلاس و حتی پروژه دسترسی یابید. به عنوان مثال به کد زیر توجه کنید:

```
1  using System;
2
3  public class Test
4  {
5      public int number;
```

```
}

public class Program
{
    public static void Main()
    {
        Test x = new Test();

        x.number = 10;
    }
}
```

در این مثال کلاس test را به صورت public تعریف کرده‌ایم. این کار به کلاس program اجازه می‌دهد که از کلاس Test نمونه ایجاد کند. سپس یک عضو داده‌ای به صورت public در داخل کلاس Test تعریف می‌کنیم (خط ۵). با تعریف این عضو به صورت public می‌توانیم آن را در خارج از کلاس Test و در داخل متد ()Main کلاس Program مقدار دهی کنیم. اگر از کلمه کلیدی public استفاده نکنیم، نمی‌توانیم در داخل کلاس program نمونه‌ای از کلاس Test ایجاد کنیم و به اعضای آن دسترسی یابیم و این به نوعی به معنی استفاده از سطح دسترسی private می‌باشد.

```
using System;

private class Test
{
    public int number;
}

public class Program
{
    public static void Main()
    {
        Test x = new Test();

        x.number = 10;
    }
}
```

همانطور که در مثال بالا مشاهده می‌کنید این بار از کلمه private در کلاس Test استفاده کرده‌ایم (خط ۳). وقتی که برنامه را کامپایل می‌کنیم با خطا مواجه می‌شویم، چون کلاس Test در داخل کلاس Program و یا هر کلاس دیگر قابل دسترسی نیست. به این نکته توجه کنید که با اینکه عضو داده‌ای کلاس Test (یعنی number) به صورت Public تعریف شده است، باز هم نمی‌توان به آن در داخل کلاس Program دسترسی یافت، چون کلاسی که در داخل آن قرار دارد از نوع Private است در نتیجه تمام اعضای مربوطه در خارج از آن غیر قابل دسترسی هستند.

نکته دیگر اینکه اگر شما برای یک کلاس سطح دسترسی تعریف نکنید، آن کلاس دارای سطح دسترسی داخلی (internal) می‌شود، به این معنی که فقط کلاس‌های داخل پروژه‌ای که با آن کار می‌کنید، می‌توانند به آن کلاس دسترسی یابند. استفاده از سطح دسترسی internal معادل استفاده از سطح دسترسی public است با این تفاوت که در خارج از پروژه قابل دسترسی نیست. کدهای زیر دارای تأثیر یکسانی هستند.

```
class Test
{

}
```

```
internal class Test
{
```

```
}
```

اگر یک کلاس را به صورت public و اعضای آن را به صورت Private تعریف کنیم، آنگاه می‌توان یک نمونه از کلاس را در داخل کلاس‌های دیگر ایجاد کرد، ولی اعضای آن قابل دسترسی نیستند. اعضای داده‌ای private فقط به وسیله متد داخل کلاس Test قابل دسترسی هستند.

```
using System;

public class Test
{
    private int number;
}

public class Program
{
    public static void Main()
    {
        Test x = new Test();

        x.number = 10; //Error, number is private
    }
}
```

سطوح دسترسی دیگری هم در سی‌شارپ وجود دارد که بعد از مبحث وراثت در درس‌های آینده در مورد آنها توضیح خواهیم داد.

کپسوله سازی

کپسوله کردن (تلفیق داده‌ها با یکدیگر) یا مخفی کردن اطلاعات فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می‌شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می‌شود. وقتی که یک کلاس تعریف می‌کنیم، معمولاً تعدادی اعضای داده‌ای (فیلد) برای ذخیره مقادیر مربوط به شیء نیز تعریف می‌کنیم. برخی از این اعضای داده‌ای توسط خود کلاس برای عملکرد متدها و برخی دیگر از آنها به عنوان یک متغیر موقت به کار می‌روند. به این اعضای داده ای، اعضای مفید نیز می‌گویند چون فقط در عملکرد متدها تأثیر دارند و مانند یک داده قابل رویت کلاس نیستند. لازم نیست که کاربر به تمام اعضای داده‌ای یا متدهای کلاس دسترسی داشته باشد. اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی باشند بسیار خطرناک است، چون ممکن است کاربر رفتار و نتیجه یک متد را تغییر دهد. به برنامه ساده زیر توجه کنید:

```
using System;

public class Test
{
    public int five = 5;

    public int AddFive(int number)
    {
        number += five;
        return number;
    }
}

public class Program
{
```

```
    public static void Main()
    {
        Test x = new Test();

        x.five = 10;
        Console.WriteLine(x.AddFive(100));
    }
}
```

```
110
```

متد داخل کلاس Test به نام AddFive() دارای هدف ساده‌ای است و آن اضافه کردن مقدار ۵ به هر عدد می‌باشد. در داخل متد Main() یک نمونه از کلاس Test ایجاد کرده‌ایم و مقدار فیلد آن را از ۵ به ۱۰ تغییر می‌دهیم. حال قابلیت متد AddFive() به خوبی تغییر می‌کند و شما نتیجه متفاوتی مشاهده می‌کنید. اینجاست که اهمیت کپسوله سازی مشخص می‌شود. اینکه ما در درس‌های قبلی فیلدها را به صورت public تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد، کار اشتباهی بود. فیلدها باید همیشه به صورت private تعریف شوند.

خواص

property (خصوصیت)، استانداردی در سی‌شارپ برای دسترسی به اعضای داده ای (فیلدها) با سطح دسترسی private در داخل یک کلاس می‌باشد. همانطور که در درس قبل اشاره شد، تعریف فیلدها در داخل کلاس به صورت public اشتباه است، چون کاربران می‌توانند با ایجاد یک شیء از کلاس به آنها دسترسی داشته باشند و هر مقداری که دوست دارند به آنها اختصاص دهند. برای رفع این مشکل مفهوم property ارائه شد. هر property دارای دو بخش می‌باشد، یک بخش جهت مقدار دهی (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک داده private می‌باشد. Property ها باید به صورت public تعریف شوند تا در کلاسهای دیگر نیز قابل دسترسی می‌باشند. در مثال زیر نحوه تعریف و استفاده از property آمده است:

```
using System;

public class Person
{
    private string name;
    private int age;
    private double height;

    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }

    public int Age
    {
        get
        {
            return age;
```

```
        }
        set
        {
            age = value;
        }
    }

    public double Height
    {
        get
        {
            return height;
        }
        set
        {
            height = value;
        }
    }

    public Person(string name, int age, double height)
    {
        this.name = name;
        this.age = age;
        this.height = height;
    }

}

public class Program
{
    public static void Main()
    {
        Person person1 = new Person("Jack", 21, 160);
        Person person2 = new Person("Mike", 23, 158);

        Console.WriteLine("Name: {0}", person1.Name);
        Console.WriteLine("Age: {0} years old", person1.Age);
        Console.WriteLine("Height: {0}cm", person1.Height);

        Console.WriteLine(); //Seperator

        Console.WriteLine("Name: {0}", person2.Name);
        Console.WriteLine("Age: {0} years old", person2.Age);
        Console.WriteLine("Height: {0}cm", person2.Height);

        person1.Name = "Frank";
        person1.Age = 19;
        person1.Height = 162;

        person2.Name = "Ronald";
        person2.Age = 25;
        person2.Height = 174;

        Console.WriteLine(); //Seperator

        Console.WriteLine("Name: {0}", person1.Name);
        Console.WriteLine("Age: {0} years old", person1.Age);
        Console.WriteLine("Height: {0}cm", person1.Height);

        Console.WriteLine();

        Console.WriteLine("Name: {0}", person2.Name);
        Console.WriteLine("Age: {0} years old", person2.Age);
        Console.WriteLine("Height: {0}cm", person2.Height);
    }
}
```

```
Name: Jack
Age: 21 years old
Height: 160cm

Name: Mike
Age: 23 years old
Height: 158cm

Name: Frank
Age: 19 years old
Height: 162cm

Name: Ronald
Age: 25 years old
Height: 174cm
```

در برنامه بالا نحوه استفاده از property آمده است. همانطور که مشاهده می‌کنید، در این برنامه ما سه فیلد با سطح دسترسی private (خطوط ۷-۵) و برای این سه فیلد سه خاصیت با سطح دسترسی public تعریف کرده‌ایم (خطوط ۴۳-۹).

```
private string name;
private int age;
private double height;
```

دسترسی به مقادیر این فیلدها، فقط از طریق property های ارائه شده امکان پذیر است.

```
9       public string Name
10      {
11          get
12          {
13              return name;
14          }
15          set
16          {
17              name = value;
18          }
19      }
20
21      public int Age
22      {
23          get
24          {
25              return age;
26          }
27          set
28          {
29              age = value;
30          }
31      }
32
33      public double Height
34      {
35          get
36          {
37              return height;
38          }
39          set
40          {
41              height = value;
42          }
43      }
```

وقتی یک خاصیت ایجاد می‌کنیم، باید سطح دسترسی آن را public تعریف کرده و نوع داده ای را که بر می‌گرداند یا قبول می‌کند را مشخص کنیم. به این نکته توجه کنید که نام property ها همانند نام فیلدهای مربوطه می‌باشد با این تفاوت که حرف اول آنها بزرگ نوشته می‌شود. البته یادآور می‌شویم که شباهت نام property ها و فیلدها اجبار نیست و یک قرارداد در سی‌شارپ می‌باشد. در داخل بدنه دو بخش می‌بینید، یکی بخش set و دیگری بخش get. بخش set، که با کلمه کلیدی set نشان داده شده است برای مقدار دهی به فیلدها (اعضای داده ای) به کار می‌رود. بخش get، که با کلمه کلیدی get نشان داده شده است، به شما اجازه می‌دهد که یک مقدار را از فیلدها (اعضای داده ای) استخراج کنید. به کلمه کلیدی value در داخل بلوک set توجه کنید. Value، همان مقداری است که از طریق property به فیلد اختصاص می‌دهیم.

برای اختصاص یک مقدار به یک فیلد از طریق property کافیست که به صورت زیر عمل کنید:

```
Object.Property = Value;
```

این کار (قرار دادن یک مقدار بعد از علامت مساوی) به منزله فراخوانی بخش set است، و ما به برنامه می‌فهمانیم که می‌خواهیم از طریق بخش set یک فیلد را مقداردهی کنیم. Object، شیء ایجاد شده از کلاس، Property نام پراپرتی و Value مقداری است که می‌خواهیم به فیلد اختصاص دهیم. برای دسترسی به یک خاصیت می‌توانید از علامت دات (.) استفاده کنید. مثلاً برای اختصاص مقدار به سه فیلد age، name و height از طریق property باید به صورت زیر عمل کنید:

```
person1.Name = "Frank";
person1.Age = 19;
person1.Height = 162;
```

دستورات بالا بخش set مربوط به هر property را فراخوانی کرده و مقادیری به هر یک از فیلدها اختصاص می‌دهد. برای فراخوانی بخش get کافیست که نام شیء و سپس علامت نقطه و در آخر نام property را بنویسیم. با این کار به برنامه می‌فهمانیم که ما نیاز به مقدار فیلد داریم.

```
Console.WriteLine("Name: {0}", person1.Name);
Console.WriteLine("Age: {0} years old", person1.Age);
Console.WriteLine("Height: {0}cm", person1.Height));
```

به این نکته توجه کنید که در بخش get هم می‌توان تغییراتی بر روی فیلدها اعمال کرد. مثلاً فرض کنید که یک فیلد دارید که مقادیر پولی را در خود ذخیره می‌کند. شما می‌توانید در بخش get نحوه نمایش مقدار موجود در این فیلد را مشخص کنید. مثلاً خروجی به صورت سه رقم سه رقم نمایش داده شود. استفاده از property ها، کد نویسی را انعطاف پذیر می‌کند. مخصوصاً اگر بخواهید یک اعتبارسنجی برای اختصاص یک مقدار به فیلدها یا استخراج یک مقدار از آنها ایجاد کنید.

پس می‌توان گفت که، کاربرد اصلی property ها، اعتبار سنجی مقادیری است که، کاربر می‌خواهد به فیلدها اختصاص دهد.

مثلاً شما می‌توانید یک محدودیت ایجاد کنید که فقط اعداد مثبت به فیلد age (سن) اختصاص داده شود. همانطور که در کد ابتدای درس مشاهده می‌کنید ما نوع فیلد age را int قرار داده‌ایم. یعنی کاربر می‌تواند هر رقمی بین اعداد ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را به این فیلد اختصاص دهد. ولی چون غیر معقولانه است و سن (age) باید یک عدد مثبت و از لحاظ عقلی عددی از ۱ تا ۱۰۰ باشد، می‌توانیم کاربر را با

استفاده از بخش set مجبور کنیم که رقمی بین این دو عدد را به age اختصاص دهد. می‌توانید با تغییر بخش set خاصیت Age این کار را انجام دهید:

```
public int Age
{
    get
    {
        return age;
    }
    set
    {
        if (value > 0 && value <= 100)
        age = value;
        else
        age = 0;
    }
}
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد age اختصاص دهد، مقدار age صفر خواهد شد. همچنین می‌توان یک property فقط خواندنی (read-only) ایجاد کرد. این property فاقد بخش set است. به عنوان مثال می‌توان یک خاصیت Name فقط خواندنی مانند زیر ایجاد کرد :

```
public string Name
{
    get
    {
        return name;
    }
}
```

در این مورد اگر بخواهید یک مقدار جدید به فیلد name اختصاص دهید، با خطا مواجه می‌شوید. نکته دیگری که باید به آن توجه کنید این است که، شما می‌توانید برای بخش set یا get سطح دسترسی ایجاد کنید. به تکه کد زیر توجه کنید:

```
public string Name
{
    get
    {
        return name;
    }
    private set
    {
        name = value;
    }
}
```

خاصیت Name فقط در خارج از کلاس قابل خواندن است اما متدها فقط داخل کلاس Person می‌توانند مقادیر جدید بگیرند. یک property می‌تواند دارای بیش از یک فیلد باشد. به کد زیر توجه کنید:

```
private string firstName;
private string lastName;

public FullName
{
   get { return firstName + " " + lastName; }
}
```

همانطور که در مثال بالا مشاهده می‌کنید یک property فقط خواندنی تعریف کرده‌ایم که مقدار برگشتی آن ترکیبی از دو فیلد firstName و lastName است که به وسیله فاصله از هم جدا شده‌اند. سی‌شارپ همچنین یک راه حل کوتاه برای ایجاد property ارائه می‌دهد. در این روش می‌توانید یک property بدون فیلد ایجاد کنید. گاهی اوقات ممکن است که شما اصلاً نخواهید اعتبار سنجی انجام دهید. در این صورت بهتر است که آینده نگر باشید و باز هم به ازای هر فیلد موجود در کلاس یک خاصیت تعریف کنید. البته برای کاهش کدنویسی، می‌توانید از نوع خلاصه شده property ها یعنی property های خودکار استفاده کنید:

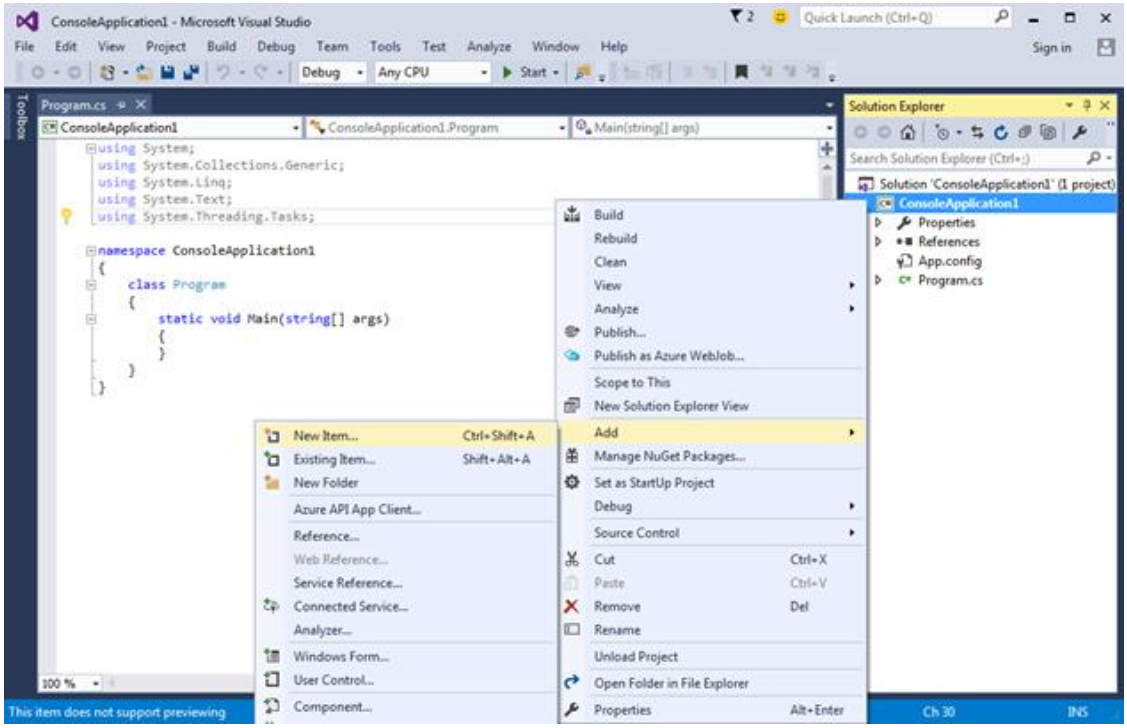
```
public int MyProperty { get; set; }
```

این ویژگی فراخوانی خودکار property نام دارد و در سی‌شارپ ۳،۰ معرفی شده است. به این نکته توجه کنید که در این روش هیچ کدی برای بخش set و get نمی‌نویسیم. دستور بالا معادل تعریف یک فیلد از نوع int با سطح دسترسی private است و property مربوط در یک خط مختصر نوشته شده و اجرا می‌شود. کامپایلر کد بالا را به صورت خودکار به عنوان یک property شناسایی می‌کند و فیلد مربوط به آن در طول زمان اجرای برنامه ساخته می‌شود. توجه کنید وقتی یک property خودکار ایجاد می‌کنید، باید هر دو بخش get و set آن را نیز تعریف کنید. همچنین نباید هیچ کدی در داخل این دو بخش بنویسید. بعدها اگر لازم بود که برای فیلدها اعتبارسنجی صورت بگیرد، می‌توانید بدون اینکه کدی که از کلاس استفاده می‌کند نیاز به تغییری داشته باشد، property های خودکار را به صورت property های معمولی ولی با اعتبارسنجی بنویسید.

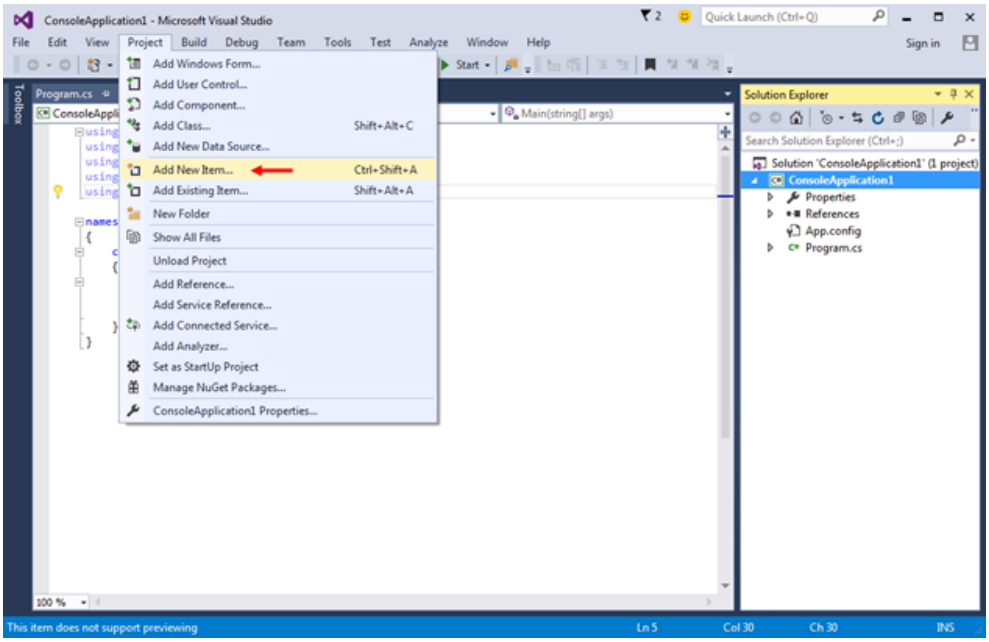
فضای نام

فضای نام راهی برای دسته بندی کدهای برنامه می‌باشد. هر چیز در دات‌نت حداقل در یک فضای نام قرار دارد. وقتی برای یک کلاس اسمی انتخاب می‌کنید، ممکن است برنامه نویسان دیگر به صورت اتفاقی اسمی شبیه به آن، برای کلاسشان انتخاب کنند. وقتی شما از آن کلاس‌ها در برنامه‌تان استفاده کنید، از آنجاییکه از کلاس‌های همنام استفاده می‌کنید، در برنامه ممکن است خطا به وجود آید.

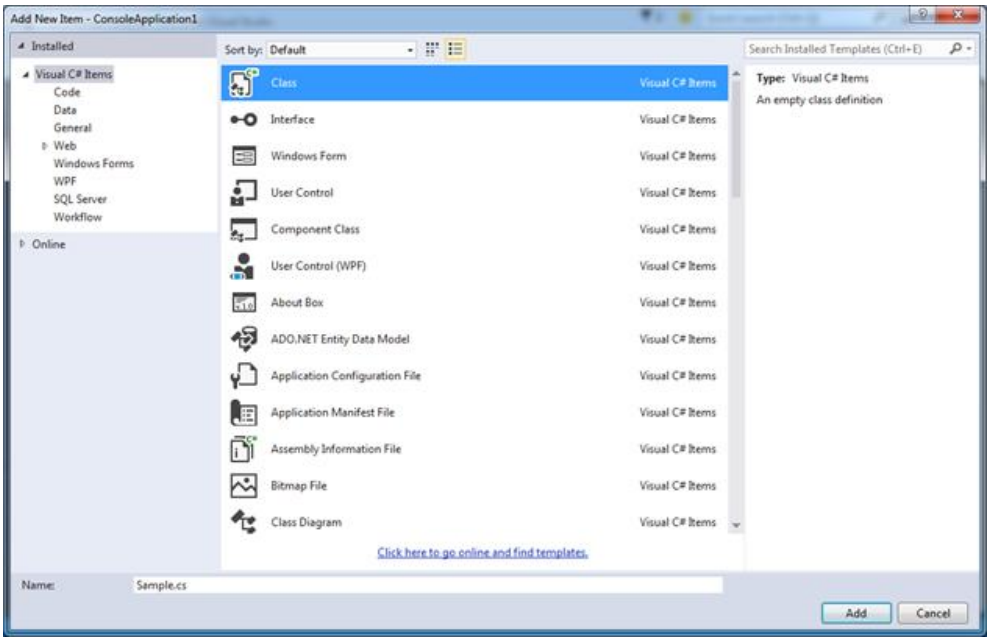
فضاهای نامی از وقوع این خطاها جلوگیری کرده یا آنها را کاهش می‌دهند. تا کنون و در درس‌های قبلی ما فقط با یک فضای نام آشنا شده‌ایم و آن فضای نام System است که شامل تعداد زیادی کلاس و متد، مانند کلاس Console و متد Writeline() می‌باشد. اما اگر یک پروژه جدید ایجاد کنید به صورت پیش‌فرض یک فضای نام برای شما ایجاد خواهد شد که نام آن شبیه به نام پروژه‌تان می‌باشد. در این درس به شما نشان می‌دهیم که چگونه کلاسهایتان را در کدهای جداگانه بنویسید و سپس از آنها در فایل‌های جدا استفاده کنید. برنامه Visual Studio Community را اجرا و یک پروژه جدید ایجاد کنید. بعد از اینکه پروژه ایجاد شد، یک فایل جدید ایجاد کنید. چندین راه برای ایجاد یک فایل وجود دارد. یکی از راه‌ها این است که، بر روی پروژه‌تان در Solution Explorer راست کلیک کرده و سپس گزینه Add و بعد New Item را انتخاب کنید.



راه دیگر این است که در نوار منو بر روی گزینه `Project` و سپس روی دکمه `Add New Item` کلیک کنید:



همچنین می‌توان از دکمه تر کیبی `Ctrl + Shift + A` استفاده کرد . هر کدام از این راه‌ها را که انتخاب کنید در نهایت یک صفحه برای شما نشان داده می‌شود و از شما سؤال می‌شود که چه فایلی را می‌خواهید ایجاد کنید:



گزینه Class را انتخاب کرده و نام آنرا Sample.cs بگذارید. کلاس‌های سی‌شارپ دارای پسوند cs هستند. برای درک منظور این درس همه کدهایی که در هنگام ایجاد کلاس به وجود می‌آیند را حذف کنید و کدهای زیر را وارد کنید:

```
namespace MyNamespace
{
    class Sample
    {
        public void ShowMessage()
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

همانطور که در کد بالا مشاهده می‌کنید فضای نام مان را تعریف کرده و نام آن را MyNamespace می‌گذاریم. در داخل کلاس مان (Sample) یک متد برای نمایش پیغام وجود دارد. به این نکته توجه کنید که با استفاده از فضای نام System، به متد WriteLine() کلاس Console دسترسی یافته‌ایم. می‌توانید با استفاده از کلمه کلیدی using فضای نام را از قبل تعریف کنید و آن را هنگام فراخوانی متد WriteLine() ننویسید (کد زیر). حال به فایل Program.cs برنامه‌ای که قبلاً ایجاد کردید بروید. محتویات آنرا پاک کرده و کدهای زیر را می‌نویسید:

```
using MyNamespace;

class Program
{
    static void Main()
    {
        Sample test = new Sample();

        test.ShowMessage();
    }
}
```

با استفاده از کلمه کلیدی using همه محتویات فضای نام MyNamespace را که قبلاً ایجاد کردیم، وارد برنامه جدید می‌کنیم (خط ۱). اگر خط اول کد بالا را حذف کنیم، برای استفاده از هر چیز باید قبل از Sample از کلمه MyNamespace استفاده کنیم.

```
MyNamespace.Sample test = new MyNamespace.Sample();
```

می‌توان چندین کلاس یا رابط (interface) را به یک فضای نام اضافه کرد:

```
namespace MyNamespace
{
    class Sample1
    {

    }

    class Sample2
    {

    }
}
```

شما محدود به دسته بندی کدهای کلاستان در داخل یک فضای نام نیستید. می‌توانید یک فضای نام تو در تو ایجاد کرده و کدهایتان را در درون آن بنویسید:

```
namespace MyNamespace1
{
    namespace MyNamespace2
    {
        class Sample
        {
            public void ShowMessage()
            {
                System.Console.WriteLine("Hello World!");
            }
        }
    }
}
```

برای دسترسی به کلاس Sample، مجبورید اول نام تمام فضاهای نامی را که کلاس Sample در آنها قرار دارد بنویسید:

```
MyNamespace1.MyNamespace2.Sample
```

یا می‌توان از کلمه کلیدی using استفاده کرد:

```
using MyNamespace1.MyNamespace2;
```

دات نت فریم ورک دارای فضاهای نام تو در تو می‌باشد. به عنوان مثال System.Data.SqlClient سه فضای تو در تو می‌باشد. می‌توان برای راحتی در کد نویسی فضاهای نامی تو در تو، یک فضای نامی مستعاری ایجاد کنید:

```
using AliasNamespace = MyNamespace1.MyNamespace2;
```

ساختارها در برابر کلاس‌ها

تفاوت بین کلاس و ساختار چیست؟ ساختارها انواع مقداری هستند مانند int، Double و String. وقتی یک مقدار از ساختار را در یک متغیر کپی می‌کنید، در اصل خود مقدار را کپی کرده‌اید نه آدرس یا مرجع آن را. کلاس‌ها انواع مرجع هستند مانند همه کلاس‌های دات نت. اجازه دهید تفاوت این دو را با یک مثال توضیح دهیم.

```
using System;

struct MyStructure
{
    public string Message { get; set; }
}

class MyClass
{
    public string Message { get; set; }
}

class Program
{
    static void Main()
    {
        MyStructure structure1 = new MyStructure();
        MyStructure structure2 = new MyStructure();

        structure1.Message = "ABC";
        structure2 = structure1;

        Console.WriteLine("Showing that structure1 " +
            "was copied to structure2.");
        Console.WriteLine("structure2.Message = {0}", structure2.Message);

        Console.WriteLine("\nModifying the value of structure2.Message...");
        structure2.Message = "123";

        Console.WriteLine("\nShowing that structure1 was not affected " +
            "by the modification of structure2");
        Console.WriteLine("structure1.Message = {0}", structure1.Message);


        MyClass class1 = new MyClass();
        MyClass class2 = new MyClass();

        class1.Message = "ABC";
        class2 = class1;

        Console.WriteLine("\n\nShowing that class1 " +
            "was copied to class2.");
        Console.WriteLine("class2.Message = {0}", class2.Message);

        Console.WriteLine("\nModifying the value of class2.Message...");
        class2.Message = "123";

        Console.WriteLine("\nShowing that class1 was also affected " +
            "by the modification of class2");
        Console.WriteLine("class1.Message = {0}", class1.Message);
    }
}
```

```
Showing that structure1 was copied to structure2.
structure2.Message = ABC
```

```
Modifying the value of structure2.Message...

Showing that structure1 was not affected by the modification of structure2
structure1.Message = ABC


Showing that class1 was copied to class2.
class2.Message = ABC

Modifying the value of class2.Message...

Showing that class1 was also affected by the modification of class2
class1.Message = 123
```

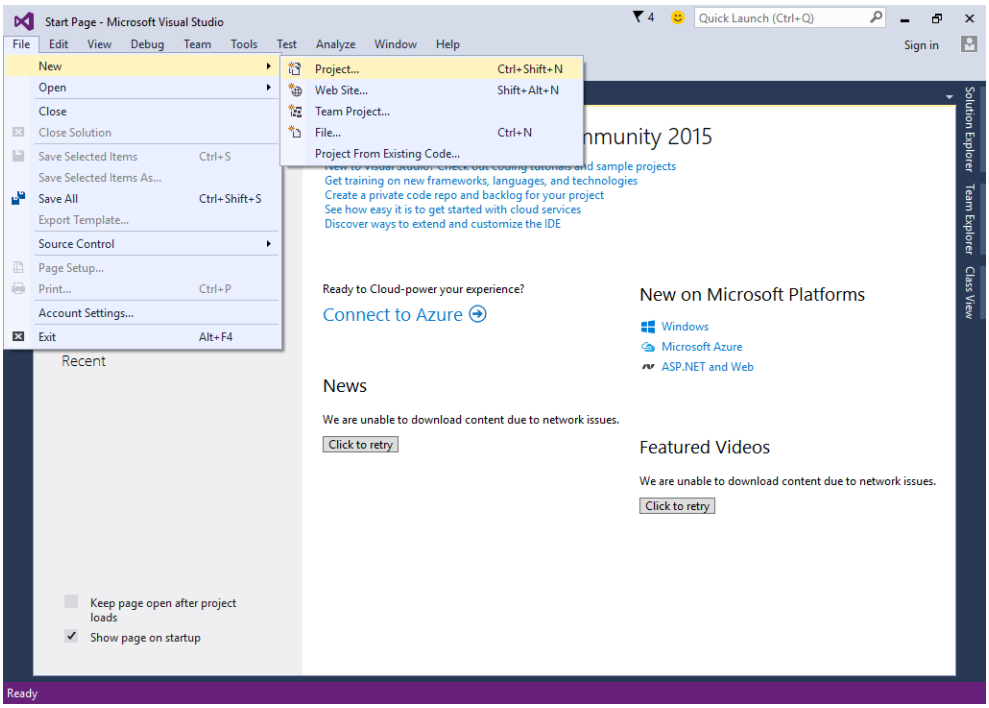
در بالا یک ساختار و یک کلاس ایجاد کرده‌ایم و تفاوت بین استفاده از این دو را نشان داده‌ایم. یک خاصیت به نام Message برای هر دو قرار داده‌ایم. سپس دو نمونه از هر کدام ایجاد کرده‌ایم (خطوط ۱۸-۱۷ و ۳۶-۳۵). مقداری به خاصیت Message از نمونه اول ایجاد شده از ساختار (structure1) اختصاص می‌دهیم. سپس مقدار structure1 را برابر structure2 قرار می‌دهیم، با این کار همه چیزهای داخل structure1 در structure2 کپی می‌شود.

برای ثابت کردن اینکه همه محتویات structure1 کپی شده است، مقدار خاصیت Message، structure2 را نشان می‌دهیم و مشاهده می‌کنیم که همان مقدار خاصیت Message، structure1 می‌باشد. برای اثبات اینکه ساختارها انواع مقداری هستند یک پیغام دیگر را به خاصیت Message، structure2 اختصاص می‌دهیم. خاصیت Message، structure1 تحت تأثیر قرار نمی‌گیرد چون structure2 یک کپی از structure1 می‌باشد.

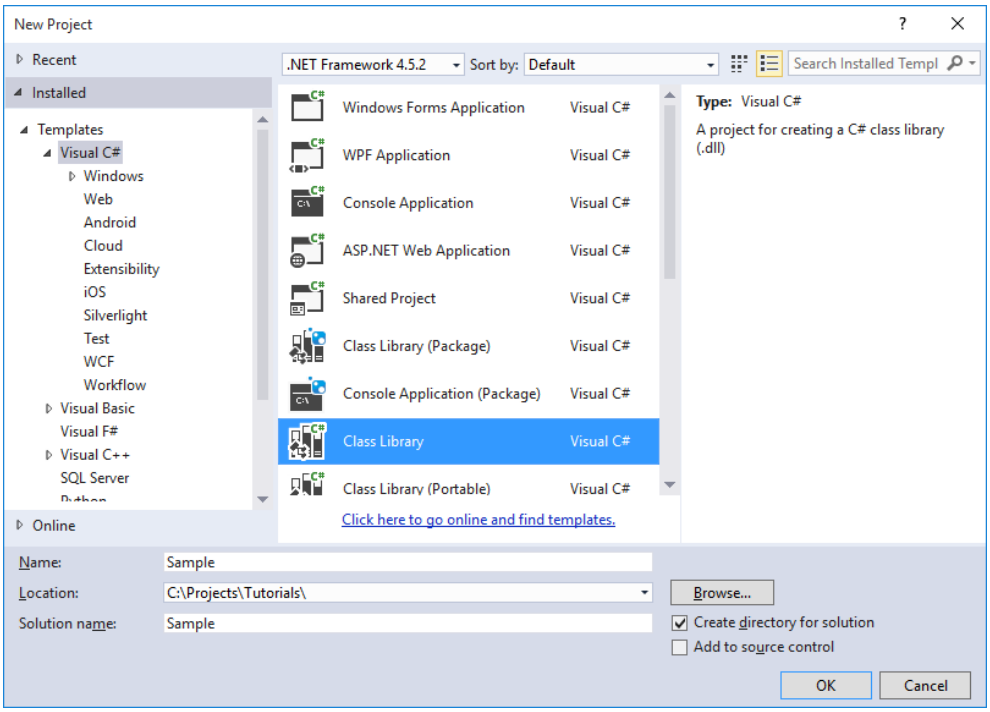
حال اثبات می‌کنیم که چرا کلاس‌ها انواع مرجع هستند. کلاس‌ها وقتی با یک متغیر برابر قرار داده می‌شوند، آدرس خود را ارسال می‌کنند نه مقدارشان را. بنابراین وقتی یک خاصیت از شیی که دارای آدرس شیء اصلی است را ویرایش می‌کنید خاصیت شیء اصلی نیز تغییر می‌کند. وقتی یک شیء را به عنوان آرگومان به متد ارسال می‌کنید، فقط آدرس شیء ارسال می‌شود. هر تغییری که در شیء داخل متد به وجود بیاید، بر شیء اصلی که آدرس آن به متد ارسال شده است نیز تأثیر می‌گذارد.

کتابخانه کلاس

می‌توانید کتابخانه کلاسی ایجاد کنید که مجموعه‌ای از کلاس‌ها و کدهایی است که می‌توانند کامپایل شوند و در نرم افزارهای دیگر برای استفاده مجدد به کار روند. کتابخانه‌های کلاس به فایل‌های DLL کامپایل می‌شوند. DLLها با هیچ برنامه‌ی خواندن فایل متنی، قابل خواندن نیستند. کتابخانه کلاس پروژه‌ای است که می‌توان از آن به عنوان یک راه حل شخصی برای حل مشکلات استفاده کرد. حال یک کتابخانه کلاس به روش زیر ایجاد کرده و نام آن را Sample می‌گذاریم. به مسیر Go to File > New> Project می‌رویم:



در لیست ظاهر شده گزینه `Class Library` را انتخاب می‌کنیم و همانطور که نشان داده شده است نام آن را `Sample` می‌گذاریم.

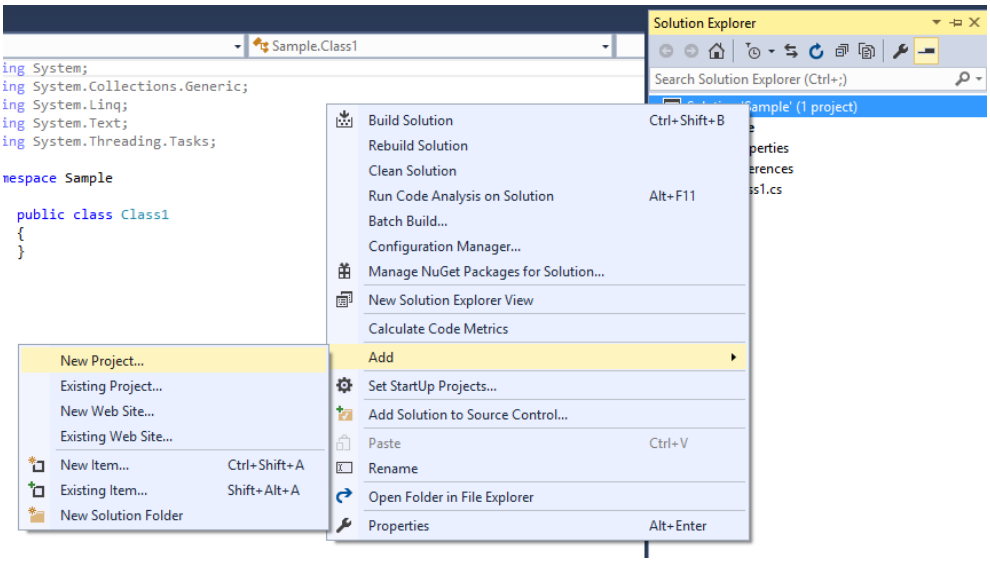


بعد از ایجاد کتابخانه کلاس، یک کلاس به طور خودکار با نام `Class1` ایجاد می‌شود و شما می‌توانید کدهایی که لازم دارید را به آن اضافه کنید:

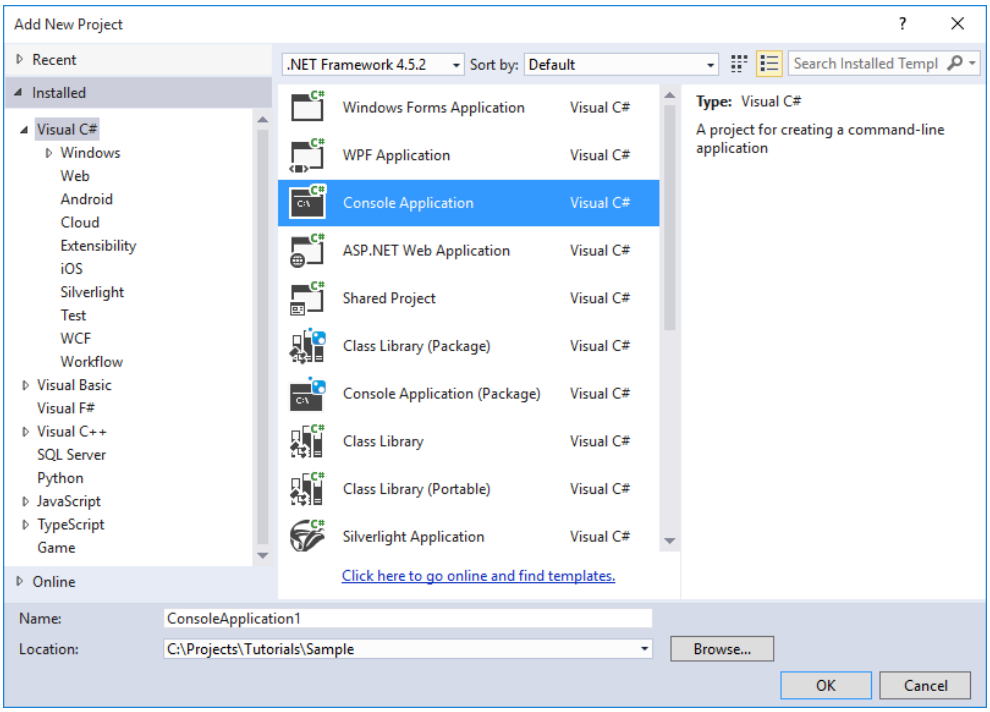
```
public class Class1
{
    public void ShowMessage()
    {
```

```
        Console.WriteLine("Hello World!");
    }
}
```

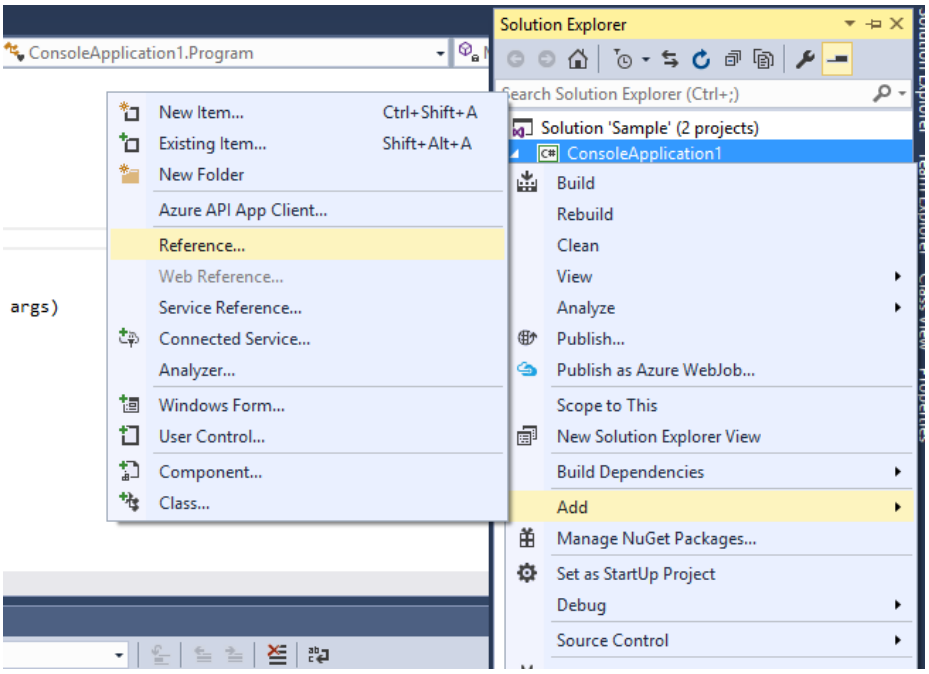
بعد از نوشتن کدها در داخل کلاس در قسمت Solution Explorer به مسیر Build > Build choosing بروید. این کار باعث می‌شود که یک فایل با پسوند DLL ایجاد شود که کدهای نوشته شده توسط شماست و می‌توانید از آن در پروژه‌های بعدی استفاده کنید. یک برنامه کنسولی دیگر ایجاد و بر روی نام پروژه راست کلیک کنید و سپس به مسیر New Project > Add choose بروید:



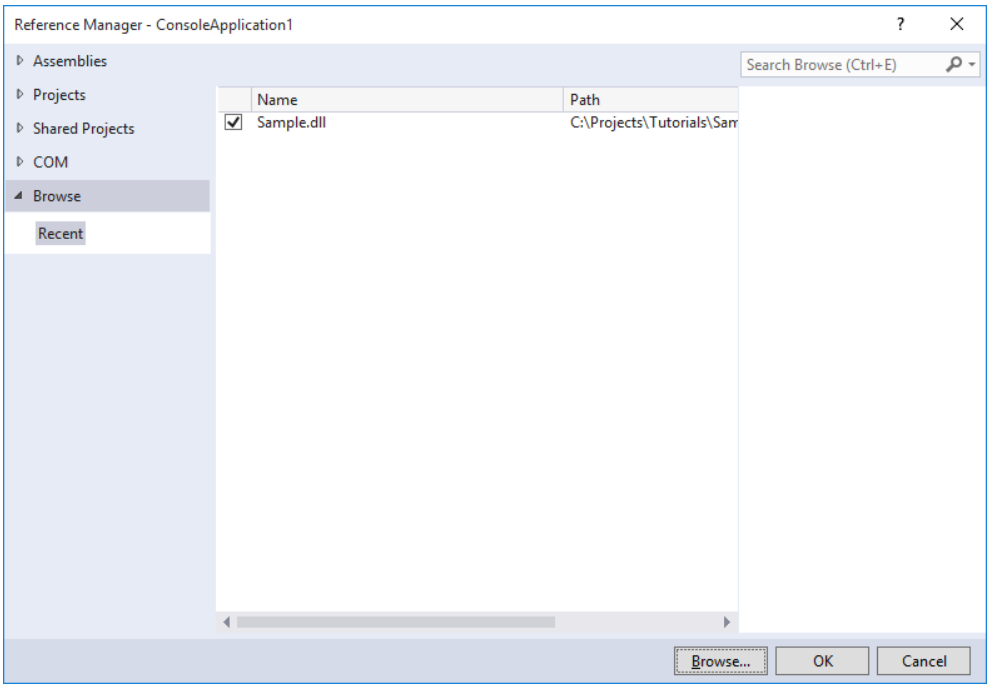
سپس Console Application را انتخاب کنید:



بعد از ایجاد برنامه جدید، DLL ی را که از قبل ایجاد کرده‌اید به آن اضافه کنید. برای اضافه کردن DLL بر روی پروژه راست کلیک کرده و مانند شکل زیر گزینه Add Reference را انتخاب می‌کنیم:

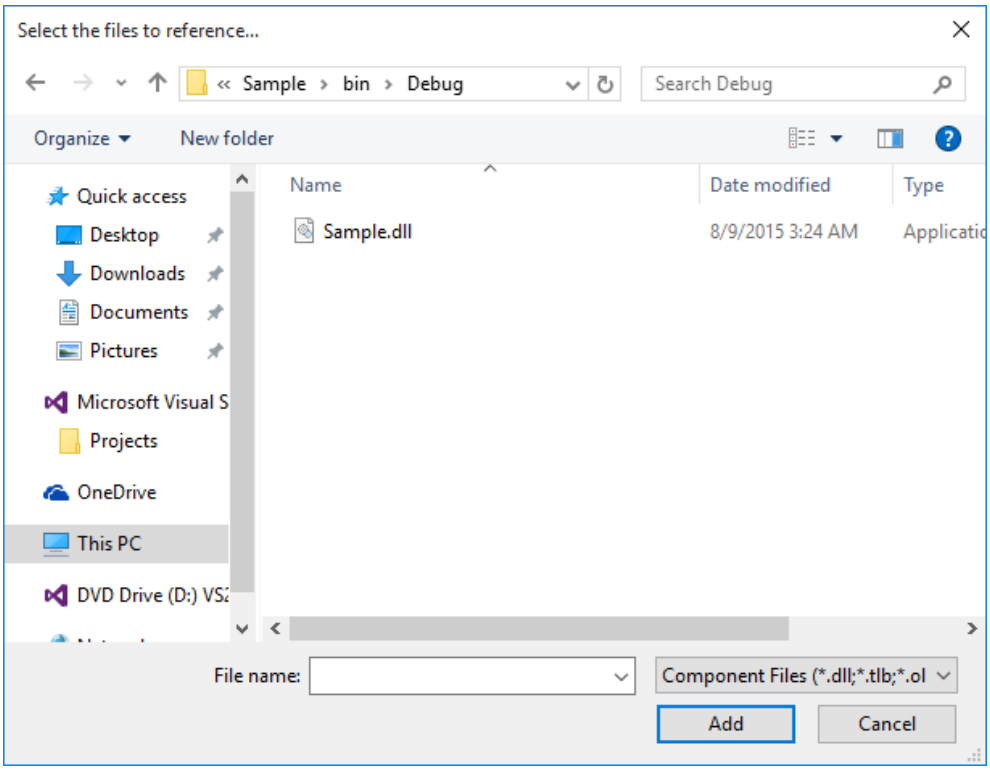


با کلیک بر روی این گزینه کادر زیر نمایش داده خواهد شد:

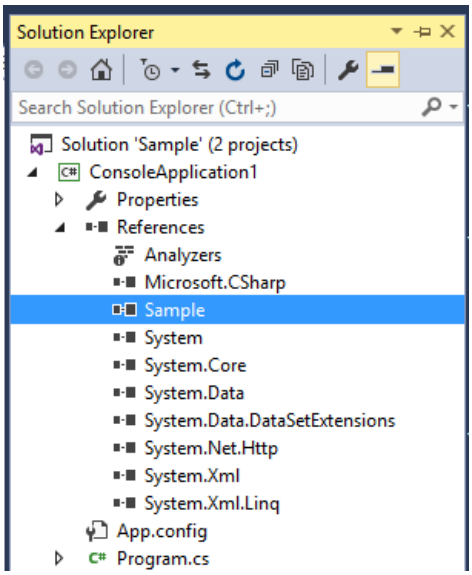


به این نکته توجه کنید که همه این فایل‌ها در حالت پیش‌فرض وارد پروژه شما نمی‌شوند. اگر پوشه References داخل Solution Explorer را باز کنید، همه فایل‌های اسمبلی که در دسترس شما قرار داده شده‌اند را، مشاهده خواهید کرد. اگر بخواهید از فایل‌های اسمبلی بیشتری در

دات‌نت استفاده کنید، می‌توانید آنها را از اولین سربرگ (سربرگ Assemblies) به پروژه اضافه کنید. سربرگ projects، solution جاری را برای پروژه‌های کتابخانه کلاس اسکن کرده و بعد از یافتن، آنها را لیست می‌کند. سومین سربرگ (Shared Projects) برای پیدا کردن فایل‌های DLL ی که در خارج از solution جاری هستند به کار می‌رود. به عنوان مثال اگر شما یک کتابخانه کلاس ساده در یک solution جداگانه ایجاد کنید، می‌توانید آن را وارد solution دیگر کرده و از آن استفاده کنید. سربرگ Browse هم به شما اجازه می‌دهد که، فایل‌های DLL ی که قبلاً در کامپیوتر ذخیره کرده‌اید را یافته و به برنامه اضافه کنید. مکان فایل DLL. در داخل پوشه Debug یا Release پروژه کتابخانه کلاس می‌باشد. این دو پوشه نیز به نوبه خود در داخل پوشه bin واقع در پوشه Solution کتابخانه کلاس قرار دارند.



بعد از زدن دکمه add فایل DLL به صورت زیر به پروژه اضافه می‌شود:



حال کدهای زیر را در فایل ConsoleApplication1's Program.cs بنویسید:

```
using Sample;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Class1 sampleClass = new Class1();
            sampleClass.ShowMessage();
        }
    }
}
```

سپس بر روی نام پروژه ConsoleApplication1 در Solution Explorer راست کلیک کرده و گزینه Set as Startup Project را کلیک کنید. سپس برنامه را اجرا و نتیجه را مشاهده کنید:

```
Hello World!
```

وراثت

وراثت به یک کلاس اجازه می‌دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد. وراثت مانند رابطه پدر و پسری می‌ماند، به طوریکه فرزند خصوصیاتی از قبیل قیافه و رفتار را از پدر خود به ارث برده باشد.

- کلاس پایه یا کلاس والد کلاسی است که بقیه کلاس‌ها از آن ارث می‌برند.
- کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می‌کند.

همه متد و خصوصیات کلاس پایه می‌توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضاء و متدهای با سطح دسترسی private.

همه کلاس‌ها در .NET Framework از کلاس Object ارث می‌برند. مفهوم اصلی وراثت در مثال زیر نشان داده شده است:

```
1  using System;
2
3  class Parent
4  {
5      private string message;
6
7      public string Message
8      {
9          get { return message; }
10         set { message = value; }
11     }
12
13     public void ShowMessage()
14     {
15         Console.WriteLine(message);
16     }
17
18     public Parent(string message)
19     {
20         this.message = message;
21     }
22 }
23
24 class Child : Parent
25 {
26     public Child(string message)
27         : base(message)
28     {
29
30     }
31 }
```

در این مثال دو کلاس با نام‌های Parent و Child تعریف شده است. در این مثال یک عضو را یکبار با سطح دسترسی private (خط ۵) و یکبار با سطح دسترسی public (خط ۱۱-۷)، و سپس یک متد را برای نمایش پیام تعریف کرده‌ایم. یک سازنده در کلاس Parent تعریف شده است که یک آرگومان از نوع رشته قبول می‌کند و یک پیغام نمایش می‌دهد. حال به کلاس Child توجه کنید (خط ۲۴). این کلاس تمام متدها و خاصیت‌های کلاس Parent را به ارث برده است. نحوه ارث بری یک کلاس به صورت زیر است:

```
class DerivedClass : BaseClass
```

براحتی می‌توان با قرار دادن یک کالن (:) بعد از نام کلاس و سپس نوشتن نام کلاسی که از آن ارث بری می‌شود (کلاس پایه) این کار را انجام داد. در داخل کلاس Child هم یک سازنده ساده وجود دارد که یک آرگومان رشته‌ای قبول می‌کند. وقتی از وراثت در کلاس‌ها استفاده می‌کنیم، هم سازنده کلاس مشتق و هم سازنده پیش‌فرض کلاس پایه، هر دو اجرا می‌شوند. سازنده پیش‌فرض یک سازنده بدون پارامتر است. اگر برای یک کلاس سازنده‌ای تعریف نکنیم، کامپایلر به صورت خودکار یک سازنده برای آن ایجاد می‌کند.

اگر هنگام صدا زدن سازنده کلاس مشتق بخواهیم سازنده کلاس پایه را صدا بزنیم، باید از کلمه کلیدی base استفاده کنیم. کلمه کلیدی base یک سازنده از کلاس پایه را صدا می‌زند. قبل از این کلمه کلیدی باید علامت کالن (:) را تایپ کنیم. در مثال بالا به وسیله تأمین مقدار پارامتر message سازنده کلاس مشتق و ارسال آن به داخل پرانتز کلمه کلیدی base، سازنده معادل آن در کلاس پایه فراخوانی شده و مقدار message را به آن ارسال می‌کند. سازنده کلاس parent هم این مقدار (مقدار message) را در یک عضو داده‌ای (فیلد) Private قرار می‌دهد. می‌توانید کدهایی را به داخل بدنه سازنده Child اضافه کنید تا بعد از سازنده Parent اجرا شوند. اگر از کلمه کلیدی base استفاده نشود، به جای کلاس پایه، سازنده

پیش‌فرض فراخوانی می‌شود. اجازه بدهید که اشیایی از کلاس‌های Parent و Child بسازیم تا نشان دهیم که چگونه کلاس Child متدها و خواص کلاس parent را به ارث می‌برد.

```
1   class Program
2   {
3       static void Main()
4       {
5           Parent myParent = new Parent("Message from parent.");
6           Child myChild = new Child("Message from child.");
7
8           myParent.ShowMessage();
9
10          myChild.ShowMessage();
11
12          myParent.Message = "Modified message of the parent.";
13          myParent.ShowMessage();
14
15          myChild.Message = "Modified message of the child.";
16          myChild.ShowMessage();
17
18          //myChild.message; ERROR: can't access private members of base class
19      }
20  }
```

```
Message from parent.
Message from child.
Modified message of the parent.
Modified message of the child.
```

هر دو شیء را با استفاده از سازنده‌های مربوط به خودشان مقدار دهی می‌کنیم (خطوط ۶-۵). سپس با استفاده از ارث بری و از طریق شیء Child به اعضاء و متدهای کلاس parent دسترسی می‌یابیم. حتی اگر کلاس Child از کلاس Parent ارث ببرد باز هم اعضای با سطح دسترسی Private در کلاس Child قابل دسترسی نیستند (خط ۱۸). سطح دسترسی Protect که در درس آینده توضیح داده خواهد شد، به شما اجازه دسترسی به اعضاء و متدهای کلاس پایه را می‌دهد. به نکته دیگر توجه کنید. اگر کلاس دیگری بخواهد از کلاس Child ارث بری کند، باز هم تمام متدها و خواص کلاس Child که از کلاس Parent به ارث برده است، را به ارث می‌برد.

```
class GrandChild : Child
{
    //Empty Body
}
```

این کلاس هیچ چیزی در داخل بدنه ندارد. وقتی کلاس GrandChild را ایجاد می‌کنید و یک خاصیت از کلاس Parent را فراخوانی می‌کنید با خطا مواجه می‌شوید. چون هیچ سازنده‌ای که یک آرگومان رشته‌ای قبول کند در داخل بدنه GrandChild تعریف نشده است، بنابراین شما می‌توانید فقط از سازنده پیش‌فرض یا بدون پارامتر استفاده کنید.

```
GrandChild myGrandChild = new GrandChild();

myGrandChild.Message = "Hello my grandchild!";
myGrandChild.ShowMessage();
```

وقتی یک کلاس ایجاد می‌کنیم و سازنده GrandChild را فراخوانی می‌کنیم، ابتدا سازنده کلاس Parent فراخوانی می‌شود و سپس سازنده Child و در نهایت سازنده GrandChild اجرا می‌شود. برنامه زیر ترتیب اجرای سازنده‌ها را نشان می‌دهد. دوباره کلاس‌ها را برای خوانایی بیشتر در داخل کدهای جدا قرار می‌دهیم.

```
using System;

class Parent
{
    public Parent()
    {
        Console.WriteLine("Parent constructor was called!");
    }
}

class Child : Parent
{
    public Child()
    {
        Console.WriteLine("Child constructor was called!");
    }
}

class GrandChild : Child
{
    public GrandChild()
    {
        Console.WriteLine("GrandChild constructor was called!");
    }
}

class Program
{
    static void Main()
    {
        GrandChild myGrandChild = new GrandChild();
    }
}
```

```
Parent constructor was called!
Child constructor was called!
GrandChild constructor was called!
```

سطح دسترسی Protect

سطح دسترسی protect اجازه می‌دهد که اعضای کلاس، فقط در کلاس‌های مشتق شده از کلاس پایه قابل دسترسی باشند. بدیهی است که خود کلاس پایه هم می‌تواند به این اعضاء دسترسی داشته باشد. کلاس‌هایی که از کلاس پایه ارث بری نکرده‌اند نمی‌توانند به اعضای با سطح دسترسی protect دست یابند. در مورد سطوح دسترسی public و private قبلاً توضیح دادیم. در جدول زیر نحوه دسترسی به سه سطح ذکر شده، نشان داده شده است:

protected	private	public	قابل دسترسی در
true	true	true	داخل کلاس
false	false	true	خارج از کلاس

true	false	true	کلاس مشتق

مشاهده می‌کنید که public بیشترین سطح دسترسی را داراست. صرف نظر از مکان:

- اعضای public در هر جا فراخوانی می‌شوند و قابل دسترسی هستند.
- اعضای private فقط در داخل کلاسی که به آن تعلق دارند قابل دسترسی هستند.
- اعضای protect فقط در کلاسی که در آن تعریف شده‌اند و همچنین کلاس‌های مشتق شده از آن کلاس قابل دسترسی هستند.

کد زیر رفتار اعضای دارای این سه سطح دسترسی را نشان می‌دهد:

```
1   using System;
2
3   class Parent
4   {
5       protected int protectedMember = 10;
6       private int privateMember = 10;
7       public int publicMember = 10;
8   }
9
10  class Child : Parent
11  {
12      public Child()
13      {
14          protectedMember = 100;
15          privateMember = 100;
16          publicMember = 100;
17      }
18  }
19
20  class Program
21  {
22      public static void Main()
23      {
24          Parent myParent = new Parent();
25
26          myParent.protectedMember = 100;
27          myParent.privateMember = 100;
28          myParent.publicMember = 100;
29      }
30  }
```

کدهایی خطوط ۲۶ و ۲۷ باعث بروز خطا می‌شوند، چون آنها اجازه دسترسی به فیلدهای Protect و private کلاس Parent را ندارند. همانطور که در خط ۱۵ مشاهده می‌کنید کلاس Child سعی می‌کند که به عضو Private کلاس Parent دست یابد. از آنجاییکه اعضای Private در خارج از کلاس قابل دسترسی نیستند، حتی کلاس مشتق در خط ۱۵ نیز ایجاد خطا می‌کند. اگر شما به خط ۱۴ توجه کنید کلاس Child می‌تواند به عضو Protect کلاس Parent دسترسی یابد، چون کلاس Child از کلاس Parent مشتق شده است. حال به خط ۲۶ جایی که می‌خواهیم در کلاس Program به فیلد Protect کلاس Parent دسترسی یابیم، نگاهی بیندازید. می‌بینید که برنامه پیغام خطا می‌دهد چون کلاس Program از کلاس Parent مشتق نشده است. همچنین کلاس Program به اعضای Private کلاس Parent نیز نمی‌تواند دسترسی یابد.

اعضای Static

اگر بخواهیم عضو داده‌ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه‌های کلاس قابل دسترسی باشد از کلمه کلیدی static استفاده می‌کنیم. کلمه کلیدی static برای اعضای داده‌ای و خاصیت‌هایی به کار می‌رود که، می‌خواهند در همه نمونه‌های کلاس تقسیم شوند. وقتی که یک متد یا خاصیت به صورت static تعریف شود، می‌توانید آنها را بدون ساختن نمونه‌ای از کلاس، فراخوانی کنید. به مثالی در مورد متدها و خاصیت‌های static توجه کنید:

```
using System;

class SampleClass
{
    public static string StaticMessage { get; set; }
    public string NormalMessage { get; set; }

    public static void ShowStaticMessage()
    {
        Console.WriteLine(StaticMessage);
    }

    public void ShowNormalMessage()
    {
        Console.WriteLine(NormalMessage);
    }

    public void ShowStaticFromInstance()
    {
        Console.WriteLine(StaticMessage);
    }
}

class Program
{
    public static void Main()
    {
        SampleClass sample1 = new SampleClass();
        SampleClass sample2 = new SampleClass();

        SampleClass.StaticMessage = "This is the static message!";
        SampleClass.ShowStaticMessage();

        sample1.NormalMessage = "\nMessage from sample1!";
        sample1.ShowNormalMessage();
        sample1.ShowStaticFromInstance();

        sample2.NormalMessage = "\nMessage from sample2!";
        sample2.ShowNormalMessage();
        sample2.ShowStaticFromInstance();
    }
}
```

```
This is the static message!

Message from sample1!
This is the static message!

Message from sample2!
This is the static message!
```

در مثال بالا یک خاصیت استاتیک به نام StaticMessage (خط ۵) و یک متد استاتیک به نام ()ShowStaticMessage (خطوط ۱۱-۸) تعریف کرده‌ایم. مقدار خاصیت StaticMessage در همه نمونه‌های کلاس SampleClass قابل دسترسی است. متد استاتیک را نمی‌توان به وسیله نمونه ایجاد شده از کلاس SampleClass فراخوانی کرد. برای فراخوانی یک متد با خاصیت استاتیک، به سادگی می‌توان نام کلاس و بعد از آن علامت دات (.) و در آخر نام متد یا خاصیت را نوشت. این موضوع را می‌توان در خطوط (۳۲-۳۱) مشاهده کرد. مشاهده می‌کنید که لازم نیست هیچ نمونه‌ای از کلاس ایجاد شود. همانطور که می‌بینید یک پیغام را به StaticMessage اختصاص داده‌ایم و با فراخوانی یک متد استاتیک (()ShowStaticMessage) مقدار آن را نمایش می‌دهیم.

در مرحله بعد خاصیت NormalMessage را به وسیله دو نمونه ایجاد شده فراخوانی می‌کنیم و یک متد را هم برای نشان دادن مقدار آن فراخوانی می‌کنیم. همانطور که مشاهده می‌کنید نمونه‌های ایجاد شده دارای مقدار NormalMessage مخصوص خودشان هستند چون خاصیت NormalMessage غیر استاتیک است. سپس ()ShowStaticFromInstance را فراخوانی می‌کنیم که متدی برای نشان دادن مقدار StaticMessage می‌باشد. متدهای غیر استاتیک می‌توانند از فیلدها و خاصیت‌های استاتیک استفاده کنند ولی عکس این قضیه امکان پذیر نیست. به عنوان مثال اگر شما یک متد static داشته باشید نمی‌توانید از هر خاصیت، متد، یا فیلدی که static نیست، استفاده کنید.

```
private int number = 10;

public static void ShowNumber()
{
   Console.WriteLine(number); //Error: cannot use non-static member
}
```

اصرار بر این کار باعث بروز خطا می‌شود. یک مثال متد ()WriteLine کلاس Console است. اگر از یک متد غیر استاتیک استفاده می‌کنید باید به روش زیر عمل کنید:

```
Console display = new Console();

display.WriteLine("My message!");
```

شما قبل از نمایش پیغام ابتدا باید یک نمونه از کلاس Console ایجاد کنید. به این نکته نیز توجه کنید که، متد ()Main باید به صورت static تعریف شود. کامپایلر به Static بودن متد ()Main نیاز دارد تا بتواند برنامه را بدون ساختن نمونه اجرا کند.

متدهای مجازی

متدهای مجازی، متدهایی از کلاس پایه هستند که می‌توان در کلاس مشتق آنها را override کرده و به صورت دلخواه پیاده سازی نمود. به عنوان مثال شما متد A را در کلاس A دارید و کلاس B از کلاس A ارث بری می‌کند، در این صورت متد A در کلاس B در دسترس خواهد بود. اما متد A دقیقاً همان متدی است که از کلاس A به ارث برده شده است. حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان دهد چکار می‌کنید؟ متد مجازی این مشکل را برطرف می‌کند. به تکه کد زیر توجه کنید.

```
1  using System;
2
3  class Parent
4  {
```

```
5       public virtual void ShowMessage()
6       {
7           Console.WriteLine("Message from Parent.");
8       }
9   }
10
11  class Child : Parent
12  {
13      public override void ShowMessage()
14      {
15          Console.WriteLine("Message from Child.");
16      }
17  }
18
19  class Program
20  {
21      public static void Main()
22      {
23          Parent myParent = new Parent();
24          Child myChild = new Child();
25
26          myParent.ShowMessage();
27          myChild.ShowMessage();
28      }
29  }
```

```
Message from Parent.
Message from Child.
```

متد مجازی با قرار دادن کلمه کلیدی virtual هنگام تعریف متد، تعریف می‌شود (خط ۵). این کلمه کلیدی نشان می‌دهد که متد می‌تواند override شود یا به عبارت دیگر می‌تواند به صورت دیگر پیاده سازی شود. کلاسی که از کلاس Parent ارث می‌برد شامل متدی است که متد مجازی کلاس پایه را override یا به صورت دیگری پیاده سازی می‌کند. با استفاده از کلمه کلیدی override می‌توان متد مجازی کلاس پایه را به صورت دیگر پیاده سازی کرد. با استفاده از کلمه کلیدی base (خط ۱۷ کد زیر) می‌توانید متد مجازی را در داخل متد override شده فراخوانی کنید:

```
1   using System;
2
3   class Parent
4   {
5       private string name = "Parent";
6
7       public virtual void ShowMessage()
8       {
9           Console.WriteLine("Message from Parent.");
10      }
11  }
12
13  class Child : Parent
14  {
15      public override void ShowMessage()
16      {
17          base.ShowMessage();
18          Console.WriteLine("Message from Child.");
19      }
20  }
21
22  class Program
23  {
24      public static void Main()
25      {
```

```
26          Parent myParent = new Parent();
27          Child myChild = new Child();
28
29          myParent.ShowMessage();
30          myChild.ShowMessage();
31      }
32  }
```

```
Message from Parent.
Message from Parent.
Message from Child.
```

اگر بخواهید از متد پایه استفاده کنید و هیچ کدی داخل متد override نباشد (مثلاً فرض کنید خط ۱۸ در مثال بالا وجود نداشته باشد)، شبیه به این است که از هیچ متد override ی استفاده نکرده‌اید. نمی‌توان متد غیر virtual و یا یک متد static را override کرد. متد override نیز باید دارای سطح دسترسی شبیه به متد مجازی باشد. می‌توان یک کلاس دیگر که از کلاس Child ارث بری می‌کند ایجاد کرده و دوباره متد ()ShowMessage را override کرده و آنرا به صورت دیگر پیاده سازی کنیم. اگر بخواهید متدی را که ایجاد کرده‌اید به وسیله سایر کلاس‌ها override نشود، کافیست که از کلمه کلیدی sealed به صورت زیر استفاده کنید:

```
public sealed override void ShowMessage()
```

حال اگر کلاس دیگری از کلاس Child ارث ببرد، نمی‌تواند متد ()ShowMessage را override کند. به یک مثال دیگر توجه کنید. فرض کنید می‌خواهیم متد ()ToString کلاس System.Object را override کنیم. همانطور که در درس آینده خواهید دید همه کلاس‌ها در سی‌شارپ از کلاس Object ارث می‌برند.

```
1   using System;
2
3   class Person
4   {
5       public string FirstName { get; set; }
6       public string LastName { get; set; }
7
8       public override string ToString()
9       {
10          return FirstName + " " + LastName;
11      }
12  }
13
14  class Program
15  {
16      public static void Main()
17      {
18          Person person1 = new Person();
19
20          person1.FirstName = "John";
21          person1.LastName = "Smith";
22
23          Console.WriteLine(person1.ToString());
24      }
25  }
```

```
John Smith
```

از آنجاییکه متد ()ToString کلاس System.Object را override کرده‌ایم، به جای چاپ نوع شیء پیش‌فرض خروجی ما سفارشی شده و نام و نام خانوادگی نمایش داده می‌شود.

کلاس آبجکت (System.Object Class)

همه کلاس‌های دات‌نت از کلاس آبجکت (System.Object) ارث می‌برند. کلاس آبجکت در سی‌شارپ با کلمه کلیدی object نشان داده می‌شود. برای راحتی در این درس از کلمه آبجکت به جای System.Object استفاده می‌کنیم. در زیر لیست برخی از متدهای معمول در کلاس آبجکت آمده است:

متد	نوع برگشتی	Virtual	Static
Object()	None	No	No
~Object()	None	No	No
Equals(object)	bool	Yes	No
Equals(object, object)	bool	No	Yes
ReferenceEquals (object,object)	bool	No	Yes
ToString()	string	Yes	No
MemberwiseClone()	object	No	No
GetType()	System.Type	No	No
GetHashCode()	int	Yes	No

همه متدهای این کلاس معمولاً مورد استفاده قرار نمی‌گیرند. از آنجاییکه همه کلاس‌های سی‌شارپ از این کلاس ارث می‌برند، آنها نیز دارای این متدها به جزء متدهای Static می‌باشند. وقتی یک کلاس ایجاد می‌کنید، این کلاس به صورت ضمنی از کلاس آبجکت (Object) ارث می‌برد. بنابراین وقتی یک کلاس تعریف می‌کنید کدها در حقیقت به صورت زیر به وسیله کامپایلر خوانده می‌شوند:

```
class MyClass : System.Object
{
}
```

اینکه چرا همه کلاس‌ها در دات‌نت از object ارث بری می‌کنند، به دلیل امکان استفاده از چندریختی است که در درس آینده در باره آن توضیح می‌دهیم. به عنوان مثال یکی از سربارگذاری‌های متد Console.WriteLine() قبول نوع آبجکت (object) به عنوان آرگومان است. به همین دلیل است که شما می‌توانید تقریباً هر چیز را به عنوان آرگومان به متد Console.WriteLine() ارسال کنید. برای نشان دادن اینکه هر چیز در سی‌شارپ یک شیء است. به مثال ساده زیر توجه کنید:

```
using System;
public class Program
{
    public static void Main()
    {
        int myInt = 1;
        double myDouble = 4.0;
        string myString = "Hello";

        Console.WriteLine(myInt.ToString());
```

```
        Console.WriteLine(myDouble.ToString());
        Console.WriteLine(myString.ToString());
    }
}
```

همانطور که مشاهده می‌کنید اشیاء int، double و string همگی متد ToString() را فراخوانی می‌کنند چون که این متد از کلاس object به ارث برده شده است.

Boxing و Unboxing

Boxing فرایندی است که طی آن یک نوع مقداری مانند ساختار (Struct) به یک نوع مرجع مانند یک شیء (Object) تبدیل می‌شود. Unboxing برعکس، عمل تبدیل یک نوع مرجع به یک نوع مقداری می‌باشد. کد زیر فرایند boxing را نشان می‌دهد.

```
struct MyStruct
{
    public int Number { get; set; }
}

class Program
{
    public static void Main()
    {
        MyStruct valueType = new MyStruct();
        valueType.Number = 10;
        object refType = valueType;
    }
}
```

در کد بالا یک ساختار به نام MyStruct ایجاد کرده‌ایم و یک خاصیت (property) برای تست اهداف برای آن در نظر گرفته‌ایم. در فرایند boxing نوع مقداری به سادگی با یک متغیر از نوع آبجکت برابر قرار داده می‌شود. در کد بالا refType شامل آدرس یک نوع MyStruct است نه آدرس اصلی متغیر valueType. در زیر نحوه تبدیل refType به نوع مقداری MyStruct به وسیله unboxing نشان داده شده است.

```
MyStruct valueType2 = (MyStruct)refType;
```

همانطور که مشاهده می‌کنید با استفاده از تبدیل صریح متغیر نوع مرجع refType را به متغیر نوع مقدار MyStruct تبدیل کرده‌ایم.

ترکیب (Containment)

محدود نگه داشتن یا ترکیب فرایندی است که طی آن یک کلاس به عنوان یک عضو به کلاس دیگر اضافه می‌شود. به عنوان مثال کلاس Person می‌تواند یک فیلد از نوع کلاس Name داشته باشد. به کد زیر توجه کنید:

```
using System;

class Name
{
    public string FirstName { get; set; }
    public string LastName  { get; set; }

    public Name(string f, string l)
    {
```

```
        FirstName = f;
        LastName  = l;
    }
}

class Person
{
    private Name myName;

    public Name MyName
    {
        get { return myName; }
        set { myName = value; }
    }

    public Person(Name name)
    {
        myName = new Name(name.FirstName, name.LastName);
    }

    public override string ToString()
    {
        return myName.FirstName + " " + myName.LastName;
    }
}

class Program
{
    public static void Main()
    {
        Person person1 = new Person(new Name("John", "Smith"));

        Console.WriteLine(person1.ToString());
    }
}
```

```
John Smith
```

حال برنامه را به صورت بخش بخش توضیح می‌دهیم:

```
class Name
{
    public string FirstName { get; set; }
    public string LastName  { get; set; }

    public Name(string f, string l)
    {
        FirstName = f;
        LastName  = l;
    }
}
```

یک کلاس که قرار است به عنوان یک فیلد در کلاس دیگر به کار رود را تعریف می‌کنیم. این کلاس دارای یک سازنده است که نام (FirstName) و نام خانوادگی (LastName) را از شخص دریافت می‌کند. مقادیر فیلدهای ذکر شده هم به وسیله خواص مربوطه‌شان به آنها اضافه می‌شود.

```
class Person
{
    private Name myName;

    public Name MyName
    {
        get { return myName; }
```

```
        set { myName = value; }
    }

    public Person(Name name)
    {
        myName = new Name(name.FirstName, name.LastName);
    }

    public override string ToString()
    {
        return myName.FirstName + " " + myName.LastName;
    }
}
```

این کلاس شامل یک فیلد از نوع Name و خاصیت متناظر با آن است. این خصوصیت مقدار نام هر شیء Person را در خود نگهداری می‌کند. به این نکته توجه کنید که سازنده یک شیء Name را می‌پذیرد و سپس با استفاده از نام شیء فیلد myName را مقداردهی می‌کند. به این فرایند ترکیب (aggregation) می‌گویند. همچنین در کلاس Person متد ()ToString از کلاس System.Object را بازنویسی (override) می‌کنیم به طوری که در هنگام فراخوانی نام کامل شخص را نمایش دهد.

```
class Program
{
    public static void Main()
    {
        Person person1 = new Person(new Name("John", "Smith"));

        Console.WriteLine(person1.ToString());
    }
}
```

در بالا یک شیء Person را ایجاد و از سازنده‌ای که یک شیء از نوع کلاس Name به عنوان آرگومان قبول می‌کند استفاده کرده‌ایم. این شیء را مستقیماً در داخل پرانتزها تعریف و همچنین مقادیر خصوصیات firstname و lastname را به آن ارسال کرده‌ایم. در نهایت، با استفاده از متد سفارشی ()ToString نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که کلاس‌ها می‌توانند شیءهایی از نوع خود کلاس داشته باشند. به عنوان نمونه کلاس Person می‌تواند یک عضو با نوع Person داشته باشد. به کد زیر توجه نمایید:

```
class Person
{
    public Person Sibling { get; set; }
    public string Name { get; set; }
}
```

مشاهده می‌کنید که چگونه فیلدی از نوع Person در داخل کلاس Person تعریف شده است. پس هنگامی که یک شیء Person تعریف می‌کنید، شیء ایجاد شده یک شیء از نوع Person در داخل خود دارد.

```
Person person1 = new Person();
person1.Sibling = new Person();
person1.Name = "John Smith";
person1.Sibling.Name = "Mike Smith";
```

کد بالا چگونگی دسترسی و مقدار دهی به عضو Person را نشان داده است. از آنجایی که خصوصیت Sibling از نوع Person است پس می‌تواند یک شیء Sibling در داخل خود داشته باشد. بنابراین می‌توان هر تعداد sibling که می‌خواهید در داخل شیء person1 داشته باشید:

```
person1.Sibling.Sibling = new Person();
person1.Sibling.Sibling.Name = "Franc Smith";
person1.Sibling.Sibling.Sibling = new Person();
person1.Sibling.Sibling.Sibling.Name = "Bob Smith";
//And so on...
```

سربارگذاری عملگرها

سربارگذاری عملگرها به شما اجازه می‌دهد که رفتار عملگرهای سی‌شارپ را بسته به نوع عملوندهای آنها سفارشی کنید. سربارگذاری عملگرها همچنین به عملگر اجازه می‌دهد که یک شیء را به روشی دیگر ترجمه کند. به کد زیر توجه کنید:

```
1   class MyNumber
2   {
3       public int Number { get; set; }
4   }
5
6   class Program
7   {
8       public static void Main()
9       {
10          MyNumber firstNumber  = new MyNumber();
11          MyNumber secondNumber = new MyNumber();
12
13          firstNumber.Number  = 10;
14          secondNumber.Number = 5;
15
16          MyNumber sum = firstNumber + secondNumber;
17      }
18  }
```

خط پررنگ شده در کد بالا (خط ۱۶) کد قابل قبولی نیست. چون کامپایلر نمی‌تواند دو شیء را با هم جمع کند. رفتاری که ما از کد بالا انتظار داریم اضافه کردن مقادیر به خاصیت Number دو عملوند و سپس ایجاد یک شیء جدید که حاصل جمع دو مقدار در داخل آن قرار بگیرد. سپس این شیء جدید به متغیر sum تخصیص داده شود.

سر بارگذاری عملگرهای دو تایی

برنامه را برای اضافه کردن سربارگذاری یک عملگر دوتایی (+) که دو عملوند قبول می‌کند، تغییر می‌دهیم.

```
using System;

class MyNumber
{
    public int Number { get; set; }

    public static MyNumber operator +(MyNumber n1, MyNumber n2)
    {
        MyNumber result = new MyNumber();
        result.Number = n1.Number + n2.Number;
        return result;
```

```
    }
}

class Program
{
    public static void Main()
    {
        MyNumber firstNumber  = new MyNumber();
        MyNumber secondNumber = new MyNumber();

        firstNumber.Number  = 10;
        secondNumber.Number = 5;

        MyNumber sum = firstNumber + secondNumber;

        Console.WriteLine("Sum = {0}", sum.Number);
    }
}
```

```
Sum = 15
```

برای سربارگذاری عملگرها به صورت زیر عمل کنید:

```
public static returnType operator operatorSymbol(type operand1, type operand2)
{
   //Codes here
   return result;
}
```

همانطور که مشاهده می‌کنید در سربارگذاری عملگرها از یک متد که هم static و هم public باشد، استفاده می‌شود. این متد باید static باشد چون همه نمونه‌های کلاس از آن استفاده می‌کنند و هم باید public باشد تا بتوان در خارج از کلاس از آن استفاده کرد. سپس از کلمه کلیدی operator و بعد از آن از علامت یک عملگر مانند + یا – استفاده می‌کنیم. در سربارگذاری یک عملگر دوتایی به دو عملوند نیاز است، بنابراین متد دارای دو پارامتر است که این دو عملوند را قبول می‌کند. در داخل کد یک شیء ایجاد شده است که نتیجه را در خود نگداری می‌کند.

دو خاصیت Number برای دو پارامتر اضافه کرده و حاصل جمع این دو را در خاصیت Number شیء result (که نتیجه را در خود ذخیره می‌کند) قرار می‌دهیم (خط ۱۲). و در آخر نتیجه را به فراخوان بازگشت می‌دهیم (خط ۱۳). شیء result به متغیر sum ارجاع داده شده است. همه عملگرها نمی‌توانند سر بارگذاری شوند. مثلاً شما نمی‌توانید عملگر += را سربارگذاری کنید. شما می‌توانید عملگر + را سربارگذاری کنید که در این صورت عملگر =+ به صورت خودکار سربارگذاری می‌شود. عملگرهای > یا < باید به صورت جفت سربارگذاری شوند. مثلاً نمی‌توان عملگر > را به تنهایی سربارگذاری کنید.

```
public static bool operator >(MyNumber n1, MyNumber n2)
{
    return (n1.Number > n2.Number);
}

public static bool operator <(MyNumber n1, MyNumber n2)
{
    return (n1.Number < n2.Number);
}
```

سربارگذاری عملگرهای یگانی

سربارگذاری عملگر یگانی بسیار ساده است. همه کاری که شما باید انجام دهید تهیه یک پارامتر است چون عملگر یگانی، یک عملوند قبول می‌کند.

به عنوان مثال، اجازه دهید که عملگر یگانی ++ را سربارگذاری کنیم.

```
public static MyNumber operator ++(MyNumber n1)
{
    MyNumber result = new MyNumber();
    result.Number = n1.Number + 1;
    return result;
}
```

همانطور که می‌بینید سربارگذاری عملگرهای یگانی شبیه به سربارگذاری عملگرهای دوتایی است. در سربارگذاری عملگرها به نکات زیر توجه کنید:

- نمی‌توانید یک عملگر جدید ایجاد کنید.
- دستور زبان یک عملگر را تغییر دهید.
- عملکرد یک عملگر را نمی‌توان دوباره تعریف کرد.
- نمی‌توان تقدم یک عملگر را عوض کرد.

لیست عملگرهایی که قابلیت سربارگذاری را دارند در زیر آمده است.

عملگرهای دوتایی:

```
+, -, *, /, %, &, |, ^, <<, >>, ==, !=, >, <, >=, <=
```

عملگرهای یگانی

```
+, -, !, ~, ++, --, true, false
```

عملگر is

عملگر `is` در سی‌شارپ به شما اجازه می‌دهد که تست کنید که آیا یک شیء می‌تواند به طور کامل به وسیله تبدیل صریح به شیء دیگری تبدیل شود. عملگر `is` به دو عملوند نیاز دارد و یک مقدار بولی را برمی گرداند. به عنوان مثال، فرض کنید یک کلاس به نام `Animel` داریم، سپس یک نمونه از آن ایجاد می‌کنیم:

```
using System;

class Animal
{

}

class Program
{
    public static void Main()
    {
        Animal myAnimal = new Animal();
```

```
        if (myAnimal is Animal)
        {
            Console.WriteLine("myAnimal is an Animal!");
        }
    }
}
```

```
myAnimal is an Animal
```

رفتار عملگر is را در این مثال مشاهده کردید. همانطور که می‌بینید از آن به عنوان شرط در عبارت if استفاده شده است. کاربرد آن در مثال بالا این است که چک می‌کند که آیا شیء myAnimal یک نمونه از Animal است و چون نتیجه درست است کدهای داخل دستور if اجرا می‌شود. این عملگر همچنین می‌تواند چک کند که آیا یک شیء خاص در سلسله مراتب وراثت یک نوع خاص است. به این مثال توجه کنید:

```
using System;

class Animal
{

}

class Dog : Animal
{

}

class Program
{
    public static void Main()
    {
        Dog myDog = new Dog();

        if (myDog is Animal)
        {
            Console.WriteLine("myDog is an Animal!");
        }
    }
}
```

```
myDog is an Animal!
```

همانطور که در مثال بالا می‌بینید ما یک کلاس به نام Dog ایجاد کرده‌ایم که از کلاس Animal ارث می‌برد. سپس یک نمونه از این کلاس (Dog) ایجاد می‌کنیم و سپس با استفاده از عملگر is تست می‌کنیم که آیا نمونه ایجاد شده جزء کلاس Animal است یا یک کلاس مشتق شده از کلاس Animal می‌باشد. از آنجاییکه کلاس Dog از کلاس Animal ارث می‌برد (سگ من یک حیوان است.)، نتیجه عبارت درست (true) است. حال جمله بالا را تغییر دهیم: " حیوان من یک سگ است". وقتی جمله برعکس می‌شود چه اتفاقی می‌افتد؟

```
Animal myAnimal = new Animal();

if (myAnimal is Dog)
{
    Console.WriteLine("myAnimal is a Dog!");
}
```

این باعث خطا نمی‌شود و عبارت فقط نتیجه false را بر می‌گرداند. می‌توان از کد بالا این را درک کرد که همه حیوانات سگ نیستند، ولی همه سگ‌ها حیوان هستند. راه دیگر برای چک کردن نوع یک شیء (object) استفاده از عملگر typeof و متد ()GetType کلاس System.Object است.

```
if (myAnimal.GetType() == typeof(Animal))
{
}
```

متد ()GetType یک شیء از نوع System.Type را بر می‌گرداند که نشان دهنده نوع شیئی که آن را فراخوانی کرده است، می‌باشد. عملگر typeof نام یک نوع را قبول کرده و شیء System.Type متناظر با آن را بر می‌گرداند.

رابط‌ها (Interfaces)

رابط‌ها شبیه به کلاس‌ها هستند، اما فقط شامل تعاریفی برای متدها و خواص (Property) می‌باشند. رابط‌ها را می‌توان به عنوان پلاگین (Plugin)های کلاس‌ها در نظر گرفت. کلاسی که یک رابط خاص را پیاده سازی می‌کند، لازم است که کدهایی برای اجرا توسط اعضاء و متدهای آن فراهم کند چون اعضاء و متدهای رابط هیچ کد اجرایی در بدنه خود ندارند. اجازه دهید که نحوه تعریف و استفاده از یک رابط در کلاس را توضیح دهیم:

```
1  using System;
2
3  interface ISample
4  {
5      void ShowMessage(string message);
6  }
7
8  public class Sample : ISample
9  {
10     public void ShowMessage(string message)
11     {
12         Console.WriteLine(message);
13     }
14 }
15
16 class Program
17 {
18     public static void Main()
19     {
20         Sample sample = new Sample();
21
22         sample.ShowMessage("Implemented the ISample Interface!");
23     }
24 }
```

```
Implemented the ISample Interface!
```

در خطوط ۶-۳ یک رابط به نام ISample تعریف کرده‌ایم. بر طبق قراردادهای نامگذاری، رابط‌ها به شیوه پاسکال نامگذاری می‌شوند و همه آنها باید با حرف I شروع شوند. یک متد در داخل بدنه رابط تعریف می‌کنیم (خط ۵). به این نکته توجه کنید که متد تعریف شده فاقد بدنه است و در آخر آن باید از سمیکالن استفاده شود. وقتی که متد را در داخل رابط تعریف می‌کنید، فقط لازم است که عنوان متد (نوع، نام و پارامترهای آن) را بنویسید. به این نکته نیز توجه کنید که متدها و خواص تعریف شده در داخل رابط سطح دسترسی ندارند، چون باید همیشه هنگام اجرای

کلاس‌ها در دسترس باشند. وقتی یک کلاس نیاز به اجرای یک رابط داشته باشد، از همان روشی که در وراثت استفاده می‌کردیم، استفاده می‌کنیم. کلاسی که رابط را اجرا می‌کند، کدهای واقعی را برای اعضای آن فراهم می‌کند. همانطور که در مثال بالا می‌بینید کلاس Sample، متد ShowMessage() رابط ISample را اجرا و تغذیه می‌کند. برای روشن شدن کاربرد رابط‌ها به مثال زیر توجه کنید:

```
using System;

 class CA
 {
     public string FullName;
     public int    Age;
 }

 class CB
 {
     public string FirstName;
     public string LastName;
     public double PersonsAge;
 }

 class Program
 {
     static void PrintInfo(CA item)
     {
         Console.WriteLine("Name: {0}, Age {1}", item.FullName, item.Age);
     }
     static void Main()
     {
         CA a = new CA() { FullName = "John Doe", Age = 35 };

         PrintInfo(a);

         Console.ReadLine();
     }
 }
```

در کد بالا دو کلاس CA و CB تعریف شده‌اند، در کلاس CA دو فیلد به نام FullName و Age و در کلاس CB سه فیلد به نام‌های FirstName، LastName و PersonsAge تعریف کرده‌ایم. در کلاس Program یک متد به نام PrintInfo() داریم که یک پارامتر از نوع کلاس CA دارد. به شکل ساده در این متد مقدار فیلدهای شیء ای که به این متد ارسال شده است چاپ می‌شود. در متد Main() یک شیء از کلاس CA ساخته‌ایم و فیلدهای آن را مقدار دهی کرده‌ایم. سپس این شیء را به متد PrintInfo() ارسال می‌کنیم. کلاس‌های CA و CB از نظر مفهومی شبیه یکدیگر هستند. مثلاً کلاس CA فیلد FullName را برای نمایش نام و نام خانوادگی دارد ولی کلاس CB برای نمایش نام و نام خانوادگی دو فیلد جدا از هم به نام‌های FirstName و LastName را دارد. و همچنین یک فیلد برای نگهداری مقدار سن داریم که در کلاس CA نام آن Ageو در کلاس CB نام آن PersonAge می‌باشد. مشکل اینجاست که اگر ما یک شیء از کلاس CA را به متد PrintInfo() ارسال کنیم، از آنجایی که در داخل بدنه این متد فقط مقدار دو فیلد چاپ می‌شود، اگر بخواهیم یک شیء از کلاس CB را به آن ارسال کنیم که دارای سه فیلد است با خطا مواجه می‌شویم (زیرا متد PrintInfo با ساختار کلاس CA سازگار است و فیلدهای CB را نمی‌شناسد). برای رفع این مشکل باید ساختار دو کلاس CA و CB را شبیه هم کنیم و این کار را با استفاده از Interface انجام می‌دهیم.

```
using System;

interface IInfo
{
    string GetName();
```

```
6       string GetAge();
7   }
8
9   class CA : IInfo
10  {
11      public string FullName;
12      public int Age;
13      public string GetName() { return FullName;       }
14      public string GetAge()  { return Age.ToString(); }
15  }
16
17  class CB : IInfo
18  {
19      public string FirstName;
20      public string LastName;
21      public double PersonsAge;
22      public string GetName() { return FirstName + " " + LastName; }
23      public string GetAge()  { return PersonsAge.ToString();      }
24  }
25
26  class Program
27  {
28      static void PrintInfo(IInfo item)
29      {
30          Console.WriteLine("Name: {0}, Age {1}", item.GetName(), item.GetAge());
31      }
32
33      static void Main()
34      {
35          CA a = new CA() { FullName = "John Doe", Age = 35 };
36          CB b = new CB() { FirstName = "Jane", LastName = "Doe", PersonsAge = 33 };
37
38          PrintInfo(a);
39          PrintInfo(b);
40
41          Console.ReadLine();
42      }
43  }
```

```
Name: John Doe, Age 35
Name: Jane Doe, Age 33
```

کد بالا را می‌توان به اینصورت توضیح داد که در خط ۳-۷ یک رابط به نام IInfo تعریف و آن را در خطوط ۹ و ۱۷ توسط دو کلاس CA و CB پیاده سازی کرده‌ایم. چون این دو کلاس وظیفه دارند متدهای این رابط را پیاده سازی کنند، پس در خطوط ۱۴-۱۳ و ۲۳-۲۲ کدهای بدنه دو متد این رابط را آن طور که می‌خواهیم، می‌نویسیم. در خط ۲۸ متد PrintInfo() را طوری دستکاری می‌کنیم که یک پارامتر از نوع رابط دریافت کند. حال زمانی که دو شیء از دو کلاس CA و CB در دو خط ۳۵ و ۳۶ ایجاد می‌کنیم و آنها را در دو خط ۳۸ و ۳۹ به متد PrintInfo() ارسال می‌کنیم، چونکه این دو کلاس رابط IInfo را پیاده سازی کرده‌اند، به طور صریح به رابط تبدیل می‌شود. یعنی کلاسی که یک رابط را پیاده سازی کند به طور صریح می‌تواند به رابط تبدیل شود. حال بسته به اینکه شیء کدام کلاس به متد PrintInfo() ارسال شده است، متد مربوط به آن کلاس فراخوانی شده و مقادیر فیلدها چاپ می‌شود. می‌توان چند رابط را در کلاس اجرا کرد:

```
class Sample : ISample1, ISample2, ISample3
{
    //Implement all interfaces
}
```

درست است که می‌توان از چند رابط در کلاس استفاده کرد ولی باید مطمئن شد که کلاس می‌تواند همه اعضای رابط‌ها را تغذیه کند. اگر یک کلاس از کلاس پایه ارث ببرد و در عین حال از رابط‌ها هم استفاده کند، در این صورت باید نام کلاس پایه قبل از نام رابط‌ها ذکر شود. به کد زیر توجه کنید:

```
class Sample : BaseClass, ISample1, ISample2
{
}
```

همچنین می‌توان از عملگر is برای چک کردن اینکه آیا یک شیء خاص از یک رابط استفاده می‌کند یا نه استفاده کرد:

```
Sample sample = new Sample();

if(sample is ISample)
{
   Console.WriteLine("sample implements the ISample Interface!");
}
```

نکته دیگر اینکه نمی‌توان از یک رابط نمونه‌ای ایجاد کرد چون رابط‌ها دارای سازنده نیستند، مثلاً کد زیر اشتباه است:

```
ISample sample = new ISample();
```

کد زیر یک رابط که دارای یک property هست را نشان می‌دهد:

```
interface ISample
{
    int Number { get; set; }
}
```

نباید هیچ کدی در قسمت get و set خاصیت نوشته شود. کلاسی که رابط را پیاده سازی می‌کند آن را اجرا می‌کند.

```
class Sample : ISample
{
    private int number;

    public int Number
    {
        get { return number; }
        set { number = value; }
    }
}
```

رابط‌ها حتی می‌توانند رابط‌های دیگر را پیاده سازی یا اجرا کنند. به مثال زیر توجه کنید:

```
1   using System;
2
3   interface IBase
4   {
5       void BaseMethod();
6   }
7
8   interface ISample : IBase
9   {
10      void ShowMessage(string message);
11  }
12
13  public class Sample : ISample
```

```
{
    public void ShowMessage(string message)
    {
        Console.WriteLine(message);
    }

    public void BaseMethod()
    {
        Console.WriteLine("Method from base interface!");
    }
}

class Program
{
    public static void Main()
    {
        Sample sample = new Sample();

        sample.ShowMessage("Implemented the ISample Interface!");
        sample.BaseMethod();
    }
}
```

مشاهده می‌کنید که حتی اگر کلاس Sample فقط رابط ISample را پیاده سازی کند، لازم است که همه اعضای IBase را هم پیاده سازی کند چون ISample از آن ارث بری می‌کند (خط ۸).

کلاس‌های انتزاعی (Abstract Class)

کلاس‌های مجرد (Abstract) کلاس‌هایی هستند که کلاس پایه سایر کلاس‌ها هستند. این نوع کلاس‌ها می‌توانند مانند کلاس‌های عادی دارای سازنده باشند. شما نمی‌توانید از کلاس‌های انتزاعی نمونه ایجاد کنید؛ چون که هدف اصلی از به کار بردن کلاس‌های انتزاعی استفاده از آنها به عنوان کلاس پایه برای کلاس‌های مشتق است. برای تعریف یک کلاس انتزاعی از کلمه کلیدی abstract استفاده می‌شود. به مثال زیر در مورد استفاده از کلاس‌های انتزاعی توجه کنید:

```
using System;

namespace AbstractClassDemo
{
    public abstract class Base
    {
        protected int number;
        protected string name;

        public abstract int Number
        {
            get;
            set;
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

```

```
22          public abstract void ShowMessage();
23
24          public Base(int number, string name)
25          {
26              this.number = number;
27              this.name = name;
28          }
29      }
30
31      public class Derived : Base
32      {
33          public override void ShowMessage()
34          {
35              Console.WriteLine("Hello World!");
36          }
37
38          public override int Number
39          {
40              get
41              {
42                  return number;
43              }
44              set
45              {
46                  number = value;
47              }
48          }
49
50          public Derived(int number, string name)
51              : base(number, name)
52          {
53          }
54      }
55  }
```

در داخل کلاس انتزاعی دو فیلد محافظت شده (protected) تعریف کرده‌ایم (خطوط ۷ و ۸) که توسط خواص (property) برنامه مورد استفاده قرار می‌گیرند. یکی از propertyها را به صورت انتزاعی (abstract) تعریف کرده‌ایم (خط ۱۰). به این نکته توجه کنید که برای تعریف این خاصیت کلمه کلیدی abstract را به کار برده‌ایم. این property باید به وسیله کلاس‌هایی که از این کلاس ارث می‌برند override شود ولی از آن جاییکه به صورت abstract تعریف شده است قسمت‌های set و get فاقد بدنه هستند. یک متد abstract تعریف می‌کنیم (خط ۲۲). همانطور که مشاهده می‌کنید کلاس‌های Abstract می‌توانند شامل Propertyهای معمولی مانند خاصیت Name (خطوط ۲۰-۱۶) باشند. در کلاس مشتق نیز از کلمه کلیدی abstract استفاده شده و کلاس باید این متد را override کند (به صورت دیگر پیاده سازی کند (خط ۳۳)). کلاس abstract فاقد سازنده است. کلاس‌های Abstract حداقل باید یک عضو Abstract داشته باشند.

یک کلاس دیگر تعریف می‌کنید که از کلاس Base ارث بری کند (خطوط ۵۴-۳۱). سپس با استفاده از کلمه کلیدی override یک خاصیت abstract و همچنین یک متد را به صورت دیگر پیاده سازی یا override می‌کنیم (خطوط ۳۳ تا ۴۸). همچنین یک سازنده تعریف می‌کنیم و با استفاده از کلمه کلیدی base مقادیر پارامترها را به سازنده پایه ارسال می‌کنیم. نمی‌توان از یک کلاس abstract نمونه ایجاد کرد، ولی از کلاس‌هایی که از این نوع کلاس‌ها مشتق می‌شوند، می‌توان نمونه ایجاد کرد.

کلاس‌های مهر و موم شده (Sealed Class)

کلاس مهر و موم شده، کلاسی است که دیگر کلاس‌ها نمی‌توانند از آن ارث بری کنند و چون قابلیت ارث بری ندارد، نمی‌تواند مجرد (abstract) هم باشد. مثال زیر یک کلاس مهر و موم شده را نشان می‌دهد:

```
public sealed class Base
{
    private int someField;

    public int SomeProperty
    {
        get { return someField; }
        set { field = value; }
    }

    public void SomeMethod
    {
        //Do something here
    }

    //Constructor
    public Base()
    {
        //Do something here
    }
}

public class Derived : Base
{
    //This class cannot inherit the Base class
}
```

برای تعریف این کلاس‌ها، از کلمه کلیدی sealed استفاده می‌شود. مشاهده می‌کنید که کلاس‌های مهر و موم شده مانند کلاس‌های عادی، دارای فیلد، خواص و متد می‌باشند. کلاس مشتق در مثال بالا، با خط قرمز نشان داده شده است؛ چون نمی‌تواند از کلاس پایه ارث بری کند. استفاده از این کلاس‌ها همانطور که ذکر شد زمانی مفید است که بخواهید کلاسی ایجاد کنید که دیگر کلاس‌ها نتوانند از آن ارث بری کنند.

کلاس‌های تکه تکه (partial-classes)

استفاده از کلمه کلیدی partial به شما اجازه می‌دهد که یک کلاس را در چندین فایل جداگانه تعریف کنید. به عنوان مثال می‌توانید فیلدها، خاصیت‌ها و سازنده‌ها را در یک فایل و متدها را در فایل دیگر قرار دهید. برای تعریف این نوع کلاس‌ها از کلمه کلیدی partial استفاده می‌شود. در مثال زیر نحوه تعریف یک کلاس partial در دو فایل جدا نشان داده شده است:

```
public partial class Sample
{
    private int sampleField;

    public int SampleProperty
    {
        get { return sampleField; }
        set { sample = value; }
    }
}
public partial class Sample
{
    public void DoSomething()
    {
        //Do something here
    }
}
```

در این مثال مشاهده می‌کنید که برای تعریف کلاس‌ها از کلمه کلیدی partial استفاده شده است. همچنین فایل‌های جدا در یک کلاس به نام Sample تعریف شده‌اند. در قسمت اول (فایل اول) یک فیلد و یک property و در قسمت دوم (فایل دو) یک متد تعریف شده است. هنگام ارث بری از کلاس‌ها یا پیاده سازی رابط‌ها فقط یک بخش از کلاس‌های partial برای ارث بری نیاز است. اگر بخش اول کلاس را IInterface1 و بخش دوم آن را IInterface2 بنامیم در نهایت کلاس اصلی ترکیبی از این دو بخش خواهد بود.

```
public partial class Sample : IInterface1
{
}

public partial class Sample : IInterface2
{
}
```

می‌توان کلاس را به صورت زیر نیز نوشت:

```
public class Sample : IInterface1, IInterface2
{
}
```

از کلاس‌های تکه تکه در فرم‌های ویندوزی و وب فرم‌ها برای جدا کردن فایل‌های کد از فایل‌هایی که در سرتا سر طراحی این فرم‌ها به کار می‌روند، استفاده می‌شود.

چند ریختی

چند ریختی به کلاس‌هایی که در یک سلسله مراتب وراثتی مشابه هستند اجازه تغییر شکل و سازگاری مناسب می‌دهد و همچنین به برنامه نویس این امکان را می‌دهد که به جای ایجاد برنامه‌های خاص، برنامه‌های کلی و عمومی تری ایجاد کند. به عنوان مثال در دنیای واقعی همه حیوانات غذا می‌خورند، اما روش‌های غذا خوردن آنها متفاوت است. در یک برنامه برای مثال، یک کلاس به نام Animal ایجاد می‌کنید. بعد از ایجاد این کلاس می‌توانید آن را چند ریخت (تبدیل) به کلاس Bird کنید و متد Fly() را فراخوانی کنید. به مثالی در باره چند ریختی توجه کنید:

```
using System;

class Animal
```

```
{
    public virtual void Eat()
    {
        Console.WriteLine("The animal ate!");
    }
}

class Dog : Animal
{
    public override void Eat()
    {
        Console.WriteLine("The dog ate!");
    }
}

class Bird : Animal
{
    public override void Eat()
    {
        Console.WriteLine("The bird ate!");
    }
}

class Fish : Animal
{
    public override void Eat()
    {
        Console.WriteLine("The fish ate!");
    }
}

class Program
{
    public static void Main()
    {
        Dog myDog = new Dog();
        Bird myBird = new Bird();
        Fish myFish = new Fish();
        Animal myAnimal = new Animal();

        myAnimal.Eat();

        myAnimal = myDog;
        myAnimal.Eat();
        myAnimal = myBird;
        myAnimal.Eat();
        myAnimal = myFish;
        myAnimal.Eat();
    }
}
```

```
The animal ate!
The dog ate!
The bird ate!
The fish ate!
```

همانطور که مشاهده می‌کنید ۴ کلاس مختلف تعریف کرده‌ایم. `Animal` کلاس پایه است و سه کلاس دیگر از آن مشتق می‌شوند. هر کلاس متد `Eat()` مربوط به خود را دارد. نمونه‌ای از هر کلاس ایجاد کرده‌ایم (۴۲-۳۹). حال متد `Eat()` را به وسیله نمونه ایجاد شده از کلاس `Animal` به صورت زیر فراخوانی می‌کنیم:

```
Animal myAnimal = new Animal();

myAnimal.Eat();
```

در مرحله بعد چندریختی روی می‌دهد. همانطور که در مثال بالا مشاهده می‌کنید، شیء Dog را برابر شیء ایجاد شده از کلاس Animal قرار می‌دهیم (خط ۴۶) و متد ()Eat را بار دیگر فراخوانی می‌کنیم. حال با وجود اینکه ما از نمونه کلاس Animal استفاده کرده‌ایم ولی متد ()Eat کلاس Dog فراخوانی می‌شود. این به دلیل تأثیر چند ریختی است.

سپس دو شیء دیگر (Bird و Fish) را برابر نمونه ایجاد شده از کلاس Animal قرار می‌دهیم و متد ()Eat مربوط به هر یک را فراخوانی می‌کنیم (خطوط ۵۱-۴۸). به این نکته توجه کنید که وقتی در مثال بالا اشیاء را برابر نمونه کلاس Animal قرار می‌دهیم از عمل Cast استفاده نکرده‌ایم، چون این کار (cast)، وقتی که بخواهیم یک شیء از کلاس مشتق (مثلاً Dog) را در شیی از کلاس پایه (Animal) ذخیره کنیم، لازم نیست. همچنین می‌توان کلاس Animal را با سازنده هر کلاس مشتق دیگر مقدار دهی اولیه کرد:

```
Animal myDog = new Dog();
Animal myBird = new Bird();
Animal myFish = new Fish();

myDog.Eat();
myBird.Eat();
myFish.Eat();
```

اجازه دهید که برنامه بالا را اصلاح کنیم تا مفهوم چند ریختی را بهتر متوجه شوید:

```
1   using System;
2
3   class Animal
4   {
5       public virtual void Eat()
6       {
7           Console.WriteLine("The animal ate!");
8       }
9   }
10
11  class Dog : Animal
12  {
13      public override void Eat()
14      {
15          Console.WriteLine("The dog ate!");
16      }
17
18      public override void Run()
19      {
20          Console.WriteLine("The dog ran!");
21      }
22  }
23
24  class Bird : Animal
25  {
26      public override void Eat()
27      {
28          Console.WriteLine("The bird ate!");
29      }
30
31      public override void Fly()
32      {
33          Console.WriteLine("The bird flew!");
34      }
35  }
36
37  class Fish : Animal
38  {
```

```
    public override void Eat()
    {
        Console.WriteLine("The fish ate!");
    }

    public override void Swim()
    {
        Console.WriteLine("The fish swam!");
    }
}

class Program
{
    public static void Main()
    {
        Animal animal1 = new Dog();
        Animal animal2 = new Bird();
        Animal animal3 = new Fish();

        Dog myDog = (Dog)animal1;
        Bird myBird = (Bird)animal2;
        Fish myFish = (Fish)animal3;

        myDog.Run();
        myBird.Fly();
        myFish.Swim();
    }
}
```

```
The dog ran!
The bird flew!
The fish swam!
```

در بالا سه شیء از کلاس Animal ایجاد و آنها را بوسیله سه سازنده از کلاس‌های مشتق مقدار دهی اولیه کرده‌ایم (خطوط ۵۶-۵۴). سپس با استفاده از عمل cast اشیاء ایجاد شده از کلاس Animal را در نمونه‌هایی از کلاس‌های مشتق ذخیره می‌کنیم (خطوط ۶۰-۵۸). وقتی این کار را انجام دادیم، می‌توانیم متدهای مخصوص به هر یک از کلاس‌های مشتق را فراخوانی کنیم (خطوط ۶۴-۶۲). یک راه میانبر دیگر به وسیله کد زیر مشخص شده است. ولی در این روش شما نمی‌توانید اشیاء ایجاد شده از کلاس Animal را در نمونه‌هایی از کلاس‌های مشتق، ذخیره می‌کنید.

```
((Dog)animal1).Run();
((Bird)animal2).Fly();
((Fish)animal3).Swim();
```

از چند ریختی می‌توان در رابط‌ها هم استفاده کرد. به کد زیر توجه کنید:

```
using System;

interface IAnimal
{
    void Eat();
}

class Dog : IAnimal
{
    public void Eat()
    {
        Console.WriteLine("The dog ate!");
    }
}

class Bird : IAnimal
```

```
{
    public void Eat()
    {
        Console.WriteLine("The bird ate!");
    }
}

class Fish : IAnimal
{
    public void Eat()
    {
        Console.WriteLine("The fish ate!");
    }
}

class Program
{
    public static void Main()
    {
        IAnimal myDog = new Dog();
        IAnimal myBird = new Bird();
        IAnimal myFish = new Fish();

        myDog.Eat();
        myBird.Eat();
        myFish.Eat();
    }
}
```

```
The dog ate!
The bird ate!
The fish ate!
```

تسلط کامل بر چند ریختی و وراثت برای درک بهتر دات‌نت ضروری است.

عملگر as

از عملگر as برای تبدیل یک کلاس به کلاس دیگر که در سلسله مراتب وراثتی یکسانی هستند، استفاده می‌شود. این عملگر، کاری معادل تبدیل صریح انجام می‌دهد و فقط دارای تفاوتی جزئی هستند که در ادامه توضیح می‌دهیم. نحوه استفاده از عملگر as به صورت زیر است:

```
myObject as DestinationType;
```

عملگر سمت چپ شیی است که قرار است تبدیل شود و عملگر سمت راست نوع مقصد است که قرار است شیء به آن تبدیل شود. کدهای زیر با هم برابر هستند:

```
Destination someObject = (Destination)myObject;
Destination someObject = myObject as Destination;
```

در کد اول از عمل cast (تبدیل صریح) استفاده شده است و اگر تبدیل دو کلاس با شکست مواجه شود باعث به وجود آمدن استثناء می‌شود. در کد دوم از عملگر as استفاده شده است و اگر تبدیل با شکست مواجه شود، مقدار تهی (null) را بر می‌گرداند. شما می‌توانید به وسیله عملگر as یک متد از کلاس مشتق را از طریق کلاس پایه فراخوانی کنید. به مبحث کلاس‌ها در درس قبل مراجعه نمایید.

```
(Animal as Dog).Run();
```

سربارگذاری تبدیل‌ها

تبدیل‌ها را در سی‌شارپ می‌توان سربارگذاری کرد. به عنوان مثال برای تبدیل یک کلاس به یک کلاس غیر مرتبط دیگر، می‌توان از سربارگذاری تبدیل‌ها (صریح و ضمنی) استفاده کرد. مثال زیر نشان می‌دهد که چطور می‌توان عملگرهای تبدیل را سربارگذاری کرد.

```
using System;

class Animal
{
    public int Height { get; set; }

    public static implicit operator Plant(Animal animal)
    {
        Plant result  = new Plant();
        result.Height = animal.Height;
        return result;
    }
}

class Plant
{
    public int Height { get; set; }

    public static explicit operator Animal(Plant plant)
    {
        Animal result = new Animal();
        result.Height = plant.Height;
        return result;
    }
}

class Program
{
    public static void Main()
    {
        Animal myAnimal = new Animal();
        myAnimal.Height = 100;

        //Implicit conversion
        Plant myPlant = myAnimal;

        Console.WriteLine("myAnimal.Height = {0}cm", myAnimal.Height);
        Console.WriteLine("myPlant.Height = {0}cm", myPlant.Height);

        myPlant.Height = 200;

        //Explicit conversion
        myAnimal = (Animal)myPlant;

        Console.WriteLine("\nmyAnimal.Height = {0}cm", myAnimal.Height);
        Console.WriteLine("myPlant.Height = {0}cm", myPlant.Height);
    }
}
```

```
myAnimal.Height = 100cm
myPlant.Height = 100cm

myAnimal.Height = 200cm
myPlant.Height = 200cm
```

در مثال بالا مشاهده می‌کنید که دو کلاس `Animal` و `Planet` هیچ کدام از دیگری ارث بری نمی‌کنند و درنتیجه هیچ ارتباطی به هم ندارند. برای هر کلاس یک سربارگذاری تبدیل هم تعریف کرده‌ایم. کلاس `Animal` تبدیل ضمنی خود به کلاس `Plant` را سربارگذاری می‌کند. به این نکته توجه

کنید که برای مشخص کردن اینکه تبدیل ضمنی است یا صریح، به ترتیب از کلمه کلیدی implicit و explicit استفاده کرده‌ایم، و کلمه کلیدی operator نشان دهنده این است که متد برای سربارگذاری مورد استفاده قرار می‌گیرد.

نام کلاس مقصد را می‌نویسیم. وقتی یک شیء از Animal را به یک شیء Plant اختصاص می‌دهیم، کد داخل متد سربارگذاری شده اجرا می‌شود. متد یک پارامتر دارد که ارجاع به نمونه کلاسی است که باید تبدیل شود. این متد همچنین، نه دارای نوع برگشتی هست و نه نام، در ضمن باید به صورت public و static تعریف شود.

در داخل متد، یک نمونه از کلاس Animal را ساخته و مقدار خاصیت Height شیء Plant ارسالی به متد را به خاصیت Height شیء نسبت می‌دهیم. اساساً، شما باید کدهای لازم برای انتقال مقادیر لازم از شیء مبدأ به شیء مقصد را بنویسید. در متد دوم، از کلمه کلیدی explicit به جای implicit استفاده کرده‌ایم. این کلمه نشان دهنده تبدیل صریح است. دستور زبان نوشتن این تبدیل شبیه به تبدیل قبلی است، با این تفاوت که به جای کلمه کلیدی implicit از explicit استفاده کنید. در داخل این متد از کدهای مشابهی برای سربارگذاری استفاده کرده‌ایم.
به تبدیل ضمنی کلاس Animal به Plant توجه فرمایید:

```
Plant myPlant = myAnimal;
```

نمی‌توانیم مکان قرارگیری آنها را بر عکس نماییم :

```
Animal myAnimal = myPlant;
```

چون که متدی برای تبدیل ضمنی کلاس Plant به Animal تعریف نشده است. اما چون برای تبدیل صریح متدی را تعریف کرده‌ایم، می‌توانیم کد را به شکل زیر تغییر دهیم:

```
Animal myAnimal = (Animal)myPlant;
```

برای انجام تبدیل صریح از casting استفاده کرده‌ایم.

ایجاد آرایه ای از کلاس‌ها

در این درس به شما نشان می‌دهیم که چگونه می‌توان آرایه ای از کلاس‌ها ایجاد کرد. ساخت آرایه ای از کلاس‌ها تقریباً شبیه به ایجاد آرایه ای از انواع داده‌ای مانند int است. به عنوان مثال می‌توان آرایه ای از کلاس Person ایجاد کرد:

```
using System;

public class Person
{
    public string Name { get; set; }

    public Person(string name)
    {
        Name = name;
    }
}
```

```
public class Program
{
    public static void Main()
    {
        Person[] people = new Person[3];

        people[0] = new Person("Johnny");
        people[1] = new Person("Mike");
        people[2] = new Person("Sonny");

        foreach (Person person in people)
        {
            Console.WriteLine(person.Name);
        }
    }
}
```

```
Johnny
Mike
Sonny
```

ابتدا یک کلاس که دارای یک property است، تعریف می‌کنیم. سپس یک آرایه از آن (کلاس ایجاد شده) تعریف می‌کنیم و در نهایت عناصر آن را مانند بالا مقدار دهی می‌کنیم. سپس مقدار property هر یک از نمونه‌ها را با استفاده از یک حلقه foreach نمایش می‌دهیم. می‌توان از تکنیک‌های دیگر که قبلاً در مورد ایجاد آرایه آموختید هم استفاده کنید. مثلاً، مثال بالا را می‌توان به صورت زیر هم نوشت:

```
Person[] people = new Person[]
{
    new Person("Johnny"),
    new Person("Mike"),
    new Person("Sonny")
};
```

در اینجا، تعداد عناصر آرایه people، ۳ می‌باشد و کامپایلر هم با شمارش تعداد نمونه‌ها، آن را تشخیص می‌دهد. از این تکنیک برای ساخت آرایه‌های چند بعدی و دندانه دار هم می‌توان استفاده کرد.

ایندکسرها

فرض کنید یک کلاس به نام Employee، که دارای سه فیلد از نوع رشته است، تعریف کرده‌ایم. همانطور که در داخل متد Main() مشاهده می‌کنید می‌توان با استفاده از نام فیلدها به آنها دست یافت.

```
using System;

public class Employee
{
    public string LastName;                         // Call this field 0.
    public string FirstName;                        // Call this field 1.
    public string CityOfBirth;                      // Call this field 2.
}

class Program
{
    static void Main()
    {
        Employee emp1 = new Employee();
```

```
        emp1.LastName = "Doe";
        emp1.FirstName = "Jane";
        emp1.CityOfBirth = "Dallas";

        Console.WriteLine("{0}", emp1.LastName);
        Console.WriteLine("{0}", emp1.FirstName);
        Console.WriteLine("{0}", emp1.CityOfBirth);
    }
}
```

```
Doe
Jane
Dallas
```

اما در نظر بگیرید اگر با نمونه ایجاد شده از کلاس مانند آرایه ای از فیلدها رفتار شود که در این صورت دسترسی به فیلدها با استفاده از اندیس راحت تر است. این دقیقاً کاری است که ایندکسرها به شما اجازه انجام آن را می‌دهند. اگر بخواهید یک ایندکسر برای کلاس Employee تعریف کنید، باید کد داخل متد Main() را به صورت زیر تعییر دهید. به این نکته توجه کنید که نمونه از علامت دات (.) برای دستیابی به فیلدها استفاده می‌کند ولی ایندکسرها از اندیسی که در داخل کروشه وجود دارند.

```
static void Main()
{
   Employee emp1 = new Employee();

   emp1[0] = "Doe";
   emp1[1] = "Jane";
   emp1[2] = "Dallas";

   Console.WriteLine( "{0}", emp1[0] );
   Console.WriteLine( "{0}", emp1[1] );
   Console.WriteLine( "{0}", emp1[2] );
}
```

تعریف ایندکسر

دستور تعریف ایندکسر به صورت زیر است:

```
ReturnType this [ Type param1, ... ]
{
       get
       {
           ...
       }
       set
       {
           ...
       }
}
```

به نکاتی در مورد ایندکسرها توجه کنید:

- ایندکسرها اسم ندارند و به جای اسم باید کلمه کلیدی this قرار داده شود.
- پارامترها بین کروشه قرار می‌گیرند.
- باید حداقل یک پارامتر داشته باشد.
- از آنجاییکه همواره یک عضو نمونه است، نباید به صورت static تعریف شود.

تعریف ایندکسر شبیه به تعریف خاصیت است. تفاوت و شباهت این دو به صورت زیر است:

بر خلاف خاصیت، ایندکسر:

- به جای نام دارای کلمه کلیدی this است.
- دارای لیستی از پارامترها است که در داخل یک جفت کروشه قرار دارند.

مانند خاصیت، ایندکسر:

- دارای نوع است.
- دارای متدهای get و set است.

به یک مثال در مورد ایندکسر توجه کنید:

```
using System;

namespace Indexers
{
    public class Person
    {
        private string[] friends;

        //Constructor to add friends
        public Person(params string[] friends)
        {
            this.friends = friends;
        }

        //This is the indexer
        public string this[int index]
        {
            get { return friends[index]; }
            set { friends[index] = value; }
        }
    }

    public class Program
    {
        public static void Main()
        {
            //Create a new person and add some friends
            Person person = new Person("Jenny", "Robert", "Susan", "Charles", "Mandy");

            for (int i = 0; i < 5; i++)
            {
                //You can now access each element of the
                //  private field by specifying the index
                Console.WriteLine(person[i]);
            }

            //Changing the second element to John
            person[1] = "John";
        }
    }
}
```

```
Jenny
Robert
Susan
```

```
Charles
Mandy
```

در کد بالا، یک کلاس که شامل یک آرایه خصوصی است (خط ۷) و مقادیر friends مربوط به شیء Person (خط ۲۸) را در خود ذخیره می‌کند، ایجاد کرده‌ایم. سپس یک سازنده که مقادیر رشته‌ای را دریافت کرده و آنها را به فیلد friends تخصیص می‌دهد تعریف می‌کنیم (خطوط ۱۳-۱۰). حال نوبت به تعریف ایندکسر (خطوط ۲۰-۱۶) برای کلاس می‌رسد. در داخل بدنه get مقدار یک عنصر آرایه را به وسیله اندیس آن برگشت می‌دهیم. در داخل بدنه set می‌توان به وسیله اندیس یک مقدار را به یک عنصر تخصیص داد.

سپس یک نمونه از کلاس ایجاد کرده و مقادیری را به این نمونه تخصیص می‌دهیم (خط ۲۸). همه عناصر فیلد خصوصی friend نمونه را با استفاده از یک حلقه for پرس و جو می‌کنیم (خطوط ۳۵-۳۰). به این نکته توجه کنید که ما از هیچ خاصیتی برای به دست آوردن مقادیر استفاده نکرده‌ایم، فقط به سادگی و با استفاده از اندیس بعد از نام نمونه (person[i]) به مقادیر عناصر دست یافته‌ایم (درست شبیه به دستیابی به عناصر یک آرایه). همچنین می‌توان از تکنیکی مشابه برای تخصیص مقادیر به یک آرایه خصوصی یا فیلدهای مجموعه‌ای (collection fields) استفاده نمود. بدون ایندکسر، مجبوریم یک خاصیت عمومی برای فیلد friends ایجاد کنیم.

```
public string[] Friends
{
    get { return friends; }
    set { friends = value; }
}
```

و به فیلد آرایه خصوصی به وسیله خاصیت بالا دست یابید:

```
Console.WriteLine(person1.Friends[0]);
person1.Friends[1] = "John";
```

یکی از معایب ایندکسر این است که شما فقط می‌توانید فقط یک ایندکسر برای هر کلاس ایجاد کنید. به عنوان مثال، اگر شما دارای دو فیلد خصوصی به نام friends و classmates، لازم است انتخاب کنید که ایندکسر برای کدام فیلد خصوصی به کار رود. بهتر است که ایندکسر برای یک فیلد آرایه ای که برای کلاس بیشتر مفید است به کار رود و سپس یک خاصیت عمومی برای همه آرایه‌های خصوصی و کلکسیون‌ها تعریف کنید. به این نکته توجه کنید که می‌توان از یک نوع رشته‌ای به عنوان اندیس استفاده کنید و این زمانی مورد استفاده قرار می‌گیرد که در داخل کلاس یک دیکشنری داشته باشیم. در مورد دیکشنری در درس‌های آینده توضیح می‌دهیم.

String Interpolation

String interpolation یا الحاق رشته‌ها به شما اجازه می‌دهد که عبارات رشته‌ای با خوانایی بیشتر ایجاد کنید. در نسخه‌های قبلی سی‌شارپ از متد string.format() برای الحاق رشته‌ها و در نسخه ۶ از ویژگی String interpolation برای این کار استفاده می‌شود. دستور استفاده از این ویژگی جدید به صورت زیر است:

```
$"some text {expression} some other text"
```

همانطور که در کد بالا مشاهده می‌کنید، برای الحاق رشته‌ها قبل از هر رشته یک علامت $ قرار می‌دهیم. در داخل رشته هم یک یا چند عبارت را که در داخل آکولاد هستند، می‌نویسیم. این عبارت می‌تواند یک متغیر و یا حاصل یک یا چند متغیر باشد. هر چیز در داخل آکولاد ارزیابی شده و با استفاده از متد ToString() به رشته تبدیل و در داخل رشته قرار داده می‌شود. اگر بخواهید خود آکولادها هم پردازش شده و در نتیجه نهایی نمایش داده شوند باید از دو علامت آکولاد استفاده کنید ({{ and }}). به مثال زیر توجه کنید:

```
1  public class Program
2  {
3      public static void Main(string[] args)
4      {
5          string message = "Hello World";
6
7          string interpolatedString = $"The message is {message}";
8
9          Console.WriteLine(interpolatedString);
10     }
11 }
```

```
The message is Hello World
```

در خط ۵ یک رشته تعریف شده است. می‌خواهیم این رشته را به رشته دیگر بچسبانیم. برای این کار در خط ۷ و با استفاده از ویژگی string interpolation این کار را انجام داده‌ایم. در این خط یک متغیر که نتیجه الحاق دو رشته در آن قرار می‌گیرد را تعریف کرده‌ایم. همانطور که مشاهده می‌کنید متغیر تعریف شده در خط ۵ را در خط ۷ داخل علامت آکولاد قرار داده‌ایم. مقدار این متغیر در داخل عبارت موجود در خط ۷ قرار می‌گیرد. در مثال زیر نحوه الحاق چندین رشته نمایش داده شده است

```
public class Program
{
    public static void Main(string[] args)
    {
        string firstName = "John";
        string lastName  = "Smith";
        int age = 20;

        Console.WriteLine($"His name is {firstName} {lastName} and his age is {age}.");
    }
}
```

```
His name is John Smith and his age is 20.
```

همانطور که در مثال بالا مشاهده می‌کنید، با استفاده از این ویژگی می‌توانیم متغیرهای غیر رشته‌ای را هم به رشته‌ها بچسبانیم. در مثال زیر تقریباً تمام حالات ممکن در الحاق رشته آمده است:

```
1  using System;
2
3  namespace StringInterpolation
4  {
5      class Person
6      {
7          public string FirstName { get; set; }
8          public string LastName { get; set; }
9
10         public Person()
11         {
12
13         }
```

```
    }

    class Program
    {
        static void Main(string[] args)
        {
            int num1 = 10;
            int num2 = 20;
            string message = "hello world";
            Person person = new Person { FirstName = "John", LastName = "Smith" };

            Console.WriteLine($"The sum of num1 and num2 is {num1 + num2}");
            Console.WriteLine($"num1 less 1 is {num1 - 1}");
            Console.WriteLine($"num2 divided by num1 is {num2 / num1}");

            Console.WriteLine($"num1 is equal to num2? {num1 == num2}");

            Console.WriteLine($"{message} in all caps is {message.ToUpper()}");
            Console.WriteLine($"num1 added by 100 is {Add100(num1)}");

            Console.WriteLine($"Full name is {person.FirstName}" + " " + $"{person.LastName}");
        }

        private static int Add100(int number)
        {
            return number + 100;
        }
    }
}
```

```
The sum of num1 and num2 is 30
num1 less 1 is 9
num2 divided by num1 is 2
num1 is equal to num2? False
hello world in all caps is HELLO WORLD
num1 added by 100 is 110
Full name is John Smith
```

در خطوط ۲۷ – ۲۵ از عبارات ساده ریاضی به عنوان عبارات الحاقی و در خط ۲۹ از دستور شرطی که مقدار true یا false را بر می‌گرداند استفاده کرده‌ایم. در خطوط ۳۲ و ۳۱ از متدهایی استفاده شده که مقدار برگشتی از آنها در عبارات الحاقی قرار می‌گیرند. در خط ۳۴ روش قدیمی الحاق رشته‌ها به کار برده شده است، که شما می‌توانید از عبارات الحاقی به جای آن استفاده کنید.

قالب بندی عبارات الحاقی

می‌توان نتیجه یک عبارت الحاقی را قالب بندی کرد. برای این کار از کاراکترهای خاص که در درس قالب بندی رشته‌ها و اعداد آمده‌اند می‌توان به صورت زیر استفاده کرد:

```
$"some text {expression:format} some other text"
```

به عنوان مثال:

```
public class Program
{
    public static void Main(string[] args)
    {
        double number = 100.1234;
        DateTime today = DateTime.Now;
```

```
7
8           Console.WriteLine($"Formatted to currency: {number:C}");
9           Console.WriteLine($"2 decimal places {number:F2}");
10          Console.WriteLine($"As percentage: {(number / 100):P}");
11
12          Console.WriteLine($"Current Month {today:MM}");
13          Console.WriteLine($"Current Day {today:dd}");
14          Console.WriteLine($"Current Hour {today:hh}");
15
16          Console.WriteLine($"Date today is {today:MM/dd/yyyy hh:mm:ss}");
17      }
18  }
```

```
Formatted to currency: $100.12
2 decimal places 100.12
As percentage: 100.12 %
Current Month 08
Current Day 15
Current Hour 04
Date today is 08/15/2015 04:48:28
```

لیست کاراکترهای قالب بندی اعداد و تاریخ در لینک‌های زیر آمده است:

```
https://msdn.microsoft.com/en-us/library/dwhawy9k(v=vs.110).aspx
```

مشخص کردن طول فیلدها

تعیین طول فیلدها برای تراز بندی مقادیر مفید است، مخصوصاً اگر بخواهید که آنها را در ستون‌های مختلفی نمایش دهید:

```
$"some text {expression, fieldWidth} some other text"
```

این یک مثال ساده بود. فرض کنید که می‌خواهیم یک رشته یا عدد را تراز کنیم. برای اینکار می‌توانی از اعداد مثبت و منفی به صورتی که در مثال زیر آمده است، استفاده کنیم. عدد مثبت باعث اضافه شدن فضای خالی به سمت راست و عدد منفی باعث اضافه شدن فضای خالی به سمت چپ عدد یا رشته می‌شود. البته این تعداد فضای خالی به طول رشته یا عدد بستگی دارد. فرض کنید که یک رشته به طول ۵ کاراکتر داریم و عدد ۱۰ را برای اضافه کردن فضای خالی به سمت راست آن، به کار می‌بریم. در اینصورت ۵ = ۵ – ۱۰ کاراکتر به سمت راست آن اضافه می‌شود. یعنی تعداد فضاهای خالی اضافه شده از تفاضل طول فیلد و طول رشته به دست می‌آید:

```
public static void Main(string[] args)
{
    int number = 100;

    Console.WriteLine("Using field width of 10.");
    Console.WriteLine($"Start {number, 10} End");

    Console.WriteLine("\nUsing field width of -10.");
    Console.WriteLine($"Start {number,-10} End");
}
```

```
Using field width of 10.
Start 100        End

Using field width of -10.
Start        100 End
```

در مثال بالا طول رشته "۱۰۰" سه کاراکتر است. بنابراین ۷ فضای خالی به سمت راست یا چپ آن اضافه می‌شود. می‌توان از ترازبندی و قالب بندی به طور همزمان در یک عبارت الحاقی استفاده کرد:

```
$"some text {expression, fieldWidth:format} some other text"
```

در مثال زیر نحوه استفاده از ترازبندی و قالب بندی به طور همزمان نشان داده شده است:

```
using System;
using System.Collections.Generic;

namespace StringInterpolation
{
    public class Program
    {
        public static void Main(string[] args)
        {
            List<Product> products = new List<Product>
            {
                new Product { ID = 1, Name = "Soap", Price = 5 },
                new Product { ID = 2, Name = "Toothpaste", Price = 20 },
                new Product { ID = 3, Name = "Portal Gun", Price = 999999999 }
            };

            // Print the header
            Console.WriteLine($"|{ "Code",-5}|{ "Name",-15 }|{ "Price", 20}|");

            foreach(var p in products)
            {
                Console.WriteLine($"|{ p.ID, -5 }|{ p.Name, -15 }|{ p.Price, 20:C}|");
            }
        }
    }

    public class Product
    {
        public int ID { get; set;}
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
}
```

```
|Code |Name           |               Price|
|1    |Soap           |              $5.00|
|2    |Toothpaste     |             $20.00|
|3    |Portal Gun     |    $999,999,999.00|
```

مدیریت استثناءها و خطایابی

بهترین برنامه نویسان در هنگام برنامه‌نویسی با خطاها و باگ‌ها در برنامه‌شان مواجه می‌شوند. درصد زیادی از برنامه‌ها هنگام تست برنامه، با خطا مواجه می‌شوند. بهتر است برای از بین بردن یا به حداقل رساندن این خطاها، به کاربر در مورد دلایل به وجود آمدن آنها اخطار داده شود. خوشبختانه سی‌شارپ برای این مشکل راه حلی ارائه داده است.

دات نت دارای مجموعه بزرگی از کلاس‌هایی است که برای برطرف کردن خطاهای خاص از آنها استفاده می‌کند. استثناءها در دات‌نت راهی برای نشان دادن دلیل وقوع خطا در هنگام اجرای برنامه است. دات‌نت دارای مجموعه بزرگی از کلاس‌های استثناء است که شما می‌توانید با استفاده از آنها خطاهایی که در موقعیت‌های مختلف روی می‌دهند، را برطرف کنید. حتی می‌توانید یک کلاس استثناء شخصی ایجاد کنید. استثناءها، توسط

برنامه به وجود می‌آیند و شما لازم است که آنها را اداره کنید. به عنوان مثال در دنیای کامپیوتر، یک عدد صحیح هرگز نمی‌تواند بر صفر تقسیم شود. اگر بخواهید این کار را انجام دهید (یک عدد صحیح را بر صفر تقسیم کنید)، با خطا مواجه می‌شوید. اگر یک برنامه در سی‌شارپ با چنین خطایی مواجه شود پیغام خطای " DivideByZeroException " نشان داده می‌شود که بدین معنا است که عدد را نمی‌توان بر صفر تقسیم کرد.

باگ (Bug) اصطلاحاً خطا یا کدی است که رفتارهای ناخواسته‌ای در برنامه ایجاد می‌کند. خطایابی فرایند برطرف کردن باگ‌ها است، بدین معنی که خطاها را از برنامه پاک کنیم. ویژوال استودیو و ویژوال سی‌شارپ دارای ابزارهایی برای خطایابی هستند، که خطاها را یافته و به شما اجازه می‌دهند آنها را برطرف کنید. در درس‌های آینده خواهید آموخت که چگونه از این ابزارهای کارآمد جهت برطرف کردن باگ‌ها استفاده کنید. قبل از اینکه برنامه را به پایان برسانید، لازم است که برنامه‌تان را اشکال زدایی کنید.

استثناءهای اداره نشده

استثناءهای اداره نشده، استثناءهایی هستند که به درستی توسط برنامه اداره نشده‌اند و باعث می‌شوند که برنامه به پایان برسد. در اینجا می‌خواهیم به شما نشان دهیم که وقتی یک برنامه در زمان اجرا با یک استثناء مواجه می‌شود و آن را اداره نمی‌کند چه اتفاقی می‌افتد. در آینده خواهید دید که یک استثناء چگونه به صورت بالقوه باعث نابودی جریان و اجرای برنامه شما می‌شود. از ابتدای آموزش تا کنون برای اجرای برنامه‌ها و نمونه کدها از حالت Non-Debug (بدون خطایابی) استفاده کرده‌ایم. اجرا کردن یک برنامه بدون خطا در حالت Non-Debug یا Debug دارای تفاوت‌های جزئی می‌باشد. قصد داریم این تفاوت را برای شما توضیح دهیم. یک برنامه جدید کنسول ایجاد کرده و نام آن را ExceptionTest می‌گذاریم.

```
using System;

namespace ExceptionTest
{
    public class Program
    {
        public static void Main()
        {
            int five = 5;
            int zero = 0;

            //Generate an exception by dividing 5 by 0
            int result = five / zero;
        }
    }
}
```

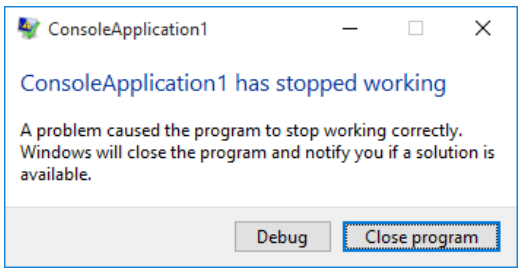
همانطور که در مثال بالا مشاهده می‌کنید تقسیم یک عدد صحیح بر صفر غیر مجاز است و باعث ایجاد خطای System.DivideByZeroException می‌شود.

حالت بدون اداره کردن استثناء (حالت Non-Debug)

برنامه را در حالت Non-Debug به وسیله کلیدهای ترکیبی Ctrl+F5 اجرا می‌کنیم. برنامه با موفقیت اجرا شده ولی با پیغام خطای زیر مواجه می‌شوید:

```
 Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
 at ExceptionTest.Program.Main() in C:UsersTheUserAppDataLocalTemporary Proje
ctsExceptionTestProgram.cs:line 9
```

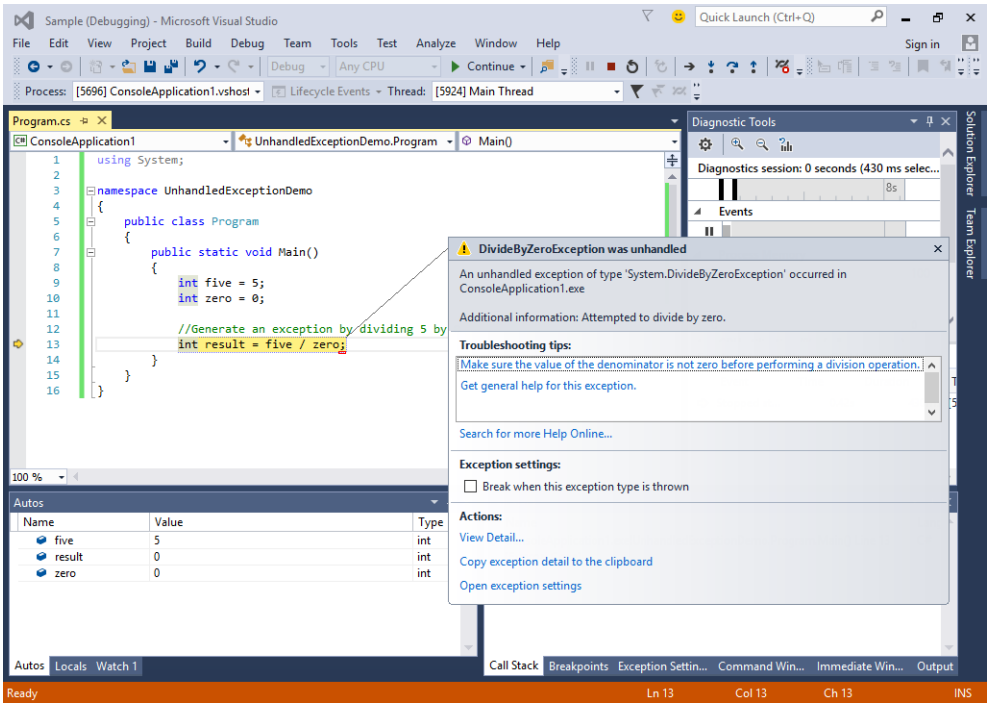
و بلافاصله پنجره زیر بعد از بستن برنامه نمایش داده می‌شود:



نمایش پنجره بالا به دلیل وجود یک استثناء اداره نشده است. چون که شما از هیچ کدام از تکنیک‌های اداره استثناء، استفاده نکرده‌اید. معمولاً جزییات بالا برای کسی که از برنامه شما استفاده می‌کند، لازم نیست.

حالت اداره کردن استثناء(Debug Mode)

یکی از راه‌های بهتر برای دیدن اطلاعاتی در مورد استثناءهای اداره نشده استفاده از حالت Debug است. برای استفاده از این حالت برنامه را از مسیر Debug > Start Debugging اجرا نمایید. همچنین می‌توانید از دکمه F5 و یا فلش سبزرنگ واقع در toolbar استفاده نمایید. به وسیله هر یک از سه حالت بالا برنامه در حالت Debug اجرا می‌شود. در طول اجرای برنامه در این حالت اگر برنامه در قسمت‌هایی دارای ایراد باشد متوقف شده و خطاهای آن با رنگ زرد نمایش داده می‌شود و پنجره دستیار استثناء (Exception Assistant) نمایان می‌گردد.



پنجره دستیار استثناء پنجره‌ای مفید است که در مورد استثناء و چگونگی برطرف کردن آن اطلاعاتی در اختیار شما می‌گذارد. اگر این پنجره مخفی شد، به سادگی و با کلیک بر روی دستوری که دارای خطا است، دوباره ظاهر می‌شود.

دستورات try و catch

می‌توان خطاها را با استفاده از دستور try…catch اداره کرد. بدین صورت که کدی را که احتمال می‌دهید ایجاد خطا کند، در داخل بلوک try قرار می‌دهید. بلوک catch هم، شامل کدهایی است که وقتی اجرا می‌شوند که برنامه با خطا مواجه شود. تعریف ساده‌ی این دو بلوک به این صورت است که بلوک try سعی می‌کند که دستورات را اجرا کند و اگر در بین دستورات خطایی وجود داشته باشد، برنامه، دستورات مربوط به بخش catch را انجام می‌دهد. برنامه زیر نحوه استفاده از دستور try…catch را نمایش می‌دهد:

```
using System;

public class Program
{
    public static void Main()
    {
        int result;
        int x = 5;
        int y = 0;

        try
        {
            result = x / y; //ERROR
        }
        catch
        {
            Console.WriteLine("An attempt to divide by 0 was detected.");
        }
    }
}
```

```
An attempt to divide by 0 was detected.
```

در داخل بلوک try، مقدار x را که ۵ است بر y که مقدار آن ۰ است تقسیم کرده‌ایم. نتیجه محاسبه به وجود آمدن خطای DivideByZeroException (عدد تقسیم بر صفر) است. از آنجاییکه در برنامه بالا خطایی به وجود آمده است، کدهای داخل بلوک catch اجرا می‌شوند. بنابراین:

```
try
{
    result = x / y; //Error: Jump to catch block
    Console.WriteLine("This line will not be executed.");
}
catch
{
    Console.WriteLine("An attempt to divide by 0 was detected.");
}
```

می‌توانید از یک نوع استثناء مخصوص به یک خطا در داخل بلوک catch استفاده کنید، مثلاً برای خطای تقسیم عدد بر صفر از DivideByZeroException به شکل زیر استفاده کنید:

```
try
```

```
{
    result = x / y; //ERROR
}
catch (DivideByZeroException)
{
    Console.WriteLine("An attempt to divide by 0 was detected.");
}
```

همچنین می‌توانید مقدار استثناء را در داخل یک متغیر قرار داده و سپس آن را نمایش دهید:

```
try
{
    result = x / y; //ERROR
}
catch (DivideByZeroException error)
{
    Console.WriteLine(error.Message);
}
```

```
Attempted to divide by zero.
```

متغیر دارای اطلاعات مفیدی در مورد استثناء به وجود آمده است. برای نمایش اطلاعاتی در مورد استثناء هم از خاصیت Message استفاده می‌کنیم. همه کلاس‌های استثناء توضیحاتی در مورد خطاها می‌دهند. در درس‌های آینده در مورد خصوصیات استثناءها بیشتر توضیح می‌دهیم. اگر فکر می‌کنید که در بلوک try ممکن است با چندین خطا مواجه شوید می‌توانید از چندین بلوک catch استفاده نمایید. ولی به یاد داشته باشید که برای هر کدام از آن خطاها از کلاس استثناء مربوط به همان خطا استفاده کنید.

```
int result;
int x = 5;
int y;

try
{
    y = Int32.Parse(Console.ReadLine());
    result = x / y;
}
catch (DivideByZeroException error)
{
    Console.WriteLine(error.Message);
}
catch (FormatException error)
{
    Console.WriteLine(error.Message);
}
```

از آنجاییکه مقدار y به وسیله ورودی که از کاربر گرفته می‌شود، تعیین می‌شود، مقدار آن باید با توجه به مثال بالا غیر صفر باشد (عدد تقسیم بر صفر تعریف نشده است). اما یک مشکل وجود دارد. چون ممکن است که کاربر یک مقدار غیر عددی وارد کند (مثلاً یک حرف)، که در این صورت برنامه نمی‌تواند حرف را به عدد تبدیل کند و خطای نوع (FormatException) اتفاق می‌افتد. وقتی استثناء اتفاق افتاد، بلوک catch مربوط به این خطا اجرا می‌شود و محاسبه خارج قسمت تقسیم x بر y نادیده گرفته می‌شود. حال فرض کنید شما می‌خواهید تمام خطاهای احتمالی که ممکن است در داخل بلوک try اتفاق می‌افتند را فهمیده و اداره کنید این کار چگونه امکانپذیر است؟ به راحتی و با استفاده از کلاس عمومی Exception می‌توانید این کار را انجام داد. هر کلاس استثناء در دات‌نت از این کلاس ارث بری می‌کند، بنابراین شما می‌توانید هر نوع استثناءیی را در شئی از کلاس Exception ذخیره نمایید.

```
try
{
    //Put your codes to test here
}
catch (Exception error)
{
    Console.WriteLine(error.Message);
}
```

با استفاده از این روش دیگر لازم نیست نگران اتفاق خطاهای احتمالی باشید چون بلوک catch برای هرگونه خطایی که در داخل بلوک try تشخیص داده شود، پیغام مناسبی نشان می‌دهد. به این نکته توجه کنید که اگر بخواهید از کلاس پایه Exception همراه با سایر کلاس‌های استثناء دیگر که از آن مشتق می‌شوند در برنامه استفاده کنید، باید کلاس پایه Exception در آخرین بلوک catch قرار گیرد.

```
try
{
    //Put your codes to test here
}
catch (DivideByZeroException)
{
    Console.WriteLine("Division by zero is not allowed.");
}
catch (FormatException)
{
    Console.WriteLine("Error on converting the data to proper type.");
}
catch (Exception)
{
    Console.WriteLine("An error occured.");
}
```

اگر کلاس پایه Exception را در اولین بلوک catch قرار دهیم و خطایی در برنامه رخ دهد، چون تمام کلاس‌های استثناء از این کلاس مشتق می‌شوند، در نتیجه، اولین بلوک catch اجرا شده و سایر بلوک‌ها حتی با وجود اینکه خطای مورد نظر به آنها مربوط باشد، اجرا نمی‌شوند. شما می‌توانید از عمگر is نیز به صورت زیر استفاده نمایید.

```
try
{

}
catch(Exception error)
{
    if (error is DivideByZeroException)
    {
        Console.WriteLine("Cannot divide by zero!");
    }
    if (error is FormatException)
    {
        Console.WriteLine("Format cannot be accepted!");
    }
}
```

بلوک catch از کلاس Exception برای به دام انداختن همه استثناءهایی که به وسیله برنامه به وجود می‌آید استفاده می‌کند. در داخل بلوک catch می‌توانید با استفاده از یک دستور if و کلمه کلیدی is نوع استثناء به وجود آمده را بیابید.

استفاده از بلوک finally

گاهی اوقات می‌خواهید برخی کدها همیشه اجرا شوند، خواه استثناء رخ دهد، خواه رخ ندهد، در این صورت از بلوک finally استفاده می‌شود. قبلاً یاد گرفتیم که اگر در بلوک try یک استثناء رخ دهد همه کدهای موجود در این بلوک نادیده گرفته شده و برنامه به قسمت catch می‌رود. کدهای نادیده گرفته شده ممکن است در برنامه نقش حیاتی داشته باشند. هدف بلوک finally هم حفظ نقش این کدها به صورت غیر مستقیم است. کدهایی را که فکر می‌کنید کدهای پایه ای هستند و برای اجرای برنامه لازم هستند را در داخل بلوک finally قرار دهید. برنامه زیر نحوه استفاده از این بلوک را نشان می‌دهد:

```
using System;

public class Program
{
    public static void Main()
    {
        int result;
        int x = 5;
        int y = 0;

        try
        {
            result = x / y;
        }
        catch (DivideByZeroException error)
        {
            Console.WriteLine(error.Message);
        }
        finally
        {
            Console.WriteLine("finally blocked was reached.");
        }
    }
}
```

```
Attempted to divide by zero.
finally blocked was reached.
```

بلوک finally بعد از بلوک catch نوشته می‌شود. اگر از چندین بلوک catch در برنامه استفاده می‌کنید بلوک finally باید بعد از همه آنها قرار گیرد. می‌توان از بلوک try و finally در صورتی که بلوک catch نداشته باشیم به صورت زیر استفاده کرد.

```
try
{
    //some code
}
finally
{
    //some code
}
```

از این بلوک معمولاً برای بستن یک اتصال پایگاه داده یا بستن یک فایل استفاده می‌شود.

ایجاد استثناء

شما می‌توانید در هر جای برنامه یک خطای ساختگی ایجاد کنید. همچنین اگر پیغام پیش‌فرض استثناءها را دوست ندارید می‌توانید به دلخواه خودتان یک پیغام برای نمایش ایجاد کنید. به مثال زیر توجه کنید:

```
1  using System;
2
3  class Program
4  {
5      public static void Main()
6      {
7          int firstNumber, secondNumber, result;
8
9          Console.Write("Enter the first number: ");
10         firstNumber = Int32.Parse(Console.ReadLine());
11
12         Console.Write("Enter the second number: ");
13         secondNumber = Int32.Parse(Console.ReadLine());
14
15         try
16         {
17             if (secondNumber == 0)
18             {
19                 throw new DivideByZeroException();
20             }
21             else
22             {
23                 result = firstNumber / secondNumber;
24             }
25         }
26         catch (DivideByZeroException error)
27         {
28             Console.WriteLine(error.Message);
29         }
30     }
31 }
```

```
Enter the first number: 10
Enter the second number: 0
Attempted to divide by zero.
```

در خط ۱۹ و درست قبل از یک نمونه ایجاد شده از کلاس exception، از کلمه کلیدی throw استفاده کرده‌ایم. می‌توان مستقیماً یک نمونه از کلاس exception ایجاد و یک خطا را به دام انداخت. به مثال زیر توجه کنید:

```
DivideByZeroException error = new DivideByZeroException();

throw error;
```

همچنین می‌توان یک پیغام خطای سفارشی را به وسیله یکی دیگر از سربارگذاری‌های کلاس Exception که یک رشته را دریافت و آن را به عنوان پیغام خطا نمایش می‌دهد، نمایش داد.

```
throw new DivideByZeroException("Cannot divide by zero!");
```

در این حالت پیغام خطای پیش‌فرض تغییر کرده و در خاصیت Message ذخیره می‌شود. ایجاد استثناء بیشتر در مواقعی به کار می‌رود که یک کد در حالت عادی خطا ندارد ولی شما می‌خواهید در هر صورت به عنوان یک خطا در نظر گرفته شود.

تعریف یک استثناء توسط کاربر

در سی‌شارپ می‌توان یک استثناء سفارشی ایجاد کرد. استثناء سفارشی، استثنایی است که توسط کاربر تعریف می‌شود و باید از کلاس پایه Exception ارث بری کند. برای این کار یک کلاس جداگانه که از کلاس پایه Exception ارث می‌برد، ایجاد می‌کنیم. یک برنامه کنسول ایجاد کنید و نام آن را UserDefinedExceptions بگذارید. بعد از ایجاد پروژه بر روی دکمه Add New Item (مسیر Project > Add New Item در نوار ابزار (toolbar) کلیک کنید و از پنجره باز شده گزینه Class را انتخاب کنید. نام کلاس را NegativeNumberException بگذارید.

```
using System;

namespace UserDefinedExceptions
{
    class NegativeNumberException : Exception
    {
        public NegativeNumberException()
            : base("The operation will result to a negative number.")
        {
        }

        public NegativeNumberException(string message)
            : base(message)
        {
        }

        public NegativeNumberException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}
```

در خط ۵ مشاهده می‌کنید کلاس ایجاد شده توسط ما از کلاس Exception ارث بری کرده است. به عنوان یک قرارداد باید به آخر نام کلاس‌های استثنایی که توسط کاربر تعریف می‌شوند کلمه Exception اضافه شده و ۳ سازنده برای آنها تعریف شود. اولین سازنده بدون پارامتر می‌باشد. دومین سازنده یک آرگومان از نوع رشته برای نمایش پیغام خطا قبول می‌کند. سومین سازنده که دو آرگومان قبول می‌کند، یکی پیغام خطا را نمایش داده و یکی بخش inner است که از آن برای نشان دادن علت وقوع استثناء استفاده می‌شود. حال می‌خواهیم یک کلاس استثناء خیلی سفارشی ایجاد کنیم. در فایل Program.cs کد زیر را وارد کنید:

```
using System;

namespace NegativeNumberException
{
    class Program
    {
        public static void Main()
        {
            int firstNumber, secondNumber, difference;

            Console.Write("Enter the first number: ");
            firstNumber = Int32.Parse(Console.ReadLine());

            Console.Write("Enter the second number: ");
            secondNumber = Int32.Parse(Console.ReadLine());

            difference = firstNumber - secondNumber;

```

```
19              try
20              {
21                  if (difference < 0)
22                  {
23                      throw new NegativeNumberException();
24                  }
25              }
26              catch (NegativeNumberException error)
27              {
28                  Console.WriteLine(error.Message);
29              }
30          }
31      }
32  }
```

```
Ente the first number: 10Enter the second number: 11
The operation will result to a negative number.
```

از آنجاییکه تولید یک عدد منفی در هیچ برنامه‌ای یک استثناء محسوب نمی‌شود، ما به صورت دستی و برای خودمان یک استثناء ایجاد کرده‌ایم. ابتدا از کاربر می‌خواهیم که دو مقدار را وارد کند (خطوط ۱۵-۱۱). سپس تفاوت دو عدد را محاسبه می‌کنیم (خط ۱۷). در داخل بلوک try تست می‌کنیم که آیا حاصل تفریق دو عدد، یک عدد منفی است (خط ۲۱). اگر یک عدد منفی بود سپس یک نمونه از کلاس NegativeNumberException ایجاد می‌کنیم (خط ۲۳). بعد از ایجاد نمونه به وسیله بلوک catch و برای نشان داده پیغام خطا آن را اداره می‌کنیم (خطوط ۲۹-۲۶).

خواص Exception

کلاس پایه System.Exception کلاسی است که سایر کلاس‌های استثناء از آن ارث بری می‌کنند. بنابراین خواص این کلاس در دسترس سایر کلاس‌های استثناء می‌باشد. در جدول زیر برخی از خواص برجسته کلاس Exception که در همه کلاس‌های استثناء وجود دارند، آمده است:

خواص	توضیحات
InnerException	استثنایی که موجب تولید مشکل شده است
Message	متنی که استثناء را شرح می‌دهد
StackTrace	برای تشخیص خطایی که باعث ایجاد مشکل شده است

خاصیت Message

این خاصیت به شما اجازه می‌دهد که در مورد استثناء به وجود آمده توضیحاتی ارائه دهید. به عنوان مثال به کد زیر توجه کنید:

```
int x = 1;
int y = 0;
int z;

try
{
    z = x / y;
}
```

```
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

```
Attempted to divide by zero.
```

در مثال بالا عمداً یک عدد را بر صفر تقسیم می‌کنیم تا بلوک catch اجرا شود. به این نکته توجه کنید که چون یک نمونه از کلاس Exception ایجاد کرده‌ایم، بنابراین می‌توانیم از خاصیت Message استفاده کنیم. سپس محتویات خاصیت Message را به وسیله متد Console.WriteLine() چاپ می‌کنیم. حال اگر برنامه را اجرا کنیم پیغام خطا نمایش داده می‌شود. همه کلاس‌های استثناء از قبل تعریف شده در دات‌نت پیغام خطاهای مرتبط با خودشان را در خاصیت property دارند. اگر یک استثناء ایجاد کنید می‌توانید از سربارگذاری سازنده کلاس Exception که یک رشته که همان پیغام خطاست را دریافت می‌کند، استفاده کنید:

```
try
{
    if (y == 0)
    {
        throw new DivideByZeroException("You cannot divide by zero. Sorry my friend.");
    }
    else
    {
        z = x / y;
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

```
You cannot divide by zero. Sorry my friend.
```

خاصیت InnerException

از این خاصیت برای تشخیص خطایی که باعث ایجاد مشکل شده است استفاده می‌شود. به عنوان مثال فرض کنید که یک برنامه نویس می‌خواهد تشخیص دهد که آیا شماره حسابی که توسط کاربر وارد شده است صحیح می‌باشد یا نه؟ اولین کاری که باید انجام شود تبدیل نوع رشته به عدد صحیح است. اگر عدد وارد شده توسط کاربر که از نوع رشته است را نتوانیم به درستی به نوع عدد تبدیل کنیم خطای FormatException رخ می‌دهد. در قسمت chatch برای FormatException، می‌توان مانور بیشتری داد و استثناء رو مدیریت کرد و پیغام مناسب را به کاربر نشان داد. به کد زیر توجه کنید:

```
1   using System;
2
3   namespace InnerExceptionDemo
4   {
5       class Program
6       {
7           static int ProcessAccountNumber(string accountNumber)
8           {
9               try
10              {
11                  return Convert.ToInt32(accountNumber);
12              }
13              catch (FormatException formatException)
```

```
            {
                throw new Exception("Invalid Account Number.", formatException);
            }
        }
        static void Main(string[] args)
        {
            Console.Write("Enter an account number: ");
            string input = Console.ReadLine();

            try
            {
                int accountNumber = ProcessAccountNumber(input);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Current exception's message: {0}", ex.Message);
                Console.WriteLine("Inner exception's message: {0}", ex.InnerException.Message);
            }
        }
    }
}
```

```
Enter an account number: abcde
Current exception's message: Invalid Account Number.
Inner exception's message: Input string was not in a correct format.
```

در خطوط ۱۷-۷ یک متد به نام ProcessAccountNumber() تعریف کرده‌ایم. این متد یک آرگومان از نوع رشته قبول می‌کند که همان رشته‌ای است که توسط کاربر وارد می‌شود. همانطور که در خط ۲۵ مشاهده می‌کنید متد ProcessAccountNumber() فراخوانی شده و رشته وارد شده توسط کاربر به عنوان آرگومان به آن ارسال می‌شود. این خط کد (خط ۲۵) در داخل یک بلوک try قرار داده می‌شود چون ممکن است باعث ایجاد خطا شود. بعد از اینکه متد فراخوانی شد برنامه به محل تعریف متد منتقل می‌شود (خط ۷). در داخل متد دستور try…catch دیگری وجود دارد. در داخل بلوک try سعی می‌کنیم که رشته را به عدد تبدیل کرده و مقدار آن را برگشت دهیم. اگر رشته نتواند به عدد تبدیل شود خطای FormatException رخ داده و در نتیجه بلوک catch اجرا می‌شود.

در داخل بلوک catch یک نمونه از کلاس Exception ایجاد کرده و پیغام دلخواه به عنوان اولین پارامتر ورودی و FormatException را به عنوان پارامتر دوم که نشان دهنده یک InnerException است، برای آن مشخص می‌کنیم. بعد از اجرای این قسمت بلاک catch تابع اصلی یعنی خط ۲۷ اجرا خواهد شد. در این بلاک اول Exception جدید و خاصیت Message آن نمایش داده می‌شود و بعد از آن خاصیت Message استثناءیی که باعث اجرای این بلاک شد است، به عنوان InnerException نمایش داده می‌شود.

خاصیت StackTrace

خاصیت StackTrace در کلاس System.Exception به شما اجازه می‌دهد تا بفهمید در کدام متد از متدهای پشته خطا رخ داده است. این خاصیت برای زمانی است که شما Method1 را داخل Method2 صدا زده‌اید و Method1 دارای خطا است. با استفاده از این خاصیت می‌توانید بفمید که در Method1 و به دنبال آن در Method2 هم خطا رخ داده است. به کد زیر دقت کنید:

```
using System;

namespace StackTraceDemo
{
```

```
    class Program
    {
        static void Method1()
        {
            Method2();
        }

        static void Method2()
        {
            Method3();
        }

        static void Method3()
         {
               throw new Exception("Exception at Method3()");
          }

        static void Main(string[] args)
        {
            try
            {
                Method1();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.StackTrace);
                Console.ReadKey();
            }
        }
    }
}
```

```
at StackTraceDemo.Program.Method3() in C:\StackTraceDemo\Program.cs:line 19
at StackTraceDemo.Program.Method2() in C:\StackTraceDemo\Program.cs:line 14
at StackTraceDemo.Program.Method1() in C:\StackTraceDemo\Program.cs:line 9
at StackTraceDemo.Program.Main(String[] args) in C:\StackTraceDemo\Program.cs:line 26
```

هر خط، فایل و متد و خطِ دارای خطا را نشان می‌دهد. Method3 به عنوان منبع خطا در اولین خط نشان داده شده است.

اشکال زدایی توسط ویژوال استودیو

اشکال زدایی، فرایندی است که طی آن خطاها و باگ‌های برنامه شما تشخیص داده می‌شود. اگر خطاهای دستوری برنامه‌تان را با استفاده از پنجره List Error برطرف کرده‌اید، مرحله بعد پیدا کردن خطاهای منطقی و استثناءها در هنگام اجرای برنامه است. ویژوال استودیو و ویژوال سی‌شارپ دارای ابزارهای کارآمدی برای اشکال زدایی هستند. با استفاده از این ابزارها می‌توانید برنامه را در نقاط خاصی (دلخواه) متوقف کرده و مقادیر هر یک از متغیرهایی را که توسط برنامه مورد استفاده قرار می‌گیرند را مشاهده نمایید. این کار را می‌توان به صورت گام به گام و خط به خط انجام داد. می‌توانید استثناء را چک کرده و از طریق راهنمای ویژوال استودیو که در درس‌های آینده توضیح می‌دهیم، مشکلات مربوط را برطرف کرد.

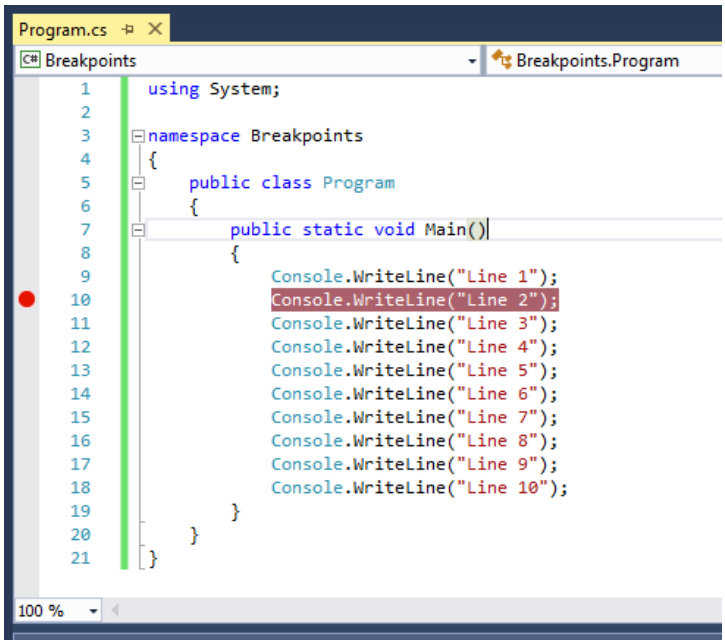
نقطه انفصال (Breakpoints)

Breakpoints به شما اجازه می‌دهد که برنامه‌تان را در نقاط خاصی متوقف کنید. برای این کار Breakpoints را در مکانی از کدتان که می‌خواهید اطلاعاتی در مورد آن کسب کنید، قرار می‌دهید. Breakpoints را فقط می‌توان در خطوط قابل اجرای کد قرار داد. به وسیله این ابزار (Breakpoints) می‌توان برنامه را موقتاً متوقف کرده و سپس با استفاده از دیگر ابزارهای ویژوال استودیو آن را خطایابی کرد. به وسیله برنامه زیر نحوه کار با Breakpoints را به شما آموزش می‌دهیم. یک برنامه کنسول ایجاد کرده و نام آن را Breakpoints بگذارید. کدهای زیر را در برنامه وارد کنید:

```
using System;

namespace Breakpoints
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Line 1");
            Console.WriteLine("Line 2");
            Console.WriteLine("Line 3");
            Console.WriteLine("Line 4");
            Console.WriteLine("Line 5");
            Console.WriteLine("Line 6");
            Console.WriteLine("Line 7");
            Console.WriteLine("Line 8");
            Console.WriteLine("Line 9");
            Console.WriteLine("Line 10");
        }
    }
}
```

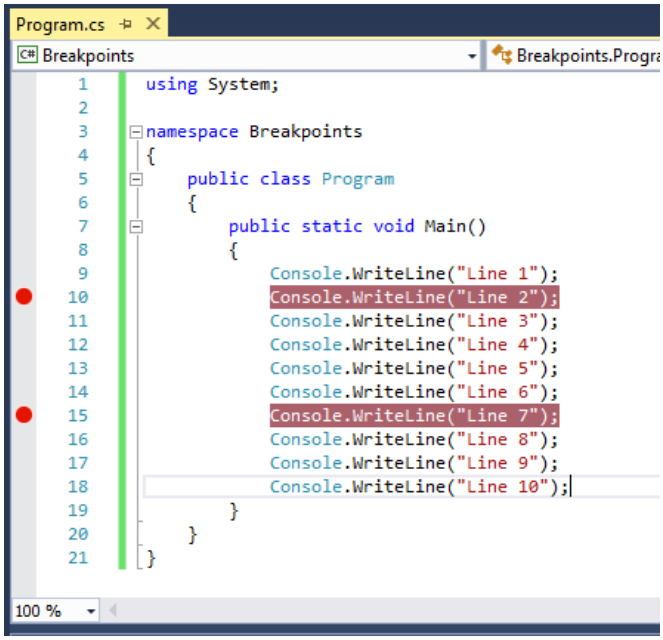
اضافه کردن Breakpoints

برای اضافه کردن Breakpoints، یک خط کد اجرایی را پیدا کرده و بر روی حاشیه خاکستری رنگ سمت چپ کد (شکل زیر) کلیک کنید. به عنوان مثال اجازه دهید یک Breakpoints به دومین دستور WriteLine() اضافه کنیم.

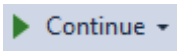


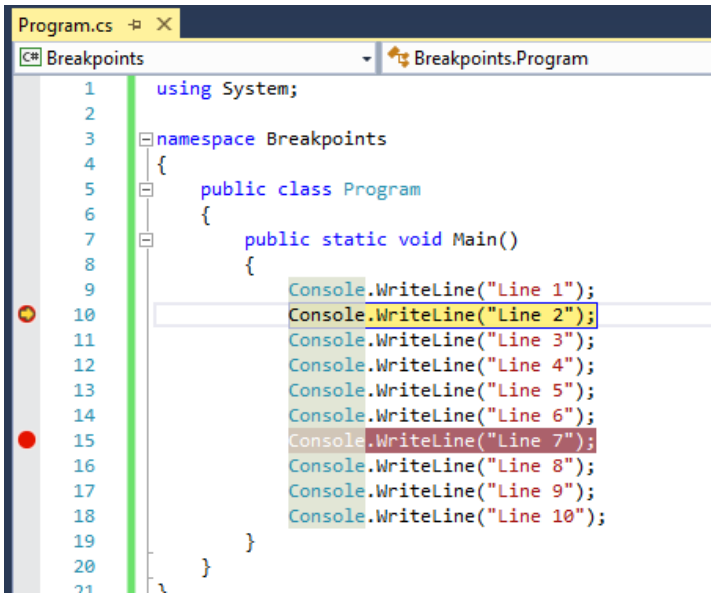

```
using System;

namespace Breakpoints
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Line 1");
            Console.WriteLine("Line 2");
            Console.WriteLine("Line 3");
            Console.WriteLine("Line 4");
            Console.WriteLine("Line 5");
            Console.WriteLine("Line 6");
            Console.WriteLine("Line 7");
            Console.WriteLine("Line 8");
            Console.WriteLine("Line 9");
            Console.WriteLine("Line 10");
        }
    }
}
```

همچنین می‌توان بر روی کد مورد نظر کلیک راست کرده و از مسیر Breakpoints > Insert Breakpoint یک Breakpoints به آن اضافه کنید. دستور یا خط کدی که دارای Breakpoint است با رنگ قرمز نشان داده می‌شود. برای پاک کردن Breakpoint کافی است بر روی کدی که دارای Breakpoint است کلیک راست کرده و از مسیر Breakpoint گزینه Delete Breakpoint را انتخاب کنید. همچنین می‌توان بر روی کد مورد نظر دکمه F9 را برای فعال و غیر فعال کردن Breakpoint فشار داد. می‌توانید تعداد زیادی Breakpoint در قسمت‌های مختلف کدتان قرار دهید. حال به هفتمین دستور WriteLine() هم یک Breakpoint اضافه کنید.

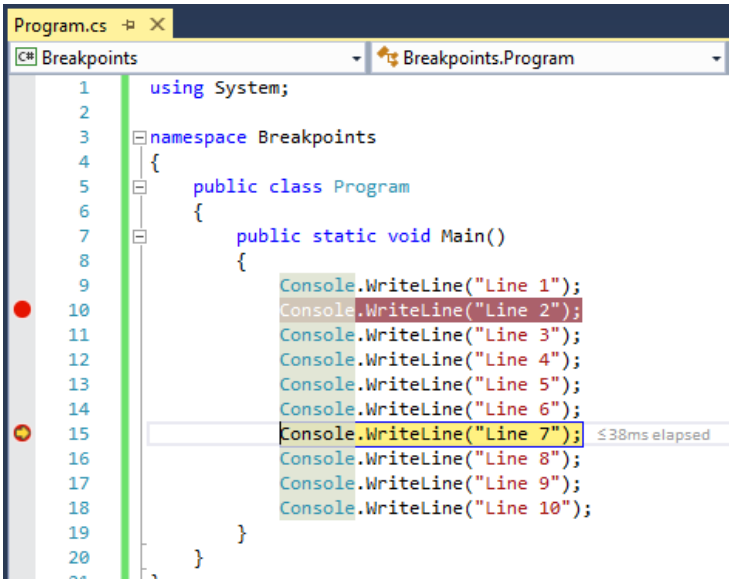



```
using System;

namespace Breakpoints
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Line 1");
            Console.WriteLine("Line 2");
            Console.WriteLine("Line 3");
            Console.WriteLine("Line 4");
            Console.WriteLine("Line 5");
            Console.WriteLine("Line 6");
            Console.WriteLine("Line 7");
            Console.WriteLine("Line 8");
            Console.WriteLine("Line 9");
            Console.WriteLine("Line 10");
        }
    }
}
```

برای مشاهده عملکرد Breakpoint برنامه را در حالت Debug به وسیله دکمه F5 اجرا کنید و یا بر روی شکل  کلیک کنید. به این نکته توجه کنید که می‌توان حالت Debug را با استفاده از دکمه آبی مربع شکلی که بعد از دکمه Debug (فلش سبزرنگ) که در Toolbar قرار دارد متوقف کرد. وقتی که برنامه اجرا می‌شود بعد از رسیدن به اولین Breakpoint متوقف می‌شود.



در این حالت کدهای قبل از دومین دستور اجرا می‌شوند. اگر به پنجره محیط کنسول نگاه کنید، متوجه می‌شوید که اولین دستور WriteLine() اجرا شده است. فلش زرد رنگ نشان می‌دهد که کد زرد رنگ متناظر با آن توسط برنامه اجرا شده است. برای ادامه Breakpoint و تست کد بعدی دوباره فلش سبز رنگ واقع در نوار Toolbar را فشار دهید. همچنین می‌توان از دکمه F5 استفاده کرد یا از مسیر Debug گزینهContinue را انتخاب کنید.



ادامه به Breakpoint باعث اجرای همه کدهای بین دو Breakpoint مبدأ و Breakpoint مقصد می‌شود. اگر به محیط کنسول نگاه کنید مشاهده می‌کنید که همه خط‌های قبل از هر Breakpoint اجرا می‌شوند. فلش زرد رنگ هم با هر بار فشار دادن دکمه F5 به BreakPoint

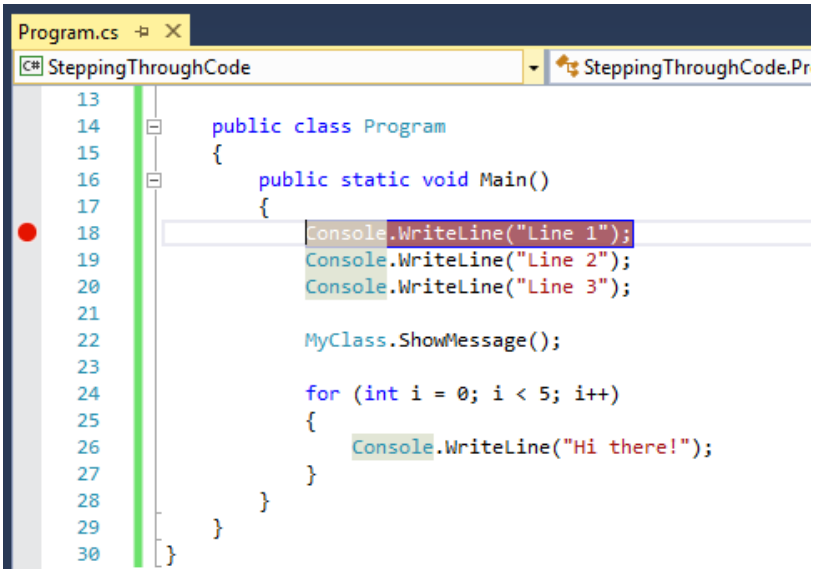
های بعدی می‌رود. بعد از اتمام BreakPoint ها برنامه نیز خود به خود بسته می‌شود. برای غیر فعال کردن موقتی Breakpoint ها هم می‌توانید با کلیک راست بر روی هر دستور از مسیر Breakpoints > Disable Breakpoint این کار را انجام دهید. Breakpoint غیر فعال با یک دایره توخالی نمایش داده می‌شود. برای فعال کردن مجدد آن بر روی دستور کلیک راست کرده و از منوی باز شده گزینه Breakpoint و سپس Enable Breakpoint را انتخاب می‌کنید. همچنین می‌توان با قرار دادن نشانگر ماوس بر روی دستور دارای Breakpoint آن را فعال یا غیر فعال کرد.

قدم زدن در میان کدها

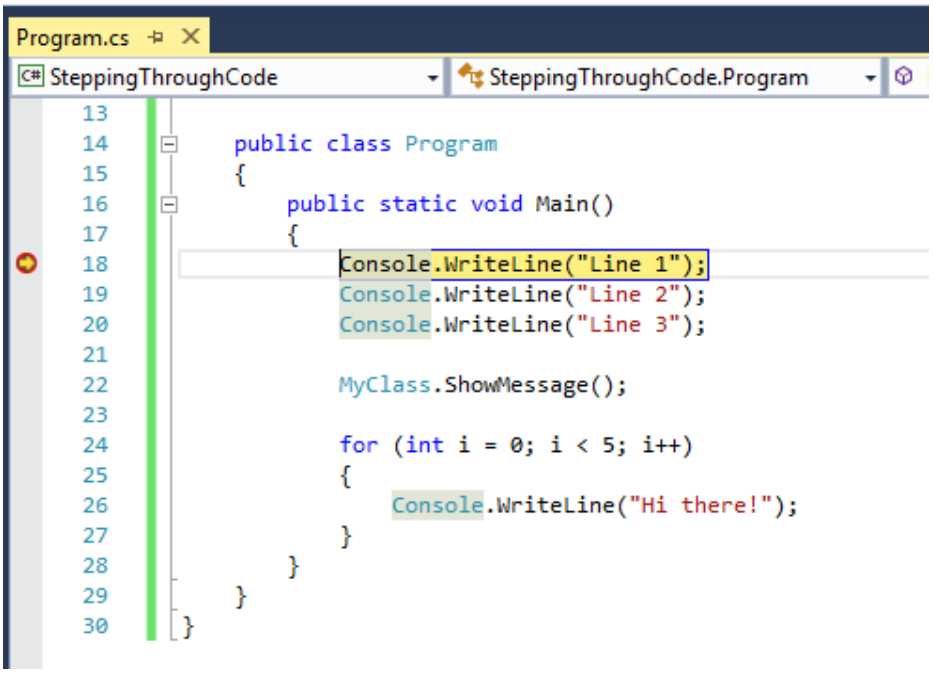
با استفاده از BreakPoint می‌توان خط به خط کدها را مورد بررسی قرار داد و از تأثیر هر خط در برنامه مطلع شد. یک برنامه کنسول جدید ایجاد کنید و کدهای زیر را در آن وارد نمایید:

```
using System;

namespace SteppingThroughCode
{
    public class MyClass
    {
        public static void ShowMessage()
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("Have a nice day!");
        }
    }

    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Line 1");
            Console.WriteLine("Line 2");
            Console.WriteLine("Line 3");

            MyClass.ShowMessage();

            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Hi there!");
            }
        }
    }
}
```

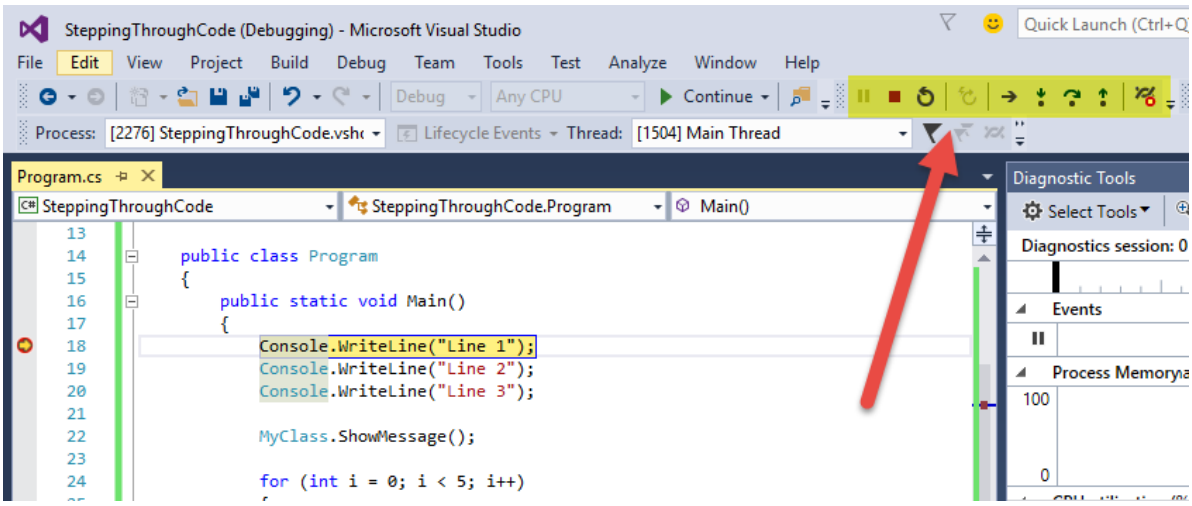
یک Breakpoint به اولین دستور WriteLine()در متد Main() (خط ۱۸) اضافه کنید.



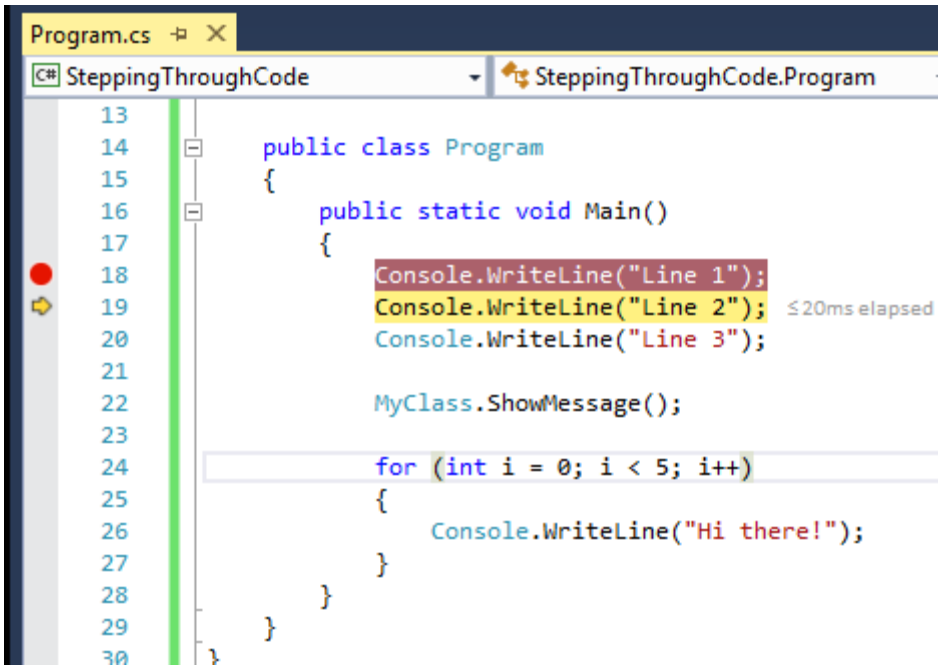
می‌توانیم به شما نشان دهیم که چگونه می‌توانید در میان بخشهای مختلف کدتان با استفاده از گزینه‌های Step Over، Step Into و Step Out قدم بزنید. برنامه‌تان در حالت Debug به وسیله دکمه F5 اجرا کنید. برنامه در اولین Breakpoint متوقف می‌شود. اکنون می‌توانیم چک کردن کدهایمان را از این Breakpoint شروع کنیم.



دستوراتی که به وسیله آنها می‌توان تک تک بخشهای کدتان را تست کنید در toolbar قرار دارند که به وسیله شکل زیر به شما نمایش داده شده‌اند:



کلیک بر روی آیکون  Step Over باعث اجرای خط جاری برنامه شده و سپس فلش زرد رنگ را به خط بعد می‌برد. حال اگر به محیط کنسول نگاهی بیندازید مشاهده می‌کنید که اولین دستور WriteLine() اجرا شده است.



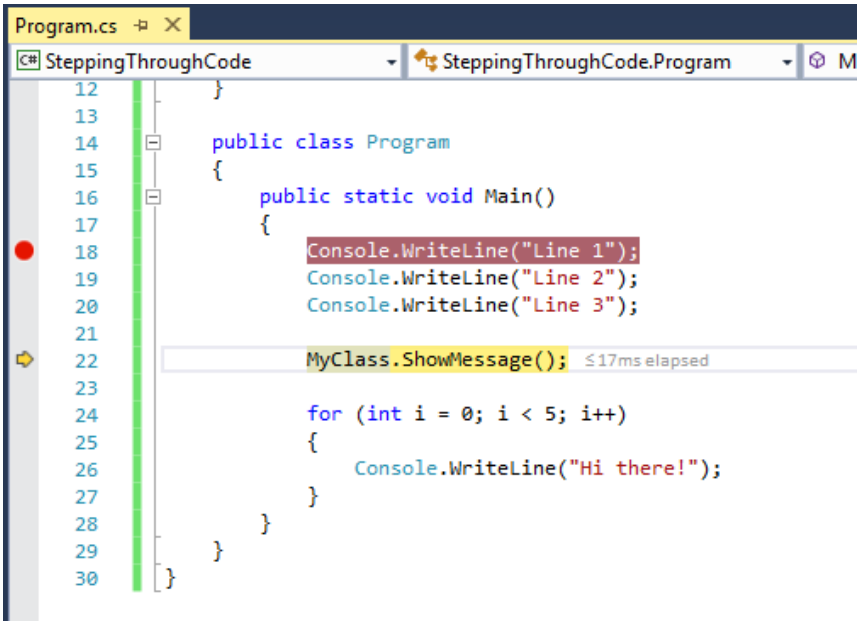
تکرار این کار (کلیک بر روی آیکون Step Over) باعث اجرای تمام کدها می‌شود. کلیک بر روی آیکون Step Over را تا جایی ادامه دهید که به خط MyClass.ShowMessage() برسید.

```
Program.cs
SteppingThroughCode                SteppingThroughCode.Program
13
14      public class Program
15      {
16          public static void Main()
17          {
18              Console.WriteLine("Line 1");
19              Console.WriteLine("Line 2");
20              Console.WriteLine("Line 3");
21
22              MyClass.ShowMessage();  ≤1ms elapsed
23
24              for (int i = 0; i < 5; i++)
25              {
26                  Console.WriteLine("Hi there!");
27              }
28          }
29      }
30  }
```

حال از دستور Step Into استفاده می‌کنیم  . این دستور وارد متد انتخاب شده می‌شود (شما را به محل تعریف متد می‌برد) و جزئیاتی در مورد آن در اختیار شما قرار می‌دهد و شما را قادر می‌سازد که خط‌های داخل متد را یک به یک اجرا کنید:

```
Program.cs
SteppingThroughCode                SteppingThroughCode.MyClass
1   using System;
2
3   namespace SteppingThroughCode
4   {
5       public class MyClass
6       {
7           public static void ShowMessage()
8           {  ≤1ms elapsed
9               Console.WriteLine("Hello World!");
10              Console.WriteLine("Have a nice day!");
11          }
12      }
13
14      public class Program
15      {
16          public static void Main()
17          {
18              Console.WriteLine("Line 1");
19              Console.WriteLine("Line 2");
```

با استفاده از دستور Step Over به خط‌های بعدی متد بروید و برای خروج از متد هم می‌توان از دستور Step Out استفاده کرد  . استفاده از دستور Step Out باعث اجرای بخشهای باقیمانده متد شده و به محل فراخوانی متد می‌رود (شکل زیر):



با کلیک دوباره بر روی دکمه Step Over وارد حلقه for می‌شوید. با هر بار زدن این دکمه هم شرط تست می‌شود هم حلقه اجرا می‌شود و هم شمارنده افزایش می‌یابد. اگر شرط درست باشد شما وارد حلقه و اگر نادرست باشد وارد دستور بعد می‌شوید. وارد شدن به کدها بدون دانستن اینکه چگونه آنها را تست کنید، بی فایده است. در درس بعد به شما نحوه به دست آوردن مقادیر متغیرها و اعضای یک شیء را آموزش می‌دهیم.

به دست آوردن مقادیر متغیرها

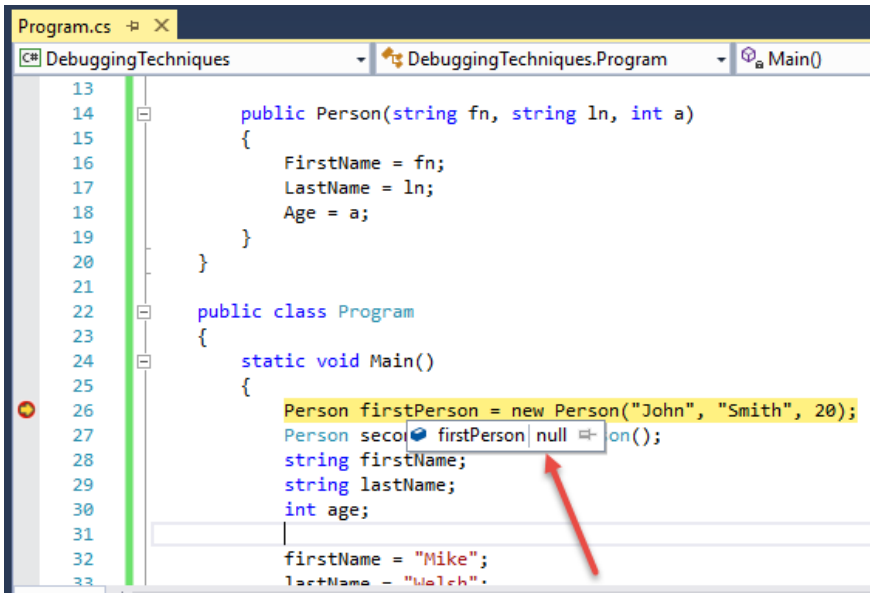
در حالت Debug می‌توان به قسمت کدنویسی رفته و مقدار و حالات اشیاء و متغیرها را مورد بررسی قرار داد. همانطور که قبلاً ذکر شد، می‌توان از Breakpoint برای جلوگیری موقت از اجرای برنامه و امتحان کردن کدهای آن استفاده کرد. ویژوال استودیو و ویژوال سی‌شارپ دارای ابزارهایی هستند که به شما نحوه اجرای برنامه‌ها را نشان می‌دهند. یک برنامه کنسول ایجاد کنید و نام آن را DebuggingTechniques بگذارید. کد زیر را به برنامه اضافه کنید:

```
using System;

namespace DebuggingTechniques
{

    public class Person
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }

        public Person()
            : this("", "", 0)
        {

        }

        public Person(string fn, string ln, int a)
        {
            FirstName = fn;
            LastName = ln;
            Age = a;
```

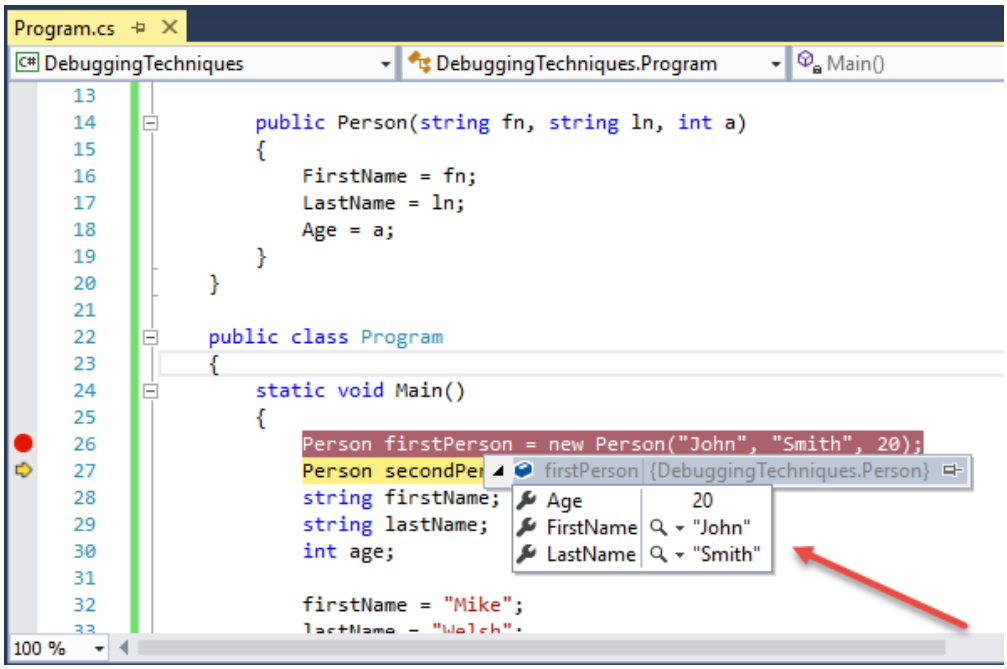
```
        }
    }

    public class Program
    {
        static void Main()
        {
            Person firstPerson = new Person("John", "Smith", 20);
            Person secondPerson = new Person();
            string firstName;
            string lastName;
            int age;

            firstName = "Mike";
            lastName = "Welsh";
            age = 30;

            secondPerson.FirstName = firstName;
            secondPerson.LastName = lastName;
            secondPerson.Age = age;

            Console.WriteLine("First Person Details: \n");
            Console.WriteLine("First Name: {0}", firstPerson.FirstName);
            Console.WriteLine("Last Name:  {0}", firstPerson.LastName);
            Console.WriteLine("Age:       {0}", firstPerson.Age);

            Console.WriteLine("\nSecond Person Details: \n");
            Console.WriteLine("First Name: {0}", secondPerson.FirstName);
            Console.WriteLine("Last Name:  {0}", secondPerson.LastName);
            Console.WriteLine("Age:        {0}", secondPerson.Age);
        }
    }
}
```

```
First Person Details:

First Name: John
Last Name:  Smith
Age:       20

Second Person Details:

First Name: Mike
Last Name:  Welsh
Age:        30
```

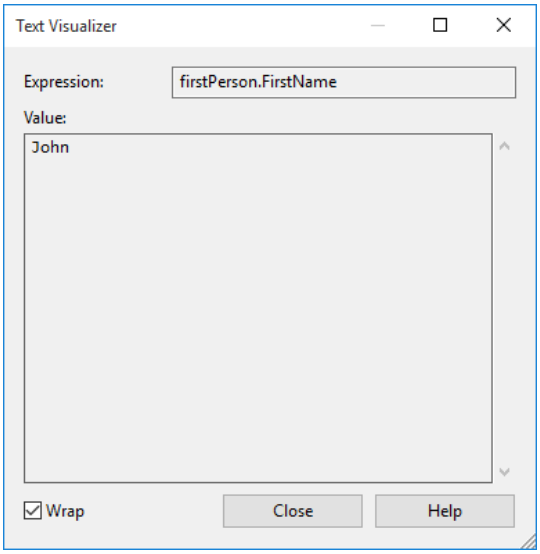
یک Break point به اولین خط کد در متد Main() اضافه کنید. حال به شما نشان می‌دهیم که چطور مقادیر موجود در متغیرها و اشیاء را مشاهده کنید .برنامه را در حالت Debug اجرا کنید. برنامه در اولین Breakpoint متوقف می‌شود نشانگر ماوس را بر روی هر شیء یا متغیر که ببرید یک پنجره کوچک ظاهر می‌شود که شامل نام و مقدار موجود در آن شیء یا متغیر می‌باشد.



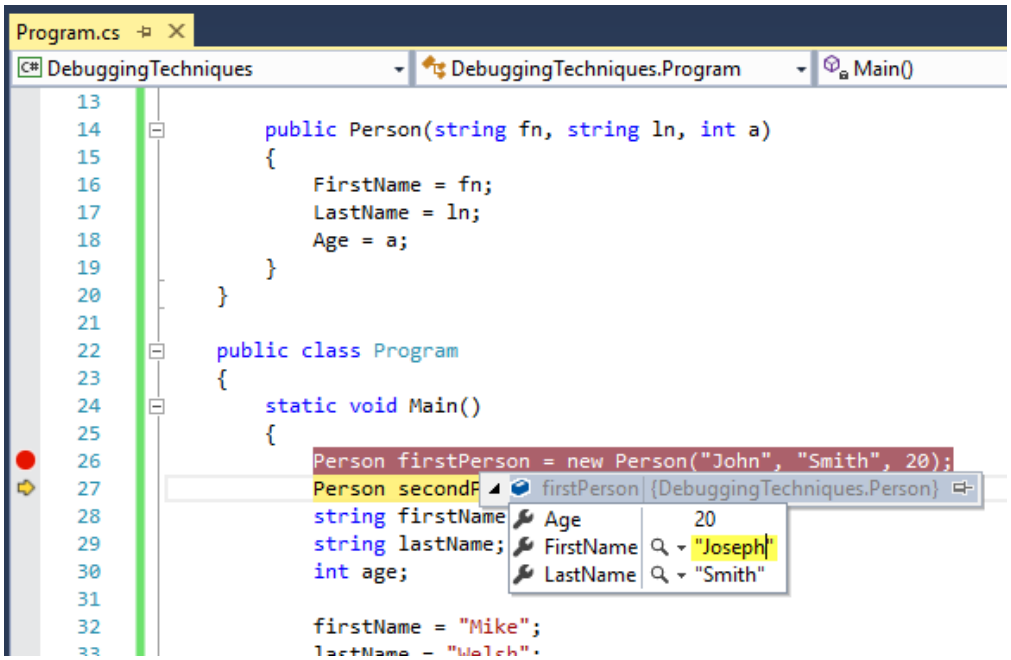
مقدار جاری شی firstPerson تهی است، چون هنوز هیچ مقداری به آن اختصاص داده نشده است. خط زرد رنگ هنوز اجرا نشده است. در پنجره کوچکی که با فلش نشان داده شده است، یک آیکون سنجاق مانند مشاهده می‌کنید. با کلیک بر روی این آیکون پنجره کوچک دیگر به صورت خودکار بسته نمی‌شود. برای اینکه پنجره کوچک دوباره به صورت عادی برگردد و به صورت خودکار مخفی شود، کافیست بار دیگر بر روی آیکون مذکور کلیک کنید. اگر ماوس را برای لحظه‌ای بر روی یک نوع یا یک کلاس قرار دهید (متوقف کنید)، جزییاتی در مورد آنها از قبیل سلسله مراتب وراثت را مشاهده می‌کنید.



بر روی آیکون Step Over کلیک کرده تا کد جاری اجرا شود و فلش زرد رنگ به خط بعد برود. از آنجاییکه خط کد قبل اجرا شده است، firstPerson به وسیله سازنده‌اش مقداردهی اولیه می‌شود. اگر ماوس را بر روی شیء firstPerson برای لحظه‌ای قرار دهید، یک پنجره ظاهر می‌شود (پنجره DataTip) که شامل مقادیر و نام کامل کلاس آن است. همچنین یک آیکون + در سمت چپ آن مشاهده خواهید کرد که با کلیک بر روی آن همه خواص و فیلدهای شیء جاری و مقادیر مربوط به آنها نمایش داده خواهد شد.



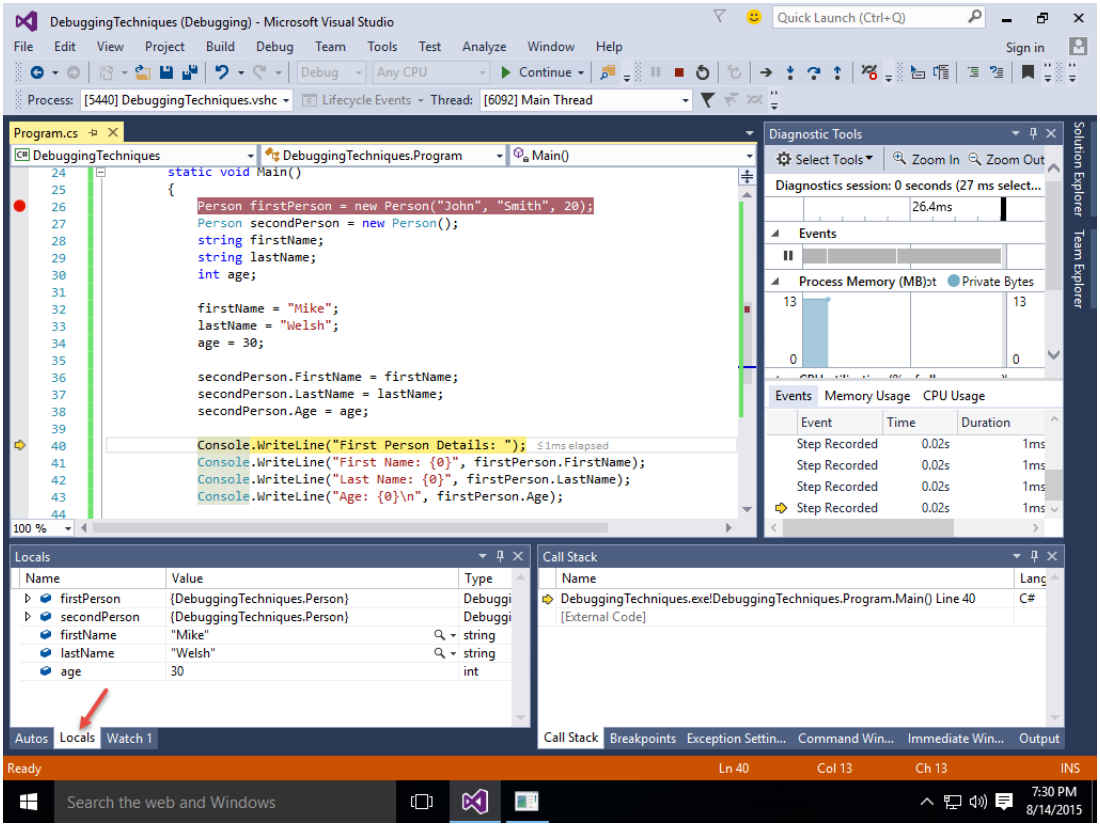
همچنین آیکون یک ذره بین را در کنار برخی از اعضاء مشاهده می‌کنید که با کلیک بر روی آن تمام محتویات یک متغیر حتی اگر بزرگ باشد به شما نشان داده خواهد شد (شکل زیر):



با کلیک بر روی مقادیر در پنجره DataTip می‌توان آنها را تغییر داد. به عنوان مثال بر روی مقدار فیلد FirstName کلیک کنید و آن را از John به Joseph تغییر داده و سپس کلید Enter را فشار دهید.

پنجره Locals

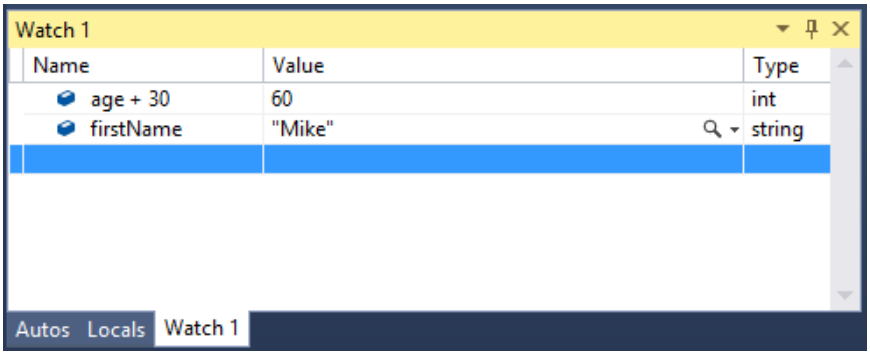
پنجره Locals، مکانی است که در آن می‌توانید همه مقادیر مربوط به متغیرها و اشیاء را که به وسیله برنامه مورد استفاده قرار می‌گیرند، مشاهده کنید. این پنجره به صورت خودکار وقتی که برنامه را در حالت Debug اجرا می‌کنید، نمایش داده می‌شود. مکان این پنجره همانطور که در شکل زیر نمایش داده شده است در قسمت پایین سمت چپ محیط ویژوال استودیو قرار دارد. تا وقتی که به اولین دستور WriteLine() می‌رسید، بر روی آیکون Step Over کلیک کنید. حال به پنجره Locals که با فلش نشان داده شده است، نگاه کنید:



لیستی از همه متغیرهایی که مورد استفاده قرار گرفته‌اند را، مشاهده می‌کنید. این پنجره شامل سه ستون به نام‌های Name، Value و Type می‌باشد. برای انواع پیچیده مانند کلاس‌ها یک آیکون + در سمت چپ نام آنها قرار دارد که با کلیک بر روی آن اعضای کلاس و مقادیر آنها نشان داده می‌شود. می‌توان با دو بار کلیک کردن بر روی مقادیر متغیرها در پنجره Locals آنها را تغییر داد.

پنجره Watch

پنجره Watch اجازه تایپ نام متغیرها و مشاهده مقادیر آنها را می‌دهد. این پنجره حتی به شما اجازه می‌دهد، دو متغیر مقداری را به وسیله عملگرهای ریاضی با هم جمع یا ...کنید. این پنجره همانطور که در شکل مشاهده می‌کنید در کنار پنجره Locals قرار دارد.



می‌توانید نام متغیری که مد نظرتان است را در ستون Name تایپ کرده و سپس کلید Enter را فشار دهید. با این کار مقدار متغیر در ستون Value نمایش داده خواهد شد. همچنین می‌توانید به عنوان مثال در ستون Name یک مقدار ثابت را با مقدار یک متغیر که در برنامه وجود دارد جمع کرده و نتیجه را مشاهده کنید.

مجموعه‌ها (Collections)

قبلاً یاد گرفتیم که آرایه‌ها به ما اجازه ذخیره چندین مقدار از یک نوع را می‌دهند. آرایه‌ها از کلاس مجرد System.Array ارث بری می‌کنند، که این کلاس دارای خواص و متدهایی برای کار با داده‌های ساده‌ای مانند طول آرایه می‌باشد. آرایه‌های ساده در سی‌شارپ دارای طول ثابتی هستند که یک بار تعریف و مقدار دهی می‌شوند و شما نمی‌توانید طول یک آرایه خاص را افزایش یا کاهش دهید. دات‌نت گزینه بهتری برای جایگزین کردن با آرایه‌ها پیشنهاد می‌دهد و بیشتر آنها، کلاس‌ها و رابط‌هایی هستند که در فضای نام System.Collections قرار دارند. به عنوان مثال کلاس ArrayList رفتاری شبیه به یک آرایه معمولی دارد با این تفاوت که به شما اجازه می‌دهد که طول آن را به صورت پویا تغییر داده یا یک عنصر را در طول اجرای برنامه به آن اضافه کرده و یا از آن حذف نمایید. دردرس بعد پی می‌برید که چگونه یک کلاس که شامل مجموعه‌ای از اشیاء است را به وسیله اجرا کردن و یا ارث بری از رابط‌ها و متدها ایجاد کنیم.

کلاس ArrayList

کلاس ArrayList به شما اجازه ذخیره مقادیر انواع داده‌ای مختلف، و توانایی حذف و اضافه عناصر آرایه در هر لحظه را می‌دهد. در مثال زیر به سادگی کاربرد کلاس ArrayList آمده است.

```
using System;
using System.Collections;

public class Program
{
    public static void Main()
    {
        ArrayList myArray = new ArrayList();

        myArray.Add("John");
        myArray.Add(5);
        myArray.Add(true);
        myArray.Add(3.65);
        myArray.Add('R');
```

```
        foreach (object element in myArray)
        {
            Console.WriteLine(element.ToString());
        }
    }
}
```

```
John
5
true
3.65
R
```

برای استفاده از این کلاس، ابتدا باید در قسمت فضاهای نامی، فضای نام System.Collections را وارد کنیم (خط ۲). همانطور که در مثال مشاهده می‌کنید یک نمونه از کلاس ArrayList ایجاد می‌کنیم. برای اضافه کردن یک عنصر به آرایه باید از متد ()Add استفاده کنیم. از آنجاییکه شیء ایجاد شده از کلاس ArrayList آرگومانی از نوع object قبول می‌کند بنابراین می‌توان مقادیری از هر نوع داده‌ای به آن ارسال کرد چون هر چیز در سی‌شارپ از object ارث بری می‌کند. حال برای نمایش توانایی این کلاس در نگهداری انواع داده‌ای مختلف پنج مقدار از پنج نوع مختلف داده را به آن اضافه می‌کنیم. سپس همه مقادیر را با استفاده از دستور foreach می‌خوانیم. چون کلاس ArrayList دارای انواع داده‌ای مختلفی است نمی‌توانیم از یک نوع داده‌ای خاص برای خواندن مقادیر استفاده کنیم. لذا برای این کار باید از نوع object که می‌تواند هر نوع داده‌ای در خود ذخیره کند استفاده نمود. در داخل حلقه از متد ()ToString برای نشان دادن مقادیر استفاده کرده‌ایم. به این نکته توجه کنید که برای دسترسی به هر عنصر می‌توانید از طریق اندیس آن قدام نمایید. کد زیر نحوه استفاده از حلقه for برای دسترسی به هر یک از اعضاء را نشان می‌دهد.

```
for (int i = 0; i < myArray.Count; i++)
{
    Console.WriteLine(myArray[i].ToString());
}
```

به خاصیت Count در کد بالا توجه کنید. این خاصیت درست شبیه به خاصیت Length آرایه معمولی است و کار آن شمارش تعداد عناصر شیء ArrayList می‌باشد. در کد بالا همانطور که نشان داده شده است، می‌توان به هر یک از عناصر با استفاده از اندیسشان دست یافت. نکته دیگر این است که شما می‌توانید به کلاس ArrayList یک ظرفیت ابتدایی بدهید. به عنوان مثال شما می‌توانید با استفاده از یک سازنده سربارگذاری شده نشان دهید که یک شیء ArrayList می‌تواند دارای ۵ عنصر باشد.

```
ArrayList myArray = new ArrayList(5);
```

کد بالا ۵ مکان خالی به وجود می‌آورد و شما می‌توانید با استفاده از متد ()Add یکی دیگر به آنها اضافه کنید. اگر همه مکآنها به وسیله مقادیر پر شوند می‌توان سایز شیء ایجاد شده از کلاس ArrayList را با استفاده از تغییر خاصیت Capacity آن تغییر داد. یکی دیگر از نسخه‌های سازنده کلاس ArrayList شیئی که رابط Icollection را اجرا می‌کند را قبول می‌کند. System.Array مثالی از این شیء است. بنابراین شما یک آرایه را به سازنده ارسال می‌کنید و مقادیر آن آرایه در شیء ArrayList کپی می‌شوند.

```
object[] array = {"John", 5, true, 3.65, 'R' };

ArrayList myArray = new ArrayList(array);
```

می‌توان با استفاده از متد Remove() کلاس ArrayList عناصر را پاک کرد. متد Remove() یک شیء که مطابق مقدار یک عنصر در آرایه است را قبول می‌کند. این متد به محض رسیدن به مقدار مورد نظر آن را حذف می‌کند. اگر عنصری را که مکانی غیر از مکان آخر آرایه باشد حذف کنید، بقیه عناصر بعد از آن عنصر مکان خود را تنظیم می‌کنند. به این معنی که فرض کنید آرایه ای دارای ۵ عنصر است و شما عنصر ۳ را حذف می‌کنید، در این صورت جای خالی این عنصر توسط عنصر ۴ و جای عنصر ۴ توسط عنصر ۵ پر می‌شود. به تکه کد زیر توجه کنید:

```
using System;
using System.Collections;

public class Program
{
    public static void Main()
    {
        ArrayList myArray = new ArrayList();

        myArray.Add("John");
        myArray.Add(5);
        myArray.Add(true);
        myArray.Add(3.65);
        myArray.Add('R');

        for (int i = 0; i < myArray.Count; i++)
        {
            Console.WriteLine("myArray[{0}] = {1}", i, myArray[i]);
        }

        //Remove element number 1
        myArray.Remove(5);

        Console.WriteLine("\nAfter removing myArray[1] (The value 5)...\n");

        for (int i = 0; i < myArray.Count; i++)
        {
            Console.WriteLine("myArray[{0}] = {1}", i, myArray[i]);
        }
    }
}
```

```
myArray[0] = John
myArray[1] = 5
myArray[2] = True
myArray[3] = 3.65
myArray[4] = R

After removing myArray[1] (The value 5)...

myArray[0] = John
myArray[1] = True
myArray[2] = 3.65
myArray[3] = R
```

از آنجاییکه در مثال بالا مقدار عنصر myArray[1] را حذف کرده‌ایم، همه عناصر متوالی در آرایه بالا مکان خود را تغییر می‌دهند. بنابراین عنصر myArray[2] جای myArray[1]، عنصر myArray[3] جای myArray[2] و ... را می‌گیرد. شما همچنین می‌توانید با استفاده از متد RemoveAt() به اندیس آرایه خاصی دست یافته و آن را حذف نمایید. این متد یک پارامتر قبول می‌کند و آن اندیس عنصری است که می‌خواهید از آرایه حذف کنید.

حذف و اضافه کردن چند آیتم

می‌توان با استفاده از متدهای AddRange() و RemoveRange() چندین آیتم را از آرایه حذف یا به آن اضافه نمود. متد AddRange() می‌تواند آرایه ای از چند مقدار را گرفته و آنها را به شیء ArrayList اضافه کند.

```
ArrayList myArray = new ArrayList();

myArray.Add(1);
myArray.Add(2);

int[] numbers = { 3, 4, 5 };

myArray.AddRange(numbers);

foreach(object element in myArray)
{
    Console.WriteLine(element);
}
```

```
1
2
3
4
5
```

متد RemoveRange() کاملاً با متد AddRange() متفاوت است. این متد دو پارامتر قبول می‌کند، اندیس عنصری که فرایند حذف از آن شروع می‌شود و تعداد عناصری که می‌خواهیم حذف کنیم. به عنوان مثال اگر بخواهید عناصر ۲ تا ۶ را حذف کنید باید تکه کد زیر را بنویسید:

```
myArray.RemoveRange(2, 5);
```

جستجوی مقادیر

با استفاده از متد Contains() می‌توان چک کرد که آیا یک مقدار خاص در داخل آرایه وجود دارد یا خیر. این متد یک آرگومان از نوع شیء را قبول کرده و اگر یک مقدار را در داخل لیست عناصر پیدا کند true را بر می‌گرداند، از متدهای IndexOf() و LastIndexOf() برای تشخیص اندیس یک مقدار خاص استفاده می‌شود. متد IndexOf() اندیس اولین محل وقوع یک مقدار خاص را بر می‌گرداند. متد LastIndexOf() اندیس آخرین محل وقوع یک مقدار خاص را بر می‌گرداند. هر دو متد، در صورتیکه مقدار مورد نظر را پیدا نکنند، مقدار ۱ را بر می‌گردانند. از متد BinarySearch() هم می‌توان برای جستجوی یک مقدار استفاده نمود. البته این متد برای جستجوی یک عنصر در داخل تعداد زیادی از عناصر مناسب است.

مرتب سازی مقادیر ArrayList

با استفاده از متد Sort() می‌توان مقادیر یک آرایه را مرتب نمود. اعداد از بزرگ به کوچک و رشته بر اساس حروف الفبا مرتب می‌شوند. اگر از این متد استفاده کنید همه اجزا با هم مقایسه می‌شوند. به عنوان مثال نمی‌توان یک رشته و یک عدد از نوع int را در داخل ArrayList قرار داد و آنها را با متد Sort() مرتب نمود. در درس آینده یاد خواهید گرفت که چگونه از یک مقایسه گر سفارشی برای مرتب کردن عناصر استفاده نمود.

ایجاد یک کلکسیون

سی‌شارپ به شما توانایی ایجاد کلکسیونی از کلاس‌ها را می‌دهد. به عنوان مثال می‌توان کلاسی ایجاد کرد که شامل چندین نمونه از کلاس‌های دیگر باشد. این کلاس خصوصیاتی مانند حذف و اضافه نمونه‌ها از کلکسیون را دارا می‌باشد. به مثال زیر توجه کنید:

```
using System.Collections;

public class Animal
{
    public string Name   { get; set; }
    public int    Age    { get; set; }
    public double Height { get; set; }

    public Animal(string name, int age, double height)
    {
        Name   = name;
        Age    = age;
        Height = height;
    }
}

public class Animals : CollectionBase
{
    public void Add(Animal newAnimal)
    {
        List.Add(newAnimal);
    }

    public void Remove(Animal oldAnimal)
    {
        List.Remove(oldAnimal);
    }
}
```

در مثال بالا دو کلاس تعریف شده است. اولین کلاس، کلاس Animal است که یک عنصر از کلکسیون کلاسمان است. دومین کلاس، کلکسیون کلاسمان است که شامل مجموعه‌ای از اشیاء کلاس Animal می‌باشد. برای استفاده بهتر از کاربرد کلکسیون‌ها، کلاسمان از کلاس CollectionBase ارث بری می‌کند. به این نکته توجه کنید که فضای نام System.Collections را قسمت تعریف فضاهای نامی وارد کنید. کلاس CollectionBase دارای متدهایی مانند Add() و Remove() و خاصیت List که مجموعه‌ای از اشیاء را در خود جای می‌دهد، می‌باشد. برای حذف یا اضافه کردن اشیاء به سادگی می‌توان از خاصیت List و متد مربوطه استفاده نمود. متدهای خاصیت List یک شیء قبول می‌کنند نه یک کلاس. بنابراین وقتی که یک ایندکسر تعریف می‌کنیم لازم است که ابتدا با استفاده از عمل cast نتیجه را تبدیل و سپس آن را به کاربر برگشت دهیم.

```
public Animal this[int index]
{
   get { return (Animal)List[index]; }
   set { List[index] = value; }
}
```

با استفاده از کد بالا می‌توانیم به هر یک از اشیاء کلکسیون بوسیله اندیس مکانشان دسترسی یابیم. از آنجاییکه خاصیت List، شیء قبول می‌کند لازم است که با استفاده از عمل cast آنها را به اشیاء Animal تبدیل کنیم. به برنامه زیر توجه کنید:

```
public class Program
{
    public static void Main()
    {
        Animals animalCollection = new Animals();

        animalCollection.Add(new Animal("Jack" , 10, 100));
        animalCollection.Add(new Animal("Sussy", 5 , 10));
        animalCollection.Add(new Animal("Frank", 3 , 5));

        for (int i = 0; i < animalCollection.Count; i++)
        {
            Console.WriteLine("Animal  {0}"  , i + 1);
            Console.WriteLine("Name:   {0}"  , animalCollection[i].Name);
            Console.WriteLine("Age:    {0}"  , animalCollection[i].Age);
            Console.WriteLine("Height: {0}\n", animalCollection[i].Height);
        }
    }
}
```

```
Animal 1
Name: Jack
Age: 10
Height: 100

Animal 2
Name: Sussy
Age: 5
Height: 10

Animal 3
Name: Frank
Age: 3
Height: 5
```

در برنامه بالا یک نمونه از کلاس Animals(animalCollection) و سپس سه نمونه به آن با استفاده از متد Add() اضافه کرده‌ایم. سپس با گردش در میان عناصر با استفاده از یک حلقه نشان داده‌ایم، که می‌توان با استفاده از اندیس به آنها دست یافت. این کار را با استفاده از یک حلقه foreach هم می‌توان انجام داد:

```
foreach (Animal animal in animalCollection)
{

}
```

در درس آینده با generic ها آشنا می‌شوید، که راهی آسان برای ایجاد کلکسیونی از هر نوع، بدون ایجاد کلاسی که از کلاس پایه CollectionBase ارث بری کند، می‌باشد.

ساخت دیکشنری

می‌توان یک کلاس ایجاد کرد که از کلاس DictionaryBase مشتق شود. با این روش شما می‌توانید به هر عنصر با استفاده از یک کلید (key) (که معمولاً از نوع رشته است) دسترسی یابید. این کلیدها دارای یک مقدار (value) وابسته به خود هستند. برای فراخوانی هر یک از آیتم‌های دیکشنری از کلید آن استفاده می‌کنیم نه از اندیس آن. به کد زیر توجه کنید:

```
1   using System.Collections;
```

```
2
3   public class Animal
4   {
5       public string Name   { get; set; }
6       public int    Age    { get; set; }
7       public double Height { get; set; }
8
9       public Animal(string name, int age, double height)
10      {
11          Name   = name;
12          Age    = age;
13          Height = height;
14      }
15  }
16
17  public class Animals : DictionaryBase
18  {
19      public void Add(string key, Animal newAnimal)
20      {
21          Dictionary.Add(key, newAnimal);
22      }
23
24      public void Remove(string key)
25      {
26          Dictionary.Remove(key);
27      }
28
29      public Animal this[string key]
30      {
31          get { return (Animal)Dictionary[key]; }
32          set { Dictionary[key] = value; }
33      }
34  }
```

یک کلاس به نام Animal تعریف کرده‌ایم که قرار است در کلاس دیکشنری مورد استفاده قرار بگیرد (خطوط ۱۵-۳). سپس یک کلاس به نام Animals ایجاد می‌کنیم (نام یکی از کلاس‌ها Animal و دیگری Animals است) که از کلاس DictionaryBase ارث بری می‌کند. حال می‌توان متدهایی برای حذف و اضافه ورودی‌های دیکشنری تعریف کرد. متد Add() به یک کلید از نوع رشته و یک شیء Animal که می‌خواهیم به دیکشنری اضافه کنیم، احتیاج دارد. از خاصیت Dictionary کلاس DictionaryBase برای اضافه کردن یک ورودی به دیکشنری استفاده می‌کنیم. متد Remove() فقط به کلید شیء ای‌ای که می‌خواهیم حذف کنیم نیاز دارد. در این متد هم از خاصیت Dictionary برای حذف آیتم‌ها استفاده می‌کنیم. ما یک ایندکسر تعریف کرده‌ایم که به جای یک مقدار int، یک کلید از نوع رشته قبول می‌کند. اجازه بدهید برای نشان دادن توانایی‌های کلاس دیکشنری مان یک کلاس ایجاد کنیم.

```
using System;

namespace DictionariesDemo
{
    public class Program
    {
        public static void Main()
        {
            Animals animalDictionary = new Animals();

            animalDictionary.Add("Animal1", new Animal("John" , 10, 100));
            animalDictionary.Add("Animal2", new Animal("Sussy", 5 , 10));
            animalDictionary.Add("Animal3", new Animal("Frank", 3 , 5));
            animalDictionary.Add("Animal4", new Animal("Mark" , 7 , 15));

            Console.WriteLine("Accessing entries by their keys");
```

```
            for (int i = 0; i < animalDictionary.Count; i++)
            {
                Console.WriteLine(animalDictionary["Animal" + (i + 1)].Name);
            }

            animalDictionary.Remove("Animal3");
            Console.WriteLine("\nFrank was removed from the dictionary.");

            Console.WriteLine("\nIterating using foreach loop.");
            foreach (DictionaryEntry animal in animalDictionary)
            {
                Console.WriteLine((animal.Value as Animal).Name);
            }
        }
    }
}
```

```
Accessing entries by their keys
John
Sussy
Frank
Mark

Frank was removed from the dictionary.

Iterating using foreach loop.
John
Mark
Sussy
```

یک نمونه جدید از کلاس دیکشنری `Animals` ایجاد می‌کنیم. سپس چند نمونه از کلاس `Animal` و کلیدهای مربوط به آنها را به آن اضافه می‌کنیم. همانطور که در مثال بالا مشاهده می‌کنید می‌توانیم با استفاده از حلقه `for` به هر یک از ورودی با استفاده از کلیدشان که حکم اندیس دارد، دست یابیم. سپس با استفاده از متد `()Remove` سومین ورودی دیکشنری را با استفاده از کلیدش حذف کرده‌ایم. و بعد مقادیر را با استفاده از یک حلقه `foreach` می‌خوانیم و چاپ می‌کنیم. شما باید به یک تفاوت توجه کنید. ما از `DictionaryEntry` به عنوان نوع استفاده کرده‌ایم چون هر آیتم در کلاس `dictionary` یک نوع دارد. شیء واقعی در داخل خاصیت `Value`، `DictionaryEntry` قرار دارد. این مقدار را با استفاده از عمل `cast` به نوع مناسب تبدیل می‌کنیم.

Hashtable در سی‌شارپ

از `Hashtable` زمانی استفاده می‌شود که بخواهید اطلاعات را بر اساس کلید/مقدار ذخیره کنید. به عنوان مثال نام دانش آموز و نمره او در امتحان. `Hashtable` به شما اجازه تلفیق متن و عدد را می‌دهد. یک پروژه جدید ایجاد کنید. با زدن دکمه F7 به محیط کدنویسی رفته و در بالای کدها و در قسمت تعریف فضای نام، فضای نام زیر را وارد کنید:

```
using System.Collections;
```

`Hashtable` در این فضای نام قرار دارد. کد زیر را هم در داخل متد `()Main` وارد کنید.

```
Hashtable students = new Hashtable();
```

کد بالا یک شئ به نام students ایجاد می‌کند. دو راه برای اضافه کردن داده‌ها به Hashtable وجود دارد. این دو روش در زیر نشان داده شده‌اند:

```
students["Jenny"] = 87;
students["Peter"] = "No Score";
students["Mary Jane"] = 64;
students["Azhar"] = 79;
```

یا

```
students.Add("Jenny", 87);
students.Add("Peter", "No Score");
students.Add("Mary Jane", 64);
students.Add("Azhar", 79);
```

در روش اول از براکت استفاده شده است:

```
students["Jenny"] = 87;
```

در داخل براکت‌ها کلید را تایپ می‌کنید. که در این مورد خاص "jenny" می‌باشد. سپس بعد از علامت مساوی مقدار کلید را می‌نویسید. به این نکته توجه کنید که سه ورودی بالا دارای مقدار عددی بوده به جزء Peter که دارای مقدار متنی می‌باشد. در روش دوم مقادیر یا استفاده از متد Add() در Hashtable ذخیره می‌شوند:

```
students.Add("Jenny", 87);
```

در بین دو پرانتز Add()، ابتدا نام کلید سپس کاما و بعد از کاما مقدار کلید را می‌نویسید. تفاوتی بین این دو وجود دارد. اگر از متد Add() استفاده کنید نمی‌توانید از کلیدهای مشابه استفاده کنید. اما هنگام استفاده از براکت می‌شود.

پیغام خطا می‌دهد:

```
students.Add("Jenny", 87);
students.Add("Jenny", 35);
```

پیغام خطا نمی‌دهد:

```
students["Jenny"] = 87;
students["Jenny"] = 35;
```

حال در زیر شئ ایجاد شده کدهای زیر را هم اضافه کنید:

```
Hashtable students = new Hashtable();
students["Jenny"] = 87;
students["Peter"] = "No Score";
students["Mary Jane"] = 64;
students["Azhar"] = 79;

foreach (DictionaryEntry child in students)
{
    Console.WriteLine("student: " + child.Key + " , Score: " + child.Value);
}
```

قبل از اجرای کد به حلقه foreach توجه کنید. در داخل پرانتزها از دستور زیر استفاده کرده‌ایم:

```
DictionaryEntry child
```

کد بالا یک متغیر به نام child را که نوع آن یک DictionaryEntry می‌باشد ایجاد می‌کند. سی‌شارپ با استفاده از یک شیء از این نوع در هنگام کار با Hashtable، مقادیر و کلیدها را برگشت می‌دهد. در مثال فوق مقادیر و کلیدها را در داخل کنترل listbox نشان می‌دهیم.

```
"student: " + child.Key + " , Score: " + child.Value
```

بعد از نوشتن نام متغیرمان در این مثال (child)، IntelliSense ظاهر می‌شود. Key خاصیتی است که نام کلید و Value خاصیتی است که مقدار کلید را برگشت می‌دهد. نتیجه اجرای برنامه به صورت زیر است:

```
student: Mary Jane , Score:64
student: Jenny , Score:87
student: Peter , Score: "No Score"
student: Azhar , Score: 79
```

همانند یک لیست (List) می‌توانید با استفاده از متدهای ()Add و ()Remove آیتم‌هایی به Hashtable اضافه و یا از آن کم کنید. مانند زیر:

```
students.Remove("Peter");
```

همانطور که می‌بینید با استفاده از نام کلید، نه مقدار، یک آیتم را می‌توان حذف نمود.

انواع Enumerator و Enumerable

در درس‌های قبلی دیدید که چگونه با استفاده از حلقه foreach عناصر یک آرایه را پیمایش می‌کردیم. در این درس می‌خواهیم کمی دقیق‌تر به قضیه آرایه‌ها نگاه کنیم و ببینیم که چرا آنها می‌توانند توسط این حلقه مورد پیمایش قرار گیرند. همچنین یاد می‌گیرید که چطور از این قابلیت در کلاس‌هایی که خودتان تعریف کرده‌اید، استفاده کنید.

استفاده از حلقه foreach

وقتی که از حلقه foreach در یک آرایه استفاده می‌کنید، این حلقه تک تک اعضای آرایه را به شما ارائه داده و اجازه می‌دهد که مقادیر آنها را مشاهده کنید. به عنوان مثال در زیر یک آرایه با چهار عنصر تعریف شده است و می‌خواهیم با استفاده از حلقه foreach مقادیر عناصر آن را چاپ کنیم:

```
int[] number = { 10, 11, 12, 13 };
foreach (int item in number)
Console.WriteLine("Item value: {0}", item);
```

```
Item value: 10
Item value: 11
Item value: 12
Item value: 13
```

اما چرا از این حلقه برای آرایه‌ها استفاده می‌کنیم؟ چون آرایه به محض درخواست، یک شیء به نام enumerator (شمارنده) تولید می‌کند. این شیء می‌تواند عناصر آرایه را یک به یک و به ترتیب برگرداند. Enumerator ترتیب عناصر آرایه را می‌داند و بعد از برگرداندن اولین عنصر موقعیت خود را حفظ کرده و در درخواست بعدی عنصر بعدی آرایه را بر می‌گرداند. برای انواعی که دارای شئ شمارنده (enumerator) هستند، یک راه برای به دست آوردن این شئ وجود دارد و آن استفاده از متد ()GetEnumerator است. به انواعی که این متد را پیاده سازی می‌کنند نوع شمارش پذیر یا enumerable می‌گویند. آرایه یک نوع شمارش پذیر است. از حلقه foreach برای کار با انواع شمارش پذیر (enumerable) استفاده می‌شود. وقتی که یک نوع شمارش پذیر به این حلقه می‌دهیم تا عناصر آن را شمارش کند، مراحل زیر طی می‌شود:

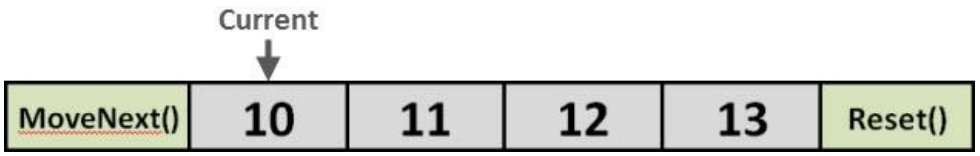
- ابتدا شئ enumerator را با فراخوانی متد ()GetEnumerator به دست می‌آورد .
- هر آیتم را از enumerator درخواست کرده و آن را برای شما قابل دسترس می‌کند (نه قابل تغییر).

رابط‌های IEnumerator و IEnumerable

تمامی کلاس‌هایی که به نحوی شامل یک Collection هستند، این دو رابط رو پیاده سازی می‌کنند. وجود IEnumerable که توسط کلاس‌ها پیاده سازی می‌شود به کلاس این امکان را می‌دهد که، بصورت ضمنی و توکار بشود شیء را پیمایش کرد. دقیقاً به همین دلیل می‌توان با استفاده از حلقه foreach یک آرایه را پیمایش کرد، چون که رابط IEnumerable توسط کلاس System.Array پیاده سازی می‌شود. رابط IEnumerator در سطح پایین تری از یک IEnumerable قرار دارد. با استفاده از این رابط می‌توان در هرجای بدنه متد اشیایی را برگشت دهیم، بدون اینکه مجبور باشیم، ابتدا نتایج را مثلاً در یک آرایه بریزیم و بعد آن آرایه را برگشت دهیم.

رابط IEnumerator

یک شمارنده (enumerator) رابط IEnumerator را پیاده سازی می‌کند که دارای دو متد ()MoveNext و ()Reset و یک خاصیت به نام Current می‌باشد. خاصیت Current عنصر جاری یک مجموعه را بر می‌گرداند. این خاصیت یک خاصیت فقط خواندن (read-only) است و چیزی که برمی گرداند از نوع object می‌باشد. متد ()MoveNext، متدی است که، مکان شمارنده را از یک آیتم در مجموعه به آیتم بعدی منتقل می‌کند. این متد یک مقدار بولی را بر می‌گرداند، که نشان می‌دهد که آیا مکان دیگری برای خواندن در دسترس است یا به انتهای مجموعه رسیده است. اگر مکان جدیدی وجود داشته باشد مقدار true و در غیر اینصورت مقدار false را بر می‌گرداند. مکان اولیه شمارنده، قبل از اولین آیتم مجموعه است، بنابراین ()MoveNext باید قبل از اولین دسترسی خاصیت Current فراخوانی شود. متد ()Reset، متدی برای برگرداندن شمارنده به مکان اولیه خود قبل از جابجایی است. به تعبیری دیگر، مکان اولیه مجموعه را برمی گرداند. با در اختیار داشتن یک شمارنده (enumerator) شما قادر خواهید بود که حلقه foreach را شبیه سازی کرده و عناصر یک مجموعه را با استفاده ازمتد ()MoveNext و خاصیت Current پیمایش کنید. برای درک بهتر عملکرد دو متد و خاصیت مذکور به شکل و کد زیر توجه کنید:



```
int[] numbers = { 10, 11, 12, 13 };

IEnumerator IEnumerator1 = numbers.GetEnumerator();
IEnumerator1.MoveNext();
int i = (int)IEnumerator1.Current;

Console.WriteLine(i.ToString());
```

```
10
```

همانطور که در کد بالا مشاهده می‌کنید متد GetEnumerator() آرایه numbers را به نوع شمارش پذیر تبدیل می‌کند، سپس با فراخوانی متد MoveNext()، عدد ۱۰ که اولین عضو آرایه است، به عنوان عنصر جاری (Current) بر گردانده می‌شود. حال فرض کنید که شما می‌خواهید عدد ۱۲ را چاپ کنید، برای این کار باید متد MoveNext() را سه با فراخوانی کنید:

```
int[] numbers = { 10, 11, 12, 13 };

IEnumerator IEnumerator1 = numbers.GetEnumerator();

IEnumerator1.MoveNext();
IEnumerator1.MoveNext();
IEnumerator1.MoveNext();

int i = (int)IEnumerator1.Current;

Console.WriteLine(i.ToString());
```

```
12
```

آرایه‌ها از انواع قابل شمارش (enumerable) هستند، بنابراین کد زیر روش دستی کاری است که حلقه foreach به صورت خودکار انجام می‌دهد. در حقیقت کامپایلر C# کدی شبیه به کد زیر را در هنگام نوشتن دستور foreach تولید می‌کند :

```
using System;
using System.Collections;

public class Program
{
    public static void Main()
    {
        int[] numbers = { 10, 11, 12, 13 };

        IEnumerator IEnumerator1 = numbers.GetEnumerator();

        while (IEnumerator1.MoveNext())
        {
            int i = (int)IEnumerator1.Current;
            Console.WriteLine("{0}", i);
        }
    }
}
```

```
10
11
12
13
```

رابط IEnumerable

یک کلاس قابل شمارش (enumerable)، کلاسی است که رابط IEnumerable را پیاده سازی کند. رابط IEnumerable فقط یک عضو دارد و آن عضو هم متد GetEnumerator() می‌باشد و پارامتر برگشتی این متد از نوع همان رابط IEnumberator است. این رابط در واقع کلاس ما را قابل پیمایش می‌کند تا بتوانیم حلقه foreach را در مورد کلاسمان بکار ببریم. فرم کلی بصورت زیر است:

```
using System.Collections;

class MyClass : IEnumerable
{
    public IEnumerator GetEnumerator
    {
        ...
    }
    ...
}
```

اکنون یک مثال را با هم مرور می‌کنیم. فرض کنید یک کلاس به نام ColorEnumerator که رابط IEnumerator را پیاده سازی می‌کند. این کلاس یک رشته از رنگ‌ها را در بر می‌گیرد، و چون رابط IEnumerator را پیاده سازی می‌کند، پس قابل پیمایش می‌شود:

```
using System;
using System.Collections;

class ColorEnumerator : IEnumerator
{
    string[] Colors;
    int Position = -1;
    public ColorEnumerator(string[] theColors) // Constructor
    {
        Colors = new string[theColors.Length];
        for (int i = 0; i < theColors.Length; i++)
            Colors[i] = theColors[i];
    }

    public object Current                                   // Implement Current.
    {
        get
        {
            return Colors[Position];
        }
    }

    public bool MoveNext()                                  // Implement MoveNext.
    {
        if (Position < Colors.Length - 1)
        {
            Position++;
            return true;
        }
        else
            return false;
    }

    public void Reset()                                     // Implement Reset.
    {
        Position = -1;
    }
}
```

اکنون ما کلاسی دیگر ایجاد می‌کنیم که رابط IEnumerable را پیاده سازی می‌کند، این کلاس در متد GetEnumerator() پارامتری از نوع کلاس بالا برمی گرداند مطابق شکل زیر:

```
class MyColors : IEnumerable
{
    string[] Colors = { "Red", "Yellow", "Blue" };
    public IEnumerator GetEnumerator()
    {
        return new ColorEnumerator(Colors);
    }
}
```

حال در برنامه براحتی می‌توانیم از حلقه foreach برای پیمایش اعضای آن استفاده کنیم. به کد زیر توجه کنید :

```
class Program
{
    static void Main()
    {
        MyColors MC = new MyColors();
        foreach (string color in MC)
            Console.WriteLine(color);
    }
}
```

```
Red
Yellow
Blue
```

پیمایشگر (Iterator)

Iterator بلوک کدی است که شامل همه مقادیری است که در یک حلقه foreach مورد استفاده قرار می‌گیرد. یک کلاس که نماینده یک کلکسیون است می‌تواند رابط System.Collections.IEnumerable را پیاده سازی کند. این رابط نیاز به پیاده سازی متد GetEnumerator() دارد که یک رابط IEnumerator را بر می‌گرداند. رابط IEnumerator دارای خاصیت Current می‌باشد که مقدار جاری برگشت داده شده بوسیله Iterator را در بر دارد. این رابط (IEnumerator) همچنین دارای متد MoveNext() است که خاصیت Current را به سوی آیتم بعدی حرکت می‌دهد و در صورت عدم وجود آیتم مقدار false را بر می‌گرداند. متد Reset() حلقه پیمایش را به اولین آیتم بر می‌گرداند. رابط IEnumerator توسط کلکسیون‌های مختلف دات‌نت پیاده سازی می‌شود که یکی از این کلکسیون‌ها آرایه‌های، می‌باشند. پس این سؤال پیش می‌آید که چگونه با استفاده از حلقه foreach می‌توان به اجزای این کلکسیون‌ها دسترسی پیدا کرد؟ فرض کنید کدی شبیه به کد زیر داریم که همه عناصر یک آرایه را با استفاده از حلقه foreach می‌خواند.

```
int[] numbers = { 1, 2, 3, 4, 5 };

foreach (int n in numbers)
{
    Console.WriteLine(n);
}
```

برای درک بهتر کاربرد تکرارکننده‌ها (iterators)، اجازه دهید که حلقه foreach کد بالا را به فراخوانی متد GetEnumerator() آرایه ترجمه کنیم:

```
int[] numbers = { 1, 2, 3, 4, 5 };

IEnumerator iterator = numbers.GetEnumerator();
while (iterator.MoveNext())
{
    Console.WriteLine(iterator.Current);
}
```

همانطور که مشاهده می‌کنید ما ابتدا یک پیمایشگر آرایه را با استفاده از متد GetEnumerator() که یک رابط IEnumerator را بر می‌گرداند به دست می‌آوریم. سپس از این پیمایشگر در حلقه while استفاده کرده و متد MoveNext() را فراخوانی می‌کنیم. متد MoveNext() اولین عنصر یک کلکسیون مانند آرایه را بر می‌گرداند و اگر عملیات به دست آوردن اولین عنصر موفقیت آمیز باشد مقدار true را بر می‌گرداند. در فراخوانی بعدی دومین عنصر آرایه را بر می‌گرداند و این کار را تا آخرین عنصر آرایه انجام می‌دهد و وقتی که به پایان عناصر رسید مقدار false را برگشت می‌دهد. مقدار برگشت داده شده یک عنصر به وسیله خاصیت IEnumerator.Current قابل دسترسی است. برای استفاده از Iterator نیاز به دستور yield return داریم. این دستور (yield) با دستور return متفاوت است. یکی از تفاوت‌های مشهود این دو، استفاده از کلمه کلیدی yield قبل از کلمه کلیدی return می‌باشد. yield یک عنصر مجموعه را برمی گرداند و موقعیت مکان نما را به عنصر بعدی هدایت می‌کند. به کد زیر توجه کنید:

```
public static IEnumerable GetMessages()
{
    yield return "Message 1";
    yield return "Message 2";
    yield return "Message 3";
}

public static void Main()
{
    foreach (string message in GetMessages())
    {
        Console.WriteLine(message);
    }
}
```

```
Message 1
Message 2
Message 3
```

متد GetMessages() یک شئ IEnumerable را بر می‌گرداند که شامل تعریفی برای یک متد GetEnumerator() می‌باشد. مقدار جلوی اولین دستور yield return در متغیر message دستور foreach قرار می‌گیرد و چاپ می‌شود. در فراخوانی بعدی متد GetMessages() در حلقه foreach مقدار جلوی دومین دستور yield return چاپ می‌شود و این کار تا آخرین دستور yield return ادامه می‌یابد. برای توقف مقادیر برگشتی از متد می‌توان از دستور yield break به صورت زیر استفاده کرد:

```
public static IEnumerable GetMessages()
{
     yield return "Message 1";
     yield return "Message 2";
     yield break;
     yield return "Message 3";
}
```

حال که با نحوه عملکرد پیمایشگرها آشنا شدید، شما را با نحوه ایجاد یک پیمایشگر سفارشی آشنا می‌کنیم. اجازه دهید با ذکر یک مثال نحوه استفاده از پیمایشگر را به وسیله ایجاد یک کلاس جدید که شامل یک فیلد ArrayList است توضیح دهیم. می‌خواهیم یک پیمایشگر ایجاد کنیم که با استفاده از حلقه foreach مقادیر ArrayList را به دست آورد .

```
1  using System.Collections;
2  using System;
3
4  public class Names : IEnumerable
5  {
6      private ArrayList innerList;
7
8      public Names(params object[] names)
9      {
10         innerList = new ArrayList();
11
12         foreach (object n in names)
13         {
14             innerList.Add(n);
15         }
16     }
17
18     public IEnumerator GetEnumerator()
19     {
20         foreach (object n in innerList)
21         {
22             yield return n.ToString();
23         }
24     }
25 }
26
27 public class Program
28 {
29     public static void Main()
30     {
31         Names nameList = new Names("John", "Mark", "Lawrence", "Michael", "Steven");
32
33         foreach (string name in nameList)
34         {
35             Console.WriteLine(name);
36         }
37     }
38 }
```

```
John
Mark
Lawrence
Michael
Steven
```

ابتدا یک کلاس مجموعه‌ای به نام Names که شامل لیستی از نام‌ها می‌باشد را، ایجاد می‌کنیم (خطوط ۲۵-۴). همانطور که در تعریف کلاس در خط ۴ مشاهده می‌کنید، ما کلاس CollectionBase را پیاده سازی نکرده‌ایم. چون که کلاس CollectionBase رابط IEnumerable را پیاده سازی می‌کند و در نتیجه دارای یک پیاده سازی از متد GetEnumerator() این رابط نیز است. ما یک کلاس مجموعه‌ای را، از صفر ایجاد و یک پیمایشگر سفارشی را تعریف کرده‌ایم. از آنجاییکه کلاس ما رابط IEnumerable را پیاده سازی کرده است پس لازم است که، متد GetEnumerator() از این رابط را هم پیاده سازی کند. متد GetEnumerator() یک شمارنده است که کلکسیون‌ها را شمارش می‌کند. کلکسیون به مجموعه‌ای از عناصر هم نوع که الزاماً در حافظه پشت سر هم نیستند گفته می‌شود. در خطوط 18-24 پیمایشگر سفارشی را تعریف کرده‌ایم. در داخل آن به پیمایش هر یک از مقادیر فیلد innerList می‌پردازیم. هر مقدار به رشته تبدیل و سپس با استفاده از دستور yield به متد

فراخوان ارسال می‌شود. خطوط ۳۶ – ۳۳ پیمایشگر ما را در عمل نمایش می‌دهد. از آنجاییکه دستور yield در پیمایشگرمان هر مقدار را به نوع رشته تبدیل می‌کند پس به راحتی می‌توانیم از نوع string در حلقه foreach استفاده کنیم (خط ۳۳). وقتی که یک مقدار به وسیله پیمایشگر از طریق حلقه foreach واقع در متد Main() برگردانده می‌شود، حلقه foreach به آیتم بعدی رفته و دستور yield مقدار پیمایش شده از innerList را بر می‌گرداند. بدون پیمایشگر ما قادر به استفاده از حلقه foreach در کلاسمان نیستیم. ایجاد پیمایشگر سفارشی، به ما این قدرت را می‌دهد که کنترل بیشتری بر رفتار حلقه foreach هنگام کار با کلاس داشته باشیم. به عنوان مثال می‌توانیم پیمایشگر را به این صورت اصلاح کنیم که، فقط نام‌هایی که با حرف M شروع می‌شوند را برگرداند:

```
public IEnumerator GetEnumerator()
{
    foreach (object n in innerList)
    {
        if (n.ToString().StartsWith("M"))
            yield return n.ToString();
    }
}
```

همانطور که در کد بالا مشاهده می‌کنید برای تشخیص اینکه چه نامی با حرف M شروع شده است از متد StartsWith() مربوط به کلاس System.String استفاده کرده‌ایم. اگر نام با حرف M شروع شده باشد دستور yield آن را بر می‌گرداند در غیر اینصورت از آن رد شده و به نام بعدی در innerList را مورد بررسی قرار می‌دهد. با دستکاری متد GetEnumerator()، هنگام استفاده از حلقه foreach در یک نمونه از کلاس Names، می‌توانیم فقط نام‌هایی را که با حرف M شروع می‌شوند را به دست آوریم. اگر چه می‌توانید از تکنیک بالا استفاده کرده و متد GetEnumerator() تغییر دهید ولی بهتر است از یک متد جدا برای به دست آوردن نام‌هایی که با یک حرف خاص شروع می‌شوند استفاده کنید:

```
public IEnumerable GetNamesStartingWith(string letter)
{
    foreach (object n in innerList)
    {
        if (n.ToString().StartsWith(letter))
            yield return n.ToString();
    }
}
```

پیمایشگر بالا نسبت به پیمایشگر قبلی منعطف تر بوده و شما می‌توانید با استفاده از آن نام‌هایی که با یک حرف یا زیررشته خاص شروع شده‌اند را پیدا کنید. به این نکته توجه کنید که در پیمایشگر بالا از IEnumerable به جای IEnumerator استفاده کرده‌ایم. IEnumerable دارای متد GetEnumerator() است. هنگام فراخوانی این پیمایشگر لازم است که حلقه foreach خطوط ۳۶-۳۳ را به صورت زیر تغییر دهید:

```
foreach (string name in nameList.GetNamesStartingWith("M"))
{
    Console.WriteLine(name);
}
```

کلکسیون‌های عمومی(Generic Collections)

می‌توان یک کلکسیون عمومی تعریف کرد که شامل هر نوع داده‌ای باشد. برای ایجاد یک کلکسیون عمومی از کلاس List<T> مربوط به فضای نامی System.Collections.Generics استفاده می‌شود. List<T> می‌تواند مجموعه‌ای از اشیاء نوع T باشد. در نتیجه List<int>

مجموعه‌ای از مقادیر صحیح است. کلاس List<T> دارای متدهای ()AddRange، ()Remove، ()RemoveAt و دیگر متدهایی است که در کلاس کلکسیون مان در درس قبلی از آنها استفاده کردیم.

```
using System;
using System.Collections.Generic;

public class Animal
{
    public string Type;

    public Animal(string type)
    {
        Type = type;
    }
}
public class Program
{
    public static void Main()
    {
        List<Animal> animals = new List<Animal>();

        animals.Add(new Animal("Dog"));
        animals.Add(new Animal("Cat"));
        animals.Add(new Animal("Rat"));

        foreach (Animal animal in animals)
        {
            Console.WriteLine(animal.Type);
        }
    }
}
```

```
Dog
Cat
Rat
```

همچنین می‌توانید از کلاس Dictionary<TKey,TVal> که در فضای نامی ذکر شده قرار دارد، استفاده کنید. Tkey نوع کلید و Tval نوع مقدار را مشخص می‌کند.

```
using System;
using System.Collections.Generic;

public class Animal
{
    public string Type;

    public Animal(string type)
    {
        Type = type;
    }
}
public class Program
{
    public static void Main()
    {
        Dictionary<string, Animal> animals = new Dictionary<string, Animal>();

        animals.Add("Animal1", new Animal("Dog"));
        animals.Add("Animal2", new Animal("Cat"));
        animals.Add("Animal3", new Animal("Rat"));

        foreach (Animal animal in animals.Values)
        {
```

```
            Console.WriteLine(animal.Type);
        }
    }
}
```

یک دیکشنری تعریف کرده‌ایم که دارای کلیدهایی از نوع string و مقادیری از نوع کلاس Animal می‌باشد. برای به دست آوردن مقدار آیتم‌های دیکشنری می‌توان از خاصیت Values که شامل همه آیتم‌های دیکشنری است، استفاده کرد.

خاصیت Keys نیز برای به دست آوردن همه کلیدهای یک دیکشنری به کار می‌رود. کلاس‌هایی را که معرفی کردیم بسیار شبیه به کلاس‌های collection و dictionary هستند که از کلاس‌های CollectionBase و DictionaryBase ارث بری می‌کنند. اما نیازی به ساخت کلاس‌های کالکشن بالا نیست. فقط کافی است که نوع آیتم مورد نظر را به کلاس‌های List<T> و Dictionary<TKey,Tvalue> ارسال نمایید. برای اینکه از قابلیت‌های بیشتری بهره مند شوید، می‌توانید از collection و dictionary سفارشی استفاده نمایید (یعنی کلاس‌هایی که از CollectionBase و DictionaryBase ارث بری می‌کنند و شما خود متدهای Add() و Remove() آنها را پیاده سازی می‌کنید).

جنریک‌ها (Generics)

جنریک‌ها کلاس‌ها، متدها یا رابط‌هایی هستند که بسته به نوع داده‌ای که به آنها اختصاص داده می‌شود رفتارشان را سازگار می‌کنند. به عنوان مثال می‌توان یک متد جنریک تعریف کرد که هر نوع داده‌ای را قبول کند. همچنین می‌توان یک متد ایجاد کرد که بسته به نوع دریافتی، مقادیری از انواع داده‌ای مانند int، double یا string را نشان دهد. اگر از جنریک‌ها استفاده نکنید باید چند متد و یا حتی چندین متد سربارگذاری شده برای نمایش هر نوع ممکن ایجاد کنید.

```
public void Show(int number)
{
    Console.WriteLine(number);
}

public void Show(double number)
{
    Console.WriteLine(number);
}

public void Show(string message)
{
    Console.WriteLine(message);
}
```

با استفاده از جنریک‌ها می‌توان متد جنریکی ایجاد کرد که هر نوع داده‌ای را قبول کند.

```
public void Show<E>(E item)
{
    Console.WriteLine(item);
}
```

متدهای جنریک را در درس‌های آینده توضیح خواهیم داد. حتماً این سؤال را از خودتان می‌پرسید که چرا نباید از نوع آبجکت که هر نوع داده‌ای را قبول می‌کند استفاده کنیم؟ در آینده مشاهده می‌کنید که با استفاده از جنریک‌ها نیاز به عمل cast (تبدیل صریح) ندارید. درباره جنریک‌ها در درس‌های بعد مطالب بیشتری توضیح می‌دهیم.

متدهای جنریک

اگر بخواهید چندین متد با عملکرد مشابه ایجاد کنید و فقط تفاوت آنها در نوع داده‌ای باشد که قبول می‌کنند (مثلاً یکی نوع int و دیگری نوع double را قبول کند) می‌توان از متدهای جنریک برای صرفه جویی در کدنویسی استفاده کرد. ساختار عمومی یک متد جنریک به شکل زیر است:

```
returnType methodName<type> (type argument1)
{
    type someVariable;
}
```

مشاهده می‌کنید که بعد از نام متد یک نوع در داخل دو علامت بزرگ‌تر و کوچک‌تر آمده است (<type>) که همه انواع در سی‌شارپ می‌توانند جایگزین آن شوند. برنامه زیر مثالی از نحوه استفاده از متد جنریک می‌باشد:

```
1   using System;
2
3   public class Program
4   {
5       public static void Show<X>(X val)
6       {
7           Console.WriteLine(val);
8       }
9
10      public static void Main()
11      {
12          int    intValue    = 5;
13          double doubleValue = 10.54;
14          string stringValue = "Hello";
15          bool   boolValue   = true;
16
17          Show(intValue);
18          Show(doubleValue);
19          Show(stringValue);
20          Show(boolValue);
21      }
22  }
```

```
5
10.54
Hello
true
```

یک متد جنریک ایجاد کرده‌ایم که هر نوع داده‌ای را قبول کرده و مقادیر آنها را نمایش می‌دهد (خطوط ۸-۵). سپس داده‌های مختلفی با وظایف یکسان به آن ارسال می‌کنیم. متد نیز نوع X را بسته به نوع داده‌ای که به عنوان آرگومان ارسال شده است تغییر می‌دهد. به عنوان مثال وقتی یک داده از نوع int ارسال می‌کنیم، همه مکآنهایی که X در آنها وجود دارد به int تبدیل می‌شوند و متد به صورت زیر در می‌آید:

```
public static void Show (int val)
{
    Console.WriteLine(val);
}
```

همچنین هنگام فراخوانی متد جنریک صریحاً می‌توانید نوعی را که به وسیله آن مورد استفاده قرار می‌گیرد ذکر کنید (البته لازم نیست). به عنوان مثال فراخوانی‌های متد بالا را می‌توان به صورت زیر هم نوشت:

```
Show<int>   (intValue);
Show<double>(doubleValue);
Show<string>(stringValue);
Show<bool>  (boolValue);
```

به یک نکته در مورد استفاده از متدهای جنریک توجه کنید و آن این است که شما نمی‌توانید در داخل کدهای مربوط به متد محاسبات انجام دهید مثلاً دو عدد را با هم جمع کنید، چون کامپایلر نمی‌تواند نوع واقعی عملوندها را تشخیص دهد، ولی به سادگی می‌توان مقادیر را در داخل متد نشان داد. چون کامپایلر هر نوع داده‌ای را که توسط متد Console.WriteLine() استفاده می‌شود را می‌تواند تشخیص دهد.

```
public static void Show<X>(X val1, X val2)
{
    Console.WriteLine(val1 + val2);
}
```

شما می‌توانید چندین نوع خاص را برای متد جنریک ارسال کنید، برای این کار هر نوع را به وسیله کاما از دیگری جدا کنید.

```
public static void Show<X, Y>(X val1, Y val2)
{
    Console.WriteLine(val1);
    Console.WriteLine(val2);
}
```

به مثال زیر که در آن دو مقدار مختلف به متد ارسال شده است توجه کنید:

```
Show(5, true);

// OR

Show<int, bool>(5, true);
```

مشاهده می‌کنید که X با نوع int و Y با نوع bool جایگزین می‌شود. این نکته را نیز یادآور شویم که شما می‌توانید دو آرگومان هم نوع را هم به متد ارسال کنید:

```
Show(5, 10);

// OR

Show<int, int>(5, true);
```

کلاس‌های جنریک

تعریف یک کلاس جنریک بسیار شبیه به تعریف یک متد جنریک است. کلاس جنریک دارای یک علامت بزرگ‌تر و کوچک‌تر و یک نوع پارامتر خاص می‌باشد. برنامه زیر مثالی از یک کلاس جنریک می‌باشد:

```
using System;

namespace GenericClassDemo
{
    public class GenericClass<T>
    {
        private T someField;
```

```
        public GenericClass(T someVariable)
        {
            someField = someVariable;
        }

        public T SomeProperty
        {
            get { return someField; }
            set { someField = value; }
        }
    }

    public class Program
    {
        public static void Main()
        {
            GenericClass<double> genericDouble = new GenericClass<double>(30.50);
            GenericClass<int>    genericInt    = new GenericClass<int>(10);

            Console.WriteLine("genericDouble.SomeProperty = {0}", genericDouble.SomeProperty);
            Console.WriteLine("genericInt.SomeProperty = {0}", genericInt.SomeProperty);

            genericDouble.SomeProperty = 100.32;
            genericInt.SomeProperty = 50;
        }
    }
}
```

```
genericDouble.SomeProperty = 30.50
genericInt.SomeProperty = 10
```

در مثال بالا یک کلاس جنریک که دارای یک فیلد، یک خاصیت و یک سازنده است، را ایجاد می‌کنیم. تمام مکآنهایی که ورودی T در آنها قرار دارد، بعداً توسط انواعی که مد نظر شما است، جایگزین می‌شوند. وقتی یک نمونه از کلاس جنریک تان ایجاد می‌کنید، یک نوع هم برای آن در نظر بگیرید (<int>). مانند متدهای جنریک می‌توانید چندین نوع پارامتر به کلاس‌های جنریک اختصاص دهید.

```
public class GenericClass<T1, T2, T3>
{
    private T1 someField1;
    private T2 someField2;
    private T3 someField3;
}
```

چون نمی‌دانید T1، T2 و T3 از چه نوعی هستند نمی‌توانید مانند مثال زیر از آنها نمونه جدید ایجاد کنید.

```
public GenericClass //Constructor
{
    someField1 = new T1();
    someField2 = new T2();
    someField3 = new T3();
}
```

کلاس‌های غیر جنریک می‌توانند از کلاس‌های جنریک ارث بری کنند، اما باید یک نوع برای پارامتر کلاس پایه جنریک تعریف کنید.

```
public class MyClass : GenericClass<int>
{

}
```

یک کلاس جنریک هم می‌تواند از یک کلاس غیر جنریک ارث بری کند.

محدودیت نوع

کد جنریک باید برای هر نوع داده‌ای کار کند. یک عمل محاسباتی مانند عمل جمع که بر روی انواع صحیح انجام می‌شود نمی‌تواند در صورتیکه عملگر + سربارگذاری نشده باشد بر روی سایر اشیاء عمل کند. در نتیجه شما باید برای یک متد یا کلاس جنریک محدودیت نوع در نظر بگیرید که فقط انواع خاصی در این لیست محدودیت قرار بگیرند.

```
public class GenericClass<T> where T : int
{
    //some code
}
```

برای ایجاد این محدودیت ابتدا کلمه کلیدی where سپس نام نوع پارامتر، یک کالن (:) و در آخر اگر پارامتر T دارای لیستی از محدودیت‌ها است، باید آنها را به وسیله کاما از هم جدا کنیم. کد بالا دارای یک محدودیت است و آن نوع int می‌باشد، بدین معنی که کلاس جنریک ما فقط می‌تواند این نوع را قبول کند. اگر بخواهید چندین محدودیت ایجاد کنید باید به صورت زیر عمل نمایید:

```
public class GenericClass<T> where T : int, string
{
    //some code
}
```

پارامتر T در کد بالا فقط نوع int و string را قبول می‌کند. می‌توانید از کلمه کلیدی struct استفاده کنید بدین معنی که کلاس جنریک فقط انواع مقداری را قبول کند و یا کلمه کلیدی Class را به کار برید که کلاس جنریک فقط انواع مرجع را پذیرا باشد. همچنین می‌توان از کلمه کلیدی interface هم استفاده کرد، که در نتیجه کلاس می‌تواند اشیایی که رابط‌ها را پیاده سازی می‌کنند را قبول کند.

```
public class GenericClass<T> where T: struct
{
    //some code
}

public class GenericClass<T> where T: class
{
    //some code
}

public class GenericClass<T> where T: interface
{
    //some code
}
```

همچنین می‌توان نام یک کلاس را به عنوان محدودیت نوع تعیین کرد. بدین معنی که نوع را محدود به آن کلاس و کلاس‌های مشتق شده از آن کند. می‌توان تعیین کرد که کلاس جنریک فقط کلاس‌هایی که دارای یک سازنده بدون پارامتر هستند را قبول کند. برای ایجاد این محدودیت باید از new() به صورت زیر استفاده شود:

```
public class GenericClass<T> where T : new()
{
    //some code
}
```

در این مورد اگر چندین محدودیت وجود داشته باشد باید ()new را در آخر آنها قرار دهیم. اگر کلاس دارای چندین پارامتر باشد و شما بخواهید برای هر یک از آنها محدودیت‌های مختلفی ایجاد کند می‌توانید دستور where را به صورت زیر اعمال نمایید:

```
public class GenericClass<T1, T2>
    where T1 : int
    where T2 : string
{
    //some code
}
```

اگر یک کلاس جنریک از یک کلاس دیگر ارث بری کند باید نام آن کلاس را قبل از محدودیت‌ها ذکر نمایید.

```
public class GenericClass<T1, T2> : BaseClass
    where T1 : int
    where T2 : string
{
    //some code
}
```

انواع تهی

می‌توانید انواع ساده‌ای مانند int و double ایجاد کنید که مقادیر آنها تهی (null) باشد. مقادیر تهی فقط قابلیت ذخیره سازی انواع مرجع مانند رشته‌ها و سایر اشیاء را دارند. سی‌شارپ به شما اجازه می‌دهد انواع مقداری را تغییر دهید به طوریکه بتوانند به عنوان انواع تهی به کار روند. می‌توانید از System.Nullable<T> استفاده کنید که در آن T نوعی است که به انواع تهی تبدیل می‌شود.

```
Nullable<int>   nullInt    = null;
Nullable<double> nullDouble = null;
```

همچنین می‌توانید به نوع علامت ? را نیز اضافه کنید.

```
int? nullInt = null;
double? nullDouble = null;
```

با استفاده از کد زیر می‌توانید تست کنید که آیا متغیر دارای مقدار تهی می‌باشد یا نه؟

```
if (nullInt == null)
{
}

if (nullDouble.HasValue)
{
}
```

از آنجاییکه ما آنها را به انواع تهی تبدیل کرده‌ایم نمی‌توان آنها را در یک متغیر از انواع غیر تهی ذخیره نمود. مثلاً کد زیر مجاز نیست:

```
int? nullInt = null;
int myNumber = nullInt;
```

برای این کار لازم است که ابتدا آنها را به حالت اصلی برگردانید:

```
int myNumber = (int)nullInt;
```

اگر بخواهید یک نوع تهی با یک مقدار تهی را به حالت اولیه تبدیل کنید یک استثناء روی می‌دهد. وقتی دو نوع تهی (به استثنای نوع بولی؟) را در یک عملیات در گیر می‌کنیم، اگر یکی از عملوندها تهی باشد نتیجه تهی خواهد بود. نتایج ممکن برای نوع بولی؟ در جدول زیر آمده است:

var1	var2	var1 & var2	var1 | var2
true	null	null	true
false	null	false	null
null	true	null	true
null	false	false	null
null	null	null	null

اگر شما بخواهید از null شدن نتیجه یک عبارت در صورتی که یکی از عملوندهای آن عبارت null باشد جلوگیری کنید باید از عملگر ?? استفاده کنید.

```
int? nullInt = null;
int number = nullInt * 5 ?? 10;
```

خط دوم نحوه‌ی استفاده از این عملگر را توضیح می‌دهد. برای نوشتن عبارتی معادل کد بالا، از یک عملگر سه تایی به صورت زیر استفاده می‌کنیم.

```
int number = (nullInt * 5) == null ? (nullInt * 5) : 10;
```

اگر نتیجه عبارت در سمت چپ عملگر ?? برابر null باشد نتیجه عبارت سمت راست عملگر و در غیر این صورت نتیجه مقدار سمت چپ عملگر در متغیر number ذخیره می‌شود.

عملگر Null Coalescing (??)

عملگر Null Coalescing یک عملگر باینری است که برای تشخیص مقدار دو عملوند به کار می‌رود. کاربرد اصلی این عملگر در قرار دادن یک مقدار nullable در یک مقدار non-nullable با استفاده از یک دستورالعمل ساده است. در زیر نحوه استفاده از عملگر Null Coalescing نشان داده شده است:

```
var result = operand1 ?? operand2;
```

operand1 و operand2 می‌توانند متغیر، فراخوانی یک متد و یا یک عبارت باشند. اگر operand1 یک متغیر با مقدار تهی باشد و یا فراخوانی یک متد و یا عبارتی باشد که یک مقدار تهی را تولید کند در این صورت عملگر null coalescing به سراغ مقدار عملوند operand2 می‌رود. اگر مقدار این عملوند غیر تهی باشد عملگر null coalescing از آن در نتیجه محاسبه استفاده می‌کند. به مثال زیر توجه کنید:

```
public class Program
{
    public static void Main(string[] args)
    {
        int? num1 = null;
        int? num2 = 100;
```

```
        int? num3 = num1 ?? num2;

        Console.WriteLine($"Value is {num3}");
    }
}
```

```
alue is 100
```

در خط ۸ مثال بالا از عملگر null coalescing استفاده کرده‌ایم. از آنجاییکه مقدار اولیه num1 برابر null و مقدار num2 یک مقدار غیر تهی است، در نتیجه مقدار num2 در متغیر num3 قرار می‌گیرد. در مثال زیر هم یک متد را فراخوانی می‌کنیم. البته به طور قطع نمی‌دانیم که آیا با فراخوانی آن یک مقدار تهی برگشت داده می‌شود و یا یک مقدار غیر تهی:

```
using System;

namespace NullCoalescingOperator
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Random randomizer = new Random();

            int? result = GetNullOrIntValue(randomizer.Next(1, 10)) ?? default(int);

            Console.WriteLine($"Result is {result}");
        }

        public static int? GetNullOrIntValue(int randomValue)
        {
            if (randomValue % 2 == 0)
            {
                return null;
            }

            return 100;
        }
    }
}
```

```
Result is 100
Result is 0
```

در مثال بالا یک متد به نام GetNullOrIntValue() تعریف کرده‌ایم که یک مقدار تصادفی به عنوان آرگومان قبول می‌کند. این متد در صورتی که یک مقدار زوج به آن ارسال شود مقدار null و در غیر اینصورت مقدار ۱۰۰ را بر می‌گرداند. وقتی که در خط ۱۱ این متد را فراخوانی می‌کنیم، یک مقدار تصافی به عنوان آرگومان به آن ارسال می‌شود. عملوند دومی که برای عملگر null coalescing استفاده می‌کنیم کلمه کلیدی defaultاست که برای تشخیص مقدار پیش‌فرض نوع int یعنی عدد صفر به کار برده شده است. اگر با فراخوانی متد به طور تصادفی یک مقدار زوج به آن ارسال شود، متد مقدار پیش‌فرض نوع int و در غیر اینصورت یک مقدار غیر تهی (یعنی ۱۰۰) را بر می‌گرداند.

رویدادها (Events)

رویدادها، رفتارها یا اتفاقاتی هستند که در هنگام اجرای برنامه روی می‌دهند. رویدادها معمولاً در برنامه‌های ویژوال مانند ویندوزها یا صفحات وب استفاده می‌شوند. برخی از رویدادها عبارت‌اند از کلیک کردن بر روی ماوس، تایپ یک متن در TextBox، تغییر انتخاب یک آیتم در یک لیست

و غیره.... در برنامه‌های کنسول، می‌توانیم به صورت دستی یک رویداد را راه اندازی کنیم بدین صورت که یک بلوک کد اضافه می‌کنیم که وقتی یک حالت خاص اتفاق افتاد اجرا شود. چندین کنترل کننده رویداد می‌توانند به یک رویداد متصل شوند. یک کنترل کننده رویداد متدی است که امضای آن شبیه به امضای delegate مربوط به رویداد است زیرا هر رویداد به طور ضمنی شامل یک delegate در داخل خود است. زمانی که شما یک کنترل کننده رویداد را به یک رویداد متصل می‌کنید در واقع شما کنترل کننده یا همان متد را در داخل delegate رویداد قرار می‌دهید. در نتیجه وقتی یک رویداد آزاد می‌شود همه کنترل کننده‌های رویداد متصل به آن اجرا می‌شوند. به طور کلی می‌توان گفت که، هر رویداد دارای یک delegate داخلی مربوط به خود است و در نتیجه وقتی که رویداد به وقوع می‌پیوندد، متدهای داخل delegate اجرا می‌شوند. نحوه تعریف یک رویداد به صورت زیر است:

```
accessSpecifier event delegateType EventName;
```

accessSpecifier سطح دسترسی رویداد است و event یک کلمه کلیدی که در تعریف رویداد به کار می‌رود. delegateType نوع delegate ی که مورد استفاده قرار گرفته است را مشخص می‌کند. از delegate برای تشخیص امضای کنترل کننده رویدادی که می‌تواند به آن متصل شود استفاده می‌شود. EventName هم نامی است که برای رویداد در نظر می‌گیریم. می‌توان ارتباط بین delegate و event را به صورت زیر خلاصه کرد:

۱. یک delegate تعریف می‌کنیم.
۲. متدهایی که قرار است هنگام وقوع رویداد اجرا شوند را به delegate اضافه می‌کنیم.
۳. یک رویداد (event) ایجاد می‌کنیم.
٤. یک متد برای اجرای رویداد تعریف می‌کنیم.
٥. آن دسته از متدهای موجود در delegate را که می‌خواهیم هنگام وقوع رویداد اجرا شوند یا نشوند را با استفاده از عملگرهای =+ و =- به رویداد معرفی می‌کنیم.
٦. متدی که باعث وقوع رویداد می‌شود را فراخوانی می‌کنیم.

برای روشن شدن موارد بالا به مثال زیر توجه کنید :

```
1  using System;
2
3  public delegate void MessageHandler();                                    //step 1
4
5  public class Message
6  {
7      public void DisplayMessage()                                          //step 2
8      {
9          Console.WriteLine("Hello World!");
10     }
11
12     public event MessageHandler ShowMessage;                              //step 3
13
14     public void ExecuteEvent()                                            //step 4
15     {
16         ShowMessage();
17     }
18 }
19
```

```
20  public class Program
21  {
22      public static void Main()
23      {
24          Message myMessage = new Message();
25          myMessage.ShowMessage += new MessageHandler(myMessage.DisplayMessage); //step 5
26
27          myMessage.ExecuteEvent();                                               //step 6
28      }
29  }
```

```
Hello World!
```

در کد بالا یک delegate تعریف کرده‌ایم که مقدار برگشتی آن از نوع void بوده و هیچ پارامتری قبول نمی‌کند (خط ۳). این امضاء، همان امضای کنترل کننده رویداد می‌باشد. بعد از تعریف delegate یک کلاس (خطوط ۱۸-۵) ایجاد می‌کنیم. یک کنترل کننده رویداد یا متد به نام DisplayMessage() که امضای آن شبیه امضای delegate است ایجاد می‌کنیم (خطوط ۱۰-۷). این متد در داخل delegate قرار می‌گیرد. در خط ۱۲ یک رویداد و سپس متدی که رویداد را به صورت دستی آزاد می‌کند تعریف می‌کنیم (۱۷-۱۴). رویدادها نمی‌توانند در خارج از کلاسی که در آن قرار دارند، آزاد شوند. بنابراین ما از متد ایجاد شده برای اجرای غیر مستقیم آنها استفاده می‌کنیم. در داخل متد Main() یک نمونه جدید از کلاس Message ایجاد می‌کنیم (خط ۲۴). در خط بعد یک کنترل کننده رویداد به رویداد ما متصل می‌شود (خط ۲۵). همانطور که در خط ۲۵ مشاهده می‌کنید، برای اینکه به برنامه بفهمانیم که می‌خواهیم هنگام وقوع رویداد چه متدی از delegate اجرا شود باید به صورت زیر عمل کنیم:

```
EventName += new DelegateType(MethodName);
```

به این نکته توجه کنید که استفاده از عملگر += بدین معنی است که می‌خواهیم یک کنترل کننده رویداد به لیست کنترل کننده‌های رویداد اضافه نماییم. یک نمونه از نماینده MessageHandler (delegate) ایجاد کرده و در داخل آن نام کنترل کننده رویداد را ارسال می‌کنیم (خط ۲۵). وقتی که متد ExecuteEvent() فراخوانی می‌شود، رویداد آزاد و پیغام نمایش داده می‌شود (خط ۲۷). در آینده به مفید بودن رویداد در هنگام کار با فرم‌های ویندوزی و صفحات وب پی خواهید برد.

متدهای بی نام(Anonymous Methods)

متدهای بی نام متدهایی هستند که در واقع تعریف نمی‌شوند، بنابراین فقط برای یکبار (one-time) مورد استفاده قرار می‌گیرند. این متدها هدفی برای delegate ها هستند. در زیر نحوه استفاده از متدهای بی نام نشان داده شده است:

```
delegate (parameters)
{
    //Code for the anonymous method
};
```

به عنوان مثال می‌توانیم یک delegate تعریف کرده و سپس یک شیء از آن ایجاد کنیم و متد بی نام را به آن اختصاص دهیم:

```
using System;

public delegate void MessageDelegate(string message);
```

```
public class Program
{
    public static void Main()
    {
        MessageDelegate ShowMessage = new MessageDelegate(
        delegate(string message)
        {
            Console.WriteLine(message);
        }
        );

        ShowMessage("Hello World!");
    }
}
```

در مثال فوق delegate دارای نوع برگشتی void و یک پارامتر از نوع رشته می‌باشد. وقتی یک شیء از delegate ایجاد کردیم آنگاه متد بی نام را به آن ارسال می‌کنیم. به این نکته توجه کنید که متد بی نام در مثال بالا دارای یک پارامتر از نوع رشته مانند delegate است. نوع برگشتی به صورت خودکار تشخیص داده می‌شود. و اگر نوع برگشتی تشخیص داده نشود متد نوع void را بر می‌گرداند. اگر deldegate ایجاد شده توسط شما دارای نوع برگشتی باشد، متد بی نامی هم که ایجاد می‌کنید باید دارای دستور return ی باشد که مقدار یک نوع مناسب را برگشت دهد. می‌توانید کد بالا را به صورت ساده تری بنویسید:

```
MessageDelegate ShowMessage = delegate(string message)
                              {
                                  Console.WriteLine(message);
                              };
```

متدهای بی نام می‌توانند هنگام وقوع رویدادها هم به کار روند:

```
myClass.myEvent += delegate(string message)
                   {
                       Console.WriteLine(message);
                   };
```

مقدار دهنده‌ها (Initializers)

Initializers به شما اجازه می‌دهند خاصیت‌ها را در داخل کلاس مقداردهی کنید. اگر به عنوان مثال چندین خاصیت داشته باشید و نخواهید که یک سازنده را جهت مقداردهی به آنها تعریف کنید می‌توانید از object initializer استفاده نمایید. به عنوان مثال به کد زیر توجه کنید:

```
using System;

public class Sample
{
    public int    Property1 { get; set; }
    public string Property2 { get; set; }
    public bool   Property3 { get; set; }
}

public class Program
{
    public static void Main()
    {
        Sample sampleClass = new Sample();
```

```
        sampleClass.Property1 = 100;
        sampleClass.Property2 = "Sample";
        sampleClass.Property3 = true;
    }
}
```

همانطور که مشاهده می‌کنید، لازم است که مقادیر را تک به تک به خاصیت‌ها اختصاص دهیم. با استفاده از object initializers می‌توان کد را ساده تر کرد:

```
public class Program
{
    public static void Main()
    {
        Sample sampleClass = new Sample
        {
            Property1 = 100,
            Property2 = "Hello",
            Property3 = true
        };
    }
}
```

مشاهده می‌کنید که بعد از ایجاد یک شیء از کلاس به جای پرانتز از آکولاد استفاده کرده و سپس با لیست کردن خاصیت‌ها مقادیری را که لازم داریم، به آنها اختصاص می‌دهیم. به این نکته نیز توجه کنید که، خواص به وسیله کاما از هم جدا می‌شوند. هنگام استفاده از object initializers سازنده پیش‌فرض بدون پارامتر قبل از هر خاصیت مقداردهی شده فراخوانی می‌شود. از آنجاییکه سازنده پیش‌فرض قبل از اختصاص مقادیر به خاصیت‌ها اجرا می‌شود، می‌توانید مقادیری پیشفرضی به هر یک از خواص اختصاص بدهید، با این کار لازم نیست که حتماً به همه خواص با استفاده از initializer مقدار اختصاص داده شود. اگر یک سازنده غیر پیش‌فرض (non-default constructor) به کلاس اضافه کنید، باز هم باید یک سازنده بدون پارامتر پیش‌فرض برای امکان استفاده از object initializers وجود داشته باشد. می‌توان از object initializers تو در تو نیز استفاده نمود. فرض کنید کلاس Sample مان یک خاصیت از نوع Animal که دارای دو خاصیت Name و Age هست را دارا می‌باشد.

```
Sample sampleClass = new Sample
{
    Property1 = 100,
    Property2 = "Hello",
    Property3 = true,
    Property4 = new Animal { Name = "Kitty", Age = 3 };
};
```

نوع دیگر از مقداردهنده ها، collection initializers می‌باشند. collection initializers بسیار شبیه به array initializers می‌باشند با این تفاوت که در کلکسیون‌های عمومی (generic) استفاده می‌شوند.

```
List<Person> people = new List<Person>
{
     new Person("John"),
     new Person("Jenny"),
     new Person("Joe")
};
```

کد زیر نشان می‌دهد که بدون استفاده از initializers collecition، لازم است که به صورت دستی هر آیتم را با استفاده از متد Add() اضافه کرده یا یک سازنده ایجاد کنید که آیتم‌هایی برای کلکسیون قبول می‌کند.

```
List<Person> people = new List<Person>();

people.Add(new Person("John"));
people.Add(new Person("Jenny"));
people.Add(new Person("Joe"));
```

نوع استنباطی (Type Inference)

نوع استنباطی به متغیر اجازه می‌دهد که حدس بزند چه نوع داده‌ای به آن اختصاص داده شده است. برای ایجاد انواع استنباطی در سی‌شارپ از کلمه کلیدی var استفاده می‌شود.

```
var myInt    = 10;
var myDouble = 5.67;
var myString = "Hello";
```

سه متغیر بالا انواع ضمنی هستند بدین معنی که نوع آنها به بسته به مقادیری که به آنها اختصاص داده می‌شود به صورت اتوماتیک تغییر می‌کند (مشخص می‌شود). برای مشخص کردن نوع یک متغیر می‌توان به سادگی از کلمه کلیدی var استفاده کرد. حتی می‌توان برای ذخیره نوع object هم از این کلمه کلیدی استفاده نمود.

```
var sample = new SampleClass();
```

این کلمه کلیدی را می‌تون برای آرایه‌ها نیز به کار برد:

```
var myArray = new int [] { 1, 2, 3 };

//OR

var myArray = new [] { 1, 2, 3 };
```

به این نکته توجه کنید که شما می‌توانید بدون ذکر نوع یک متغیر به صورت صریح (مثلاً int myInt)، از کلمه کلیدی var استفاده نمایید، که در نتیجه این کار، نوع متغیر بسته به نوع مقداری که به آن اختصاص داده می‌شود تعیین گردد. از آنجاییکه نوع متغیر بسته به نوع مقداری که به آن اختصاص داده می‌شود تعیین می‌شود، نمی‌توان از متغیری که به آن هیچ مقداری نداده‌ایم همراه با کلمه کلیدی var استفاده کنیم. مثلاً کد زیر کامپایل نمی‌شود چون هیچ مقداری به متغیر اختصاص داده نشده است.

```
var someVariable;
```

به عنوان آخرین نکته، کلمه کلیدی var برای تعیین نوع (متغیرهای محلی) به کار می‌رود و نمی‌توان از آن به جای نوع برگشتی و نوع پارامترهای یک متد استفاده نمود.

انواع بی نام(Anonymous Types)

در سی‌شارپ می‌توان انواع بی نامی تعریف کرد که یک روش عالی برای تعریف انواع موقتی جهت ذخیره انواع داده‌ها می‌باشد. فرض کنید که یک کلاس می‌خواهید که سه مقدار را در داخل property هایش جای دهد.

```
public class Sample
{
    public int    Property1 { get; set; }
    public double Property2 { get; set; }
    public string Property3 { get; set; }
}
```

لازم نیست مانند مثال بالا یک کلاس جهت ذخیره سازی ایجاد کنید بلکه می‌توانید یک نوع بی نام ایجاد کنید که property هایی مانند مثال بالا را دارا باشد.

```
var anonymousType = new
    { Property1 = 10,
      Property2 = 5.35,
      Property3 = "Hello" };
```

برای این کار باید از کلمه کلیدی var استفاده شود. دستور ایجاد یک نوع بی نام شبیه به object initializers است با این تفاوت که در آن از نام کلاس استفاده نمی‌کنیم. بعد از تعریف می‌توان از خواص انواع بی نام استفاده کرد.

```
Console.WriteLine(anonymousType.Property1);
Console.WriteLine(anonymousType.Property2);
Console.WriteLine(anonymousType.Property3);
```

به این نکته توجه کنید که مقدار دهی فقط یکبار انجام می‌شود و نمی‌توانید مقادیر خواص یک نوع بی نام را ویرایش کنید چون این خواص، خواص فقط-خواندنی هستند.

متدهای توسعه یافته

همه متدها وابسته به کلاسی هستند که در آن تعریف شده‌اند. اما ویژگی توسعه متدها به شما اجازه می‌دهد که متدی ایجاد کنید که علاوه بر کلاسی که در آن تعریف شده است، به کلاس‌های دیگر نیز وابسته باشد. می‌خواهیم نحوه استفاده از این ویژگی را به شما آموزش دهیم. به کد زیر توجه کنید. این کد شامل کلاسی به نام Class1 (خطوط ۱-۱۵) است که سه مقدار از نوع double (خط ۳) را در خود ذخیره می‌کند، همچنین شامل یک سازنده (۵-۱۰) و یک متد به نام Sum() (خطوط ۱۱-۱۴) است که جمع سه مقدار ذخیره شده را برگشت می‌دهد.

```
1   sealed class Class1
2   {
3       private double D1, D2, D3;
4
5       public Class1(double d1, double d2, double d3)
6       {
7           D1 = d1;
8           D2 = d2;
9           D3 = d3;
10      }
11      public double Sum()
12      {
13          return D1 + D2 + D3;
14      }
15  }
```

محدودیت این کلاس تا حدی خوب است، اما فرض کنید بخواهید از این کلاس بهتر استفاده کنید و متدی دیگری ایجاد کنید که میانگین سه مقدار ذکر شده را نیز برگرداند. با چیزهایی که در مورد کلاس‌ها یاد گرفته‌اید، چندین روش برای اجرای این کار (اضافه کردن متد) وجود دارد. اگر سورس اصلی کلاس را در اختیار داشته باشید و بتوانید آن را دستکاری کنید به راحتی می‌توانید متد جدیدی به آن اضافه نمایید. اگر به هر دلیلی نتوانید کلاس را دستکاری کنید، به عنوان مثال کلاس sealed باشد، مجبور می‌شوید که یک متد در کلاس دیگر تعریف کنید و آن را به صورت عمومی در دسترس اعضای کلاس قرار دهید. به عنوان مثال یک کلاس با نام ExtendClass1 باسطح دسترسی static می‌نویسیم که دارای متدی به نام Average() و سطح دسترسی static است. به این نکته توجه کنید که متد Average() یک نمونه از کلاس Class1 را به عنوان پارامتر می‌گیرد.

```
static class ExtendClass1
{
    public static double Average(Class1 C1)
    {
        return C1.Sum() / 3;
    }
}

class Program
{
    static void Main()
    {
        Class1 C1 = new Class1(3, 4, 5);
        Console.WriteLine("Average: {0}", ExtendClass1.Average(C1));
    }
}
```

```
4
```

با وجودیکه این راه حل کاملاً درست است، اما راه حل بهتر این است که متد را در نمونه کلاس خودش فراخوانی کنید، تا اینکه یک نمونه از کلاس دیگر ایجاد کرده و متد را فراخوانی کنید. در دو خط کد زیر تفاوت روشن می‌شود. در خط اول متد استاتیک به وسیله یک نمونه از کلاس دیگر و در خط دوم متد به وسیله یک شیء ایجاد شده از کلاس خودش فراخوانی شده است.

```
ExtendClass1.Average(C1);

C1.Average();
```

توسعه متدها به شما اجازه استفاده از روش دوم را می‌دهد، هر چند که روش اول روشی عادی برای فراخوانی می‌باشد. با اندکی تغییر در تعریف متد Average() از روش دوم استفاده کنید. تغییری که لازم است در متد بدهید، اضافه کردن کلمه کلیدی this قبل از نام نوع در تعریف پارامترها می‌باشد، همانطور که در زیر نشان داده شده است. اضافه کردن کلمه کلیدی this به اولین پارامتر متد static کلاس static باعث تبدیل متد عادی کلاس ExtendClass1 به متد توسعه یافته کلاس Class1 می‌شود. حال می‌توان از هر دو حالت فراخوانی استفاده نمود.

```
static class ExtendClass1
{
    public static double Average(this Class1 C1)
     {
         .....
     }
}
```

نکات مهم در باره متد توسعه یافته در زیر ذکر شده است:

- کلاسی که متد توسعه یافته در آن تعریف شده است باید به صورت static تعریف شود.
- خود متد باید به صورت static تعریف شود.
- اولین پارامتر متد توسعه یافته باید با کلمه کلیدی this شروع و بعد از این کلمه نام کلاسی که می‌خواهیم متد به آن اضافه شود، ذکر شود.

کد زیر همه برنامه را که شامل کلاس Class1 و متد توسعه یافته Average() تعریف شده در کلاس ExtendClass1 را نشان می‌دهد. به این نکته توجه کنید که که متد Average() دقیقاً طوری فراخوانی شده است که انگار یک عضو نمونه از کلاس Class1 است. کلاس‌های Class1 و ExtendClass1 هر دو با هم مانند یک کلاس با سه متد عمل می‌کنند.

```
using System;

namespace ExtensionMethods
{

    sealed class Class1
    {
        private double D1, D2, D3;

        public Class1(double d1, double d2, double d3)
        {
            D1 = d1;
            D2 = d2;
            D3 = d3;
        }
        public double Sum()
        {
            return D1 + D2 + D3;
        }
    }

    static class ExtendClass1
    {
        public static double Average(this Class1 C1)
        {
            return C1.Sum() / 3;
        }
    }

    class Program
    {
        static void Main()
        {
            Class1 C1 = new Class1(3, 4, 5);
            Console.WriteLine("Sum: {0}", C1.Sum());
            Console.WriteLine("Average: {0}", C1.Average());
        }
    }
}
```

```
Sum: 12
Average: 4
```

عبارات لامبدا(Lambda expressions)

عبارات لامبدا (Lambda expressions) ساده شده دستور زبان متدهای بی نام هستند. به عنوان مثال در برنامه زیر از یک متد بی نام که به یک delegate ارجاع داده شده است، استفاده شده است.

```
using System;

public delegate void MessageDelegate(string message);

public class Program
{
    public static void Main()
    {
        MessageDelegate ShowMessage = new MessageDelegate(
        delegate(string message)
        {
            Console.WriteLine(message);
        }
        );

        ShowMessage("Hello World!");
    }
}
```

به وسیله عبارات لامبدا می‌توان کد بالا را به صورت ساده تری نوشت:

```
using System;

public delegate void MessageDelegate(string message);

public class Program
{
    public static void Main()
    {
        MessageDelegate ShowMessage = (message) => Console.WriteLine(message);

        ShowMessage("Hello World!");
    }
}
```

مقایسه متد بی نام و عبارت لامبدا:

```
MyDel del = delegate(int x)    { return x + 1; } ;      // متد بی نام
MyDel le1 =         (    x) => { return x + 1; } ;      //عبارت لامبدا
```

در عبارت لامبدا، ابتدا پارامترها را بدون ذکر نوعشان می‌نویسیم و بعد از عملگر => استفاده می‌کنیم. سپس دستوراتی را که قرار است اجرا شوند را می‌نویسیم. نوع پارامترها به صورت خودکار به وسیله کامپایلر تشخیص داده می‌شود. البته امضاء عبارات لامبدا باید شبیه به امضاء delegate باشد. عبارات لامبدا دارای اشکال زیادی هستند. به عنوان مثال، در مثال بالا، از یک عبارت لامبدایی استفاده کرده‌ایم که دارای یک دستور ساده اجرایی است. اگر delegate شما برای عبارت لامبدا هیچ پارامتری نداشته باشد، همه کاری که لازم است انجام دهید این است که، هیچ پارامتری در داخل عبارت لامبدا قرار ندهید.

```
MessageDelegate ShowMessage = () => Console.WriteLine("Hello");
```

به این نکته توجه کنید که شما می‌توانید نوع پارامترهای عبارت لامبدا را نشان دهید.

```
MessageDelegate ShowMessage = (string message) => Console.WriteLine(message);
```

اگر نوع یک پارامتر را مشخص کنید، باید نوع سایر پارامترها را نیز مشخص کنید. به عنوان مثال نمی‌توانید به صورت زیر عمل کنید:

```
MessageDelegate ShowMessage = (string message1, message2) => Console.WriteLine(message1);
```

اگر عبارات لامبدای شما دارای چندین دستور اجرایی باشند، می‌توانید آنها را داخل آکولاد قرار دهید. به عبارت لامبدایی که دارای آکولاد باشد دستور لامبدا می‌گویند.

```
MessageDelegate ShowMessage = (message) =>
{
        Console.WriteLine(message);
        Console.WriteLine("Some more message");
}
```

در زیر مثالی از یک عبارت لامبدا که دارای مقدار برگشتی است نشان داده شده است:

```
SampleDelegate GetSquare = (number) => { return number * number; };
```

به این نکته توجه کنید که هنگام استفاده از دستور `return` باید همیشه از دستورات لامبدایی استفاده کنید که دارای آکولاد می‌باشند. اگر یک عبارت لامبدا فقط دارای یک دستور `return` ساده باشد می‌توانید به سادگی آن را به `expression lambda` تبدیل کنید.

```
SampleDelegate GetSquare = (number) => (number * number);
```

به این نکته توجه کنید که استفاده از پرانتز در کد بالا برای فهم بهتر آن است. اگر یک عبارت لامبدا دارای یک پارامتر ساده و یک دستور `return` است، می‌توانید برای سادگی بیشتر پرانتزها را حذف نمایید:

```
SampleDelegate GetSquare = number => number * number;
```

ولی اگر یک عبارت لامبدا دارای ۲ یا تعداد بیشتری پارامتر باشد باید انها را داخل پرانتز قرار دهید.

```
SampleDelegate GetSum = (num1, num2) => num1 + num2;
```

عبارت لامبدای بالا دارای دو پارامتر است و حاصل جمع آنها را بر می‌گرداند.

یک مثال از عبارت لامبدا

با استفاده از عبارات لامبدا راحت تر می‌توان متدهای بی نام را به عنوان آرگومان به دیگر متدها ارسال کرد. به مثال زیر توجه کنید:

```
1  using System;
2
3  namespace LambdaDemo
4  {
5      public delegate int Arithmetic(int x, int y);
6
7      public class Program
8      {
```

```
9           public static int GetResult(Arithmetic Operation, int num1, int num2)
10          {
11              return Operation(num1, num2);
12          }
13
14          public static void Main()
15          {
16              Console.Write("Enter first number: ");
17              int num1 = Convert.ToInt32(Console.ReadLine());
18              Console.Write("Enter second number: ");
19              int num2 = Convert.ToInt32(Console.ReadLine());
20
21              Console.WriteLine("Sum = "        + GetResult((x, y) => x + y, num1, num2));
22              Console.WriteLine("Difference = " + GetResult((x, y) => x - y, num1, num2));
23              Console.WriteLine("Product = "    + GetResult((x, y) => x * y, num1, num2));
24              Console.WriteLine("Quotient = "   + GetResult((x, y) => x / y, num1, num2));
25          }
26      }
27  }
```

```
Enter first number: 5
Enter second number: 2
Sum = 7
Difference = 3
Product = 10
Quotient = 2
```

ابتدا در خط ۵ یک delegate ایجاد می‌کنیم که دو آرگومان از نوع اعداد صحیح را قبول کرده و یک مقدار صحیح را بر می‌گرداند. سپس از این delegate به عنوان یکی از پارامترهای متد GetResult() استفاده می‌کنیم (خطوط ۱۲-۹). دو پارامتر دیگر، دو عدد هستند که همانطور که خواهیم دید، در عملیات‌های مختلف نقش دارند. از آنجاییکه اولین پارامتر یک delegate است، آرگومانی که به آن ارسال می‌شود باید مرجع یک متد، یا یک متد بی نام و یا یک عبارت لامبدا باشد. با استفاده از عبارت لامبدا می‌توان کدهای خواناتری نوشت و همچنین کدنویسی را کاهش داد. در خطوط ۲۸-۲۱ متد GetResult() را فراخوانی می‌کنیم. توجه کنید که متد در هر فراخوانی یک عبارت لامبدا را به عنوان اولین آرگومان قبول می‌کند. اگر از متدهای بی نام استفاده می‌کردیم، نیاز به کدنویسی بیشتری داشتیم و خوانایی آن نیز کمتر می‌شد.

Expression-Bodied Members

Expression-bodied members یکی از ویژگی‌های C# 6.0 بوده که به شما اجازه استفاده از لامبدا برای کدنویسی راحت تر متدها، خاصیت‌ها، سربارگذاری عملگرها و ایندکسرهای یک کلاس را می‌دهد. با وجود این ویژگی شما به جای نوشتن بدنه یک عضو می‌توانید از علامت لامبدا استفاده کنید. به عنوان مثال، متد زیر را در نظر بگیرید که دو پارامتر را دریافت کرده و جمع آنها را بر می‌گرداند:

```
public int GetSum(int x, int y)
{
    return x + y;
}
```

با استفاده از ویژگی Expression-bodied members شما می‌توانید کد بالا را خلاصه تر کرده و به صورت زیر بنویسید:

```
public int GetSum(int x, int y) => x + y;
```

همانطور که در کد بالا مشاهده می‌کنید امضای متد را دستکاری نمی‌کنیم اما بدنه را با علامت => شروع و سپس عبارتی که جمع دو مقدار را بر می‌گرداند را می‌نویسیم. به این نکته توجه کنید که دیگر لازم نیست از کلمه کلیدی return استفاده کنیم. برای تعریف خاصیت‌ها هم می‌توانیم از این ویژگی استفاده کنیم. مثلاً در کد زیر یک خاصیت فقط خواندنی که نام کامل یک شخص را بر می‌گرداند تعریف کرده‌ایم:

```
public string FullName
{
    get
    {
        return FirstName + " " + LastName;
    }
}
```

کد بالا را به صورت زیر هم می‌توان نوشت:

```
public string FullName => FirstName + " " + LastName;
```

برای متدهایی هم که مقدار برگشتی آنها از نوع viod است و دارای یک خط در بدنه هستند هم می‌توان از این ویژگی به صورت زیر استفاده کرد:

```
public void PrintMessage(string message) => Console.WriteLine(message);
```

Expression-bodied members در تعریف ایندکسرها و سربارگذاری عملگرها هم به کار می‌رود:

```
// Indexer
public int this[int index] => InternalCollection[index];

// Operator Overload
public static Point operator +(Point p1, Point p2) => new Point(p1.X + p2.X, p1.Y + p2.Y);

// Conversion Overload
public static implicit operator string(Point point) => $"({point.X}, {point.Y})";
```

ممکن است که این سؤال برایتان پیش بیاید که آیا می‌توان از ویژگی در اعضایی که دارای چندین دستور در بدنه خود هستند، استفاده کرد. جواب منفی است. این ویژگی فقط برای اعضایی به کار می‌رود که دارای یک دستور ساده در بدنه خود هستند.

استفاده از کلاس های استاتیک در فضای نام

استفاده از کلاس های استاتیک در فضای نام یکی از ویژگی هایی است که در C# 6.0 گنجانده شده است و به ما اجازه می دهد که از کلاس های استاتیک در قسمت فضاهای نامی استفاده کنیم. این ویژگی برای توسعه دهندگان بسیار مفید است و از تکرار مکررات جلوگیری می کند. همانطور که می دانید در C# 5.0 برای استفاده از متدهای استاتیک یک کلاس استاتیک باید ابتدا نام کلاس و سپس نقطه و بعد نام متد را بنویسیم. مانند Convert.ToInt32()، Console.Write() یا Console.WriteLine(). اما در C# 6.0 کافیست که نام کلاس استاتیک را یک بار در قسمت فضای نام وارد کنید و از متدهای آن بدون اینکه نام کلاس را بیاورید استفاده کنید. به مثال زیر توجه کنید:

C# 5.0

```
using System;

namespace CsharpNewFeatures
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to Visual C# Tutorials!");
        }
    }
}
```

C# 6.0

```
using System;

using static System.Console;

namespace CsharpNewFeatures
{
    class Program
    {
        static void Main(string[] args)
        {
            WriteLine("Welcome to Visual C# Tutorials!");
        }
    }
}
```

مقدار دهی اولیه به خصوصیات خودکار

مقدار دهی اولیه به خصوصیات خودکار یا (Auto property initializer) یکی از ویژگی های جدید است که در C# 6.0 معرفی شد که با استفاده از این ویژگی می توان Property ها را در هنگام تعریف مقداردهی کرد .در نسخه های قبلی (C# 5.0) ما Property ها را در سازنده پیشفرض مقدار دهی می کردیم اما در C# 6.0 می توانیم همزمان کار تعریف و مقداردهی را انجام دهیم:

C# 5.0

```
using System;

namespace CsharpNewFeatures
{
    class Program
    {
        public Program()
        {
            Name = "Jack";
            Age = 25;
        }

        public string Name { get; set; }
```

```
        public int Age { get; set; }
    }
}
```

C# 6.0

```
using System;

namespace CsharpNewFeatures
{
    class Program
    {
        public string Name { get; set; } = "Jack";
        public int Age { get; set; } = 25;
    }
}
```

فیلتر استثنائات

فیلتر های استثنائات یا Exception filters یک ویژگی جدید در C# 6.0 است که به ما اجازه ی مشخص کردن یک شرط برای یک بلاک catchرا می دهد. اگر شرط true را برگرداند، آنگاه بلاک catch اجرا میشود. این ویژگی یکی از بهترین ویژگی های C# 6.0 جدید است که کار با فیلتر کردن استثنائات را آسان کرده است. به مثال زیر توجه کنید:

```
using System;

namespace CsharpNewFeatures
{
    class Program
    {
        public static void Main()
        {
            int result;
            int x = 5;
            int y = 0;

            try
            {
                result = x / y;
            }
            catch (Exception ex) if (y == 0)
            {
                Console.WriteLine("An attempt to divide by 0 was detected.");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

```
An attempt to divide by 0 was detected.
```

دستور using

کاربرد اصلی دستور using نابود کردن اشیاء و آزاد کردن فضای حافظه است. همانطور که می دانید بعد از ساخت اشیاء و عدم استفاده از آن باید آن را از حافظه خارج کنیم تا دوباره آن بخش از حافظه مورد استفاده قرار گیرد. اگر چه garbage collectorیا زباله روب حافظه این کار را به طور اتوماتیک انجام می دهد، ولی مشخص نیست garbage collectorچه زمانی رخ می دهد و تا زمانی که garbage collector کار خود را شروع نکند همچنان حافظه بلا استفاده می ماند. در سی شارپ، شما می توانید هر لحظه که لازم باشد، شیء بلا استفاده را از حافظه پاک کنید (اصطلاحاً Dispose کنید). ساختار کلی دستور using به صورت زیر است:

```
using()
{
    ...
}
```

این دستور، به دستور try…finally ترجمه می شود. پس دو کد زیر با هم برابرند:

```
using (Person person = new Person())
{
     // Code to execute
}
Person person = new Person();
try
{
    // Code to execute
}
finally
{
    if (person != null)
    {
        person.Dispose();
    }
}
```

اگر بخواهیم از کلاسمان (Person) در داخل دستور using استفاده کنیم، باید ابتدا رابط IDisposable را توسط کلاسمان پیاده سازی کنیم. این رابط دارای متد ()Dispose می باشد که بعد از دستور using فراخوانی شده و حافظه را از اشیاء بلا استفاده پاک می کند. به تکه کد زیر توجه کنید:

```
using System;

namespace UsingStatement
{
    class Person : IDisposable
    {
        public void Dispose()
        {
            Console.WriteLine("The Dispose method executed!");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            using (Person person = new Person())
            {
```

```
                Console.WriteLine("Using Statement executed!");
            }
        }
    }
}
```

```
Using Statement executed!
The Dispose method executed!
```

همانطور که در خروجی مشاهده می کنید، ابتدا دستور داخل بلاک using و سپس دستور داخل متد Dispose() اجرا می شود.

مخفی کردن متد (Method Hiding)

همانطور که می دانید وقتی یک کلاس از کلاس دیگر ارث بری می کند، به تمام اعضای public آن کلاس دسترسی می یابد. اما در برخی موارد، ممکن است کلاس فرزند دارای متدی باشد که نام و امضای آن شبیه به متدی در کلاس پدر باشد.حال اگر یک شیء از کلاس فرزند ایجاد و متد را فراخوانی کنید، در اصل متد کلاس فرزند فراخوانی می شود، چونکه کامپایلر سی شارپ به طور خودکار متد کلاس پایه را مخفی می کند. هر چند که سی شارپ این کار را خودکار انجام می دهد ولی بهتر است شما این کار را به صورت دستی و با اضافه کردن کلمه کلیدی new قبل از نوع برگشتی متد انجام دهید، به این کار مخفی کردن متد یا Method Hiding می گویند، چون که شما عمدا متد کلاس پایه را مخفی کرده اید:

```
1  using System;
2
3  namespace Program
4  {
5      class Parent
6      {
7          public void ShowMessage()
8          {
9              Console.WriteLine("Method from Base Class");
10         }
11     }
12
13     class Child : Parent
14     {
15         public new void ShowMessage()
16         {
17             Console.WriteLine("Method from Derived Class");
18         }
19     }
20
21     class Program
22     {
23         static void Main(string[] args)
24         {
25             Child child1 = new Child();
26             child1.ShowMessage();
27         }
28     }
29
30 }
```

```
Method from Derived Class
```

ممکن است که این سوال برای شما پیش آمده باشد که اگر بخواهیم متد کلاس پایه را فراخوانی کنیم، باید چکار کنیم؟ سه روش برای این کار وجود دارد .اولین روش استفاده از کلمه کلیدی base است. خط ۱۷ را به صورت زیر تغییر داده و برنامه را اجرا کنید:

```
base.ShowMessage();
```

```
Method from Base Class
```

روش دوم تبدیل صریح شیء ایجاد شده از کلاس مشتق به وسیله کلاس پایه است. خط ۲۶ را به صورت زیر تغییر داده و برنامه را اجرا کنید:

```
((Parent)child1).ShowMessage();
```

```
Method from Base Class
```

روش سوم، دادن ارجاعی از کلاس پایه به شیء کلاس مشتق است. خطوط ۲۶-۲۵ را به صورت زیر تغییر داده و برنامه را اجرا کنید:

```
Parent parent = new Child();
parent.ShowMessage();
```

```
Method from Base Class
```

حال که با بازنویسی (Overriding) و مخفی کردن (Hiding) متدها آشنا شدید، بهتر است که با تفاوت این دو آشنا شوید. در Overriding اگر یک ارجاع از کلاس پایه به کلاس مشتق بدهیم، متد کلاس مشتق ولی در Hiding، متد کلاس پایه فراخوانی می شود.

Tuple چیست

Tuple یک ساختار داده ای است که دارای تعدادی خاص و توالی از عناصر می باشد. این ساختار داده ای به شما این امکان را می دهد که داده هایی با انواع مختلف را بدون اینکه کلاس جداگانه ای تعریف کنید، در آن قرار دهید. پس Tuple از ایجاد کلاس هایی که شما برای بسته بندی و انتقال داده ها بین قسمت های مختلف برنامه تعریف می کنید، جلوگیری می کند. فرض کنید که در یک برنامه نیاز است که ما دو مقدار را به یک متد ارسال کنیم و در آن متد آنها را نمایش دهید.

```
using System;

namespace TupleDemo
{
    public class MethodArguments
    {
        public int ID { get; set; }
        public string Name { get; set; }
    }
    class Program
    {
        public static void Print(MethodArguments args)
        {
            Console.WriteLine("ID = " + args.ID.ToString());
            Console.WriteLine("Name =  " + args.Name);
```

```
        }
        public static void Main(string[] args)
        {
            MethodArguments arguments = new MethodArguments();
            arguments.ID = 20;
            arguments.Name = "John";

            Print(arguments);

            Console.ReadKey();
        }
    }
}
```

روش کلاسیک (بدون استفاده از Tuple) به این صورت است که باید ابتدا یک کلاس در برنامه تعریف کنید که دارای دو خاصیت باشد، سپس آرگومان ورودی متد را از نوع این کلاس تعریف می کنیم. حال یک شیء از کلاس ایجاد کرده و خاصیت های آن را مقداردهی می کنیم. سپس این شیء را به متد ارسال می کنیم. در متد نیز مقادیر را از خاصیت های استخراج می کنیم و آنها را نمایش می دهیم. این روش برای تعداد زیاد پارامتر ها نیز جوابگو است و به هر تعداد پارامتری که نیاز داشتید می توانید در کلاس، خاصیت ایجاد کنید و به شیوه مشابه آنها را مقداردهی و به متد مورد نظر ارسال کنید.

مشکلات این روش این است که شما برای هر متد باید یک کلاس جداگانه تعریف کنید که بسیار وقت گیر و کسل کننده است. Tuple این کار را به صورت خودکار برای شما انجام می دهد. زمانی که شما از Tuple استفاده می کنید، کامپایلر سی شارپ کلاس مورد نظر را برای شما در پشت صحنه ایجاد می کند . مثال بالا را این بار با Tuple پیاده سازی می کنیم:

```
using System;

namespace TupleDemo
{
    class Program
    {
        public static void Print(Tuple<int, string> args)
        {
            Console.WriteLine("ID = " + args.Item1.ToString());
            Console.WriteLine("Name =  " + args.Item2);
        }
        public static void Main(string[] args)
        {
            var arguments = new Tuple<int, string>(20, "John");
            Print(arguments);

            Console.ReadKey();
        }
    }

}
```

در مثال بالا دیگر نیازی به کلاس MethodArguments نداریم. ورودی متد را از نوع Tuple<int, string> تعریف می کنیم. در متد Main() یک شی به نام arguments از نوع Tuple<int, string> تعریف کرده ایم .اینطور در نظر بگیرید که با اینکار کامپایلر در پشت صحنه یک کلاس شبیه کلاس MethodArguments تعریف می کند که دارای دو خاصیت می باشد. نوع آنها به ترتیب int و string بوده و نام آنها به ترتیب Item1 و Item2 است . این نامگذاری به صورت خودکار انجام می شود یعنی اگر ما یک پارامتر دیگر مثلا از نوع bool تعریف کنیم

کامپایلر نام آنرا Item3 می گذارد. در ادامه این شی را به متد Print ارسال می کنیم. دقت کنید که نوع پارامتر Print باید دقیقا شبیه به نوع شی ایجاد شده (در اینجا Tuple<int, string>) باشد. روش دیگر برای ایجاد Tuple استفاده از متد Create() این کلاس می باشد. در این صورت اگر خط زیر را جایگزین خط بالا کنید ، همان نتیجه حاصل می شود:

```
var arguments = Tuple.Create(20, "John");
```

به این نکته توجه کنید که خروجی متد ها می تواند از نوع Tuple باشد:

```
using System;

namespace TupleDemo
{
    class Program
    {
        public static Tuple<string, string> GetPerson()
        {
            return new Tuple<string, string>("John", "Scith");
        }
        public static void Main(string[] args)
        {
            var person = GetPerson();

            Console.WriteLine(person.Item1 + " " + person.Item2);

            Console.ReadKey();
        }
    }
}
```

در مثال بالا یک متد به نام GetPerson() با نوع خروجی Tuple<string, string> تعریف کرده ایم. در بدنه آن یک شی از نوع Tuple<string, string> ایجاد کرده ایم و آنرا برگشت می دهیم. در متد Main() خروجی متد را در متغیر person قرار داده و مقدار خاصیت های آنرا در خط بعدی نمایش می دهیم.

توابع محلی (Local Functions)

Local Function ها امکان تعریف یک متد در داخل متد دیگر را فراهم می کنند. شما می توانید با استفاده از این قابلیت در C# 7.0 یک متد را در داخل یک متد موجود تعریف کنید. به این نکته توجه کنید که، برای استفاده از این قابلیت های جدید باید آخرین نسخه ویژوال استودیو (Visual Studio 2015 Preview 4) را بر روی سیستم نصب کنید:

```
private static void Main(string[] args)
{
    int LocalFunction(int arg)
    {
        return 10 * arg;
    }

    Console.WriteLine(LocalFunction(10));
}
```

در مثال بالا، یک متد به نام LocalFunction را در "داخل" بدنه ی متد Main تعریف کرده ایم، سپس آنرا فراخوانی و خروجی آنرا بر روی صفحه ی نمایش چاپ می کنیم. در متد های محلی می توانید از متغیرهایی که قبل از تعریف متد ، تعریف شده اند استفاده کنید :

```
private static void Main(string[] args)
{
    int localVar = 10;
    int LocalFunction(int arg)
    {
        return localVar * arg;
    }
    Console.WriteLine(LocalFunction(10));

}
```

در مثال بالا، در داخل متد محلی از یک متغیر به نام localVar که قبل از متد تعریف شده است استفاده کرده ایم .

اشیاء تغییر ناپذیر (Immutable Object)

گاهی در برنامه های خود نیاز به ایجاد اشیایی داریم که وضعیت داخلی این اشیاء (مقادیر فیلد ها ، خصیصه ها و ...) بعد از ایجاد شی قابل تغییر نباشد. در چارچوب دات نت، کلاس string به این صورت پیاده سازی شده است. زمانی که شما یک مقدار رشته ای را در یک متغیر از نوع string قرار می دهید و بعدا آن مقدار را تغییر می دهید، کامپایلر C# در پشت صحنه یک شیء با مقدار جدید می سازد و در فیلد مورد نظر قرار می دهد و شیء قبلی را دور می اندازد (توسط آشغال جمع کن پاک می کند):

```
string Name = "Jason"; // First time it creates a new object.
Name = "Jack";         // It create a new object of string type
```

در کد زیر شما یک کلاس با نام Person را مشاهده می کنید که شامل دو فیلد با نام های name و family است. این دو فیلد به صورت readonly و از نوع string تعریف شده اند.فیلد های readonly تنها می توانند در خط تعریف فیلد یا در سازنده کلاس مقدار دهی شوند. سپس برای این دو فیلد، خصوصیات متناظری ایجاد می کنیم. خصوصیت هم فقط از قسمت get تشکیل شده است و نباید قسمت set داشته باشد. زیرا کاربر بعد از ایجاد شیء نباید مقادیر درون شیء را تغییر دهد. مقادیر مورد نظر را از طریق پارامتر های سازنده کلاس می گیریم و در فیلدها قرار می دهیم:

```
public class Person
{
    private readonly string name;
    private readonly string family;

    public string Name
    {
        get
        {
            return this.name;
        }
```

```
    }

    public string Family
    {
        get
        {
            return this.family;
        }
    }

    public Person(string n, string f)
    {
        this.name = n;
        this.family = f;
    }
}
```

نحوه ی استفاده از کلاس فوق به صورت زیر می باشد :

```
Person person1 = new Person("Jason", "Statham");

person1.Name = "Jack";

Console.WriteLine(person1.Name + " " + person1.Family);

Console.ReadKey();
```

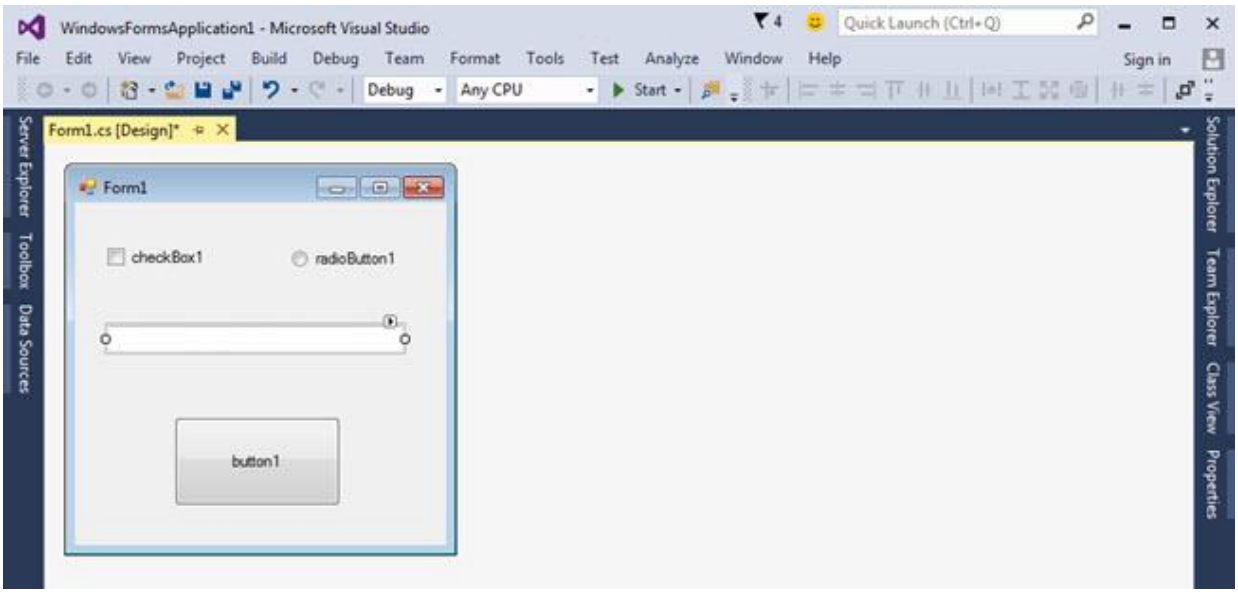
در کد بالا نمی توانیم بعد از ایجاد شیء مقادیر خصوصیات آن را تغییر دهیم. زیرا خصوصیات فقط از قسمت get تشکیل شده اند. پس خط قرمز در مثال بالا غیر مجاز می باشد.

فصل دوم

ویندوز فرم

برنامه نویسی ویژوال

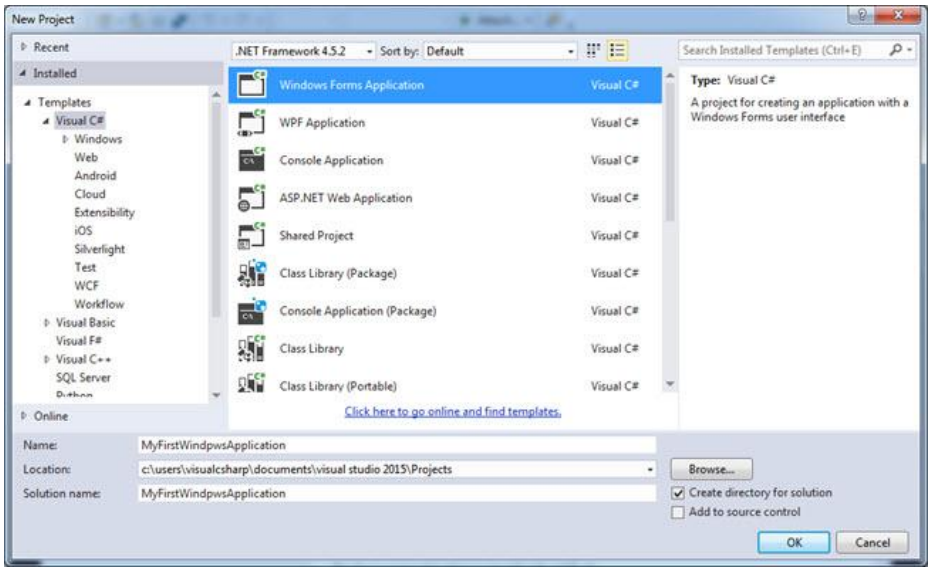
رابط گرافیکی کاربر (GUI) به کاربر اجازه می‌دهد که با استفاده از اجزای بصری مختلف با برنامه ارتباط برقرار کند. در روزهای اولیه دنیای کامپیوتر برنامه‌ها مبتنی بر متن بودند، بدین معنی که شما باید دستورات متنی زیادی برای ایجاد یک برنامه مفید تایپ می‌کردید و این کار مستلزم حفظ کردن یک لیست طولانی از دستورات بود. برنامه‌های نرم افزاری امروزی، دارای رابط گرافیکی هستند. این رابط گرافیکی تقریباً در همه برنامه‌هایی که امروزه با آنها سر و کار دارید به چشم می‌خورد. یک رابط گرافیکی حرفه‌ای باید جذاب و ساده باشد. ایجاد یک برنامه با رابط کاربری، قبلاً یک کار سخت و کسل کننده بود. مثلاً برای ایجاد یک پنجره ساده که یک متن را نمایش دهد، نیاز بود که تعداد زیادی کد تایپ شود. اما با ورود ویژوال استودیو این کار راحت شد. برنامه‌نویسی ویژوال ایجاد برنامه‌های گرافیکی را راحت کرد، به طوری که شما می‌توانید محیط برنامه خود را با کشیدن کنترل‌های لازم از جعبه ابزار به نوعی "نقاشی" کنید. کنترل‌ها اجزای بصری هستند که GUI یا رابط گرافیکی را تشکیل می‌دهند. نمونه‌ای از کنترل‌ها عبارت‌اند از buttons، text boxes، labels، check boxes و radio buttons کلمه "visual" در Visual C# از مفهوم برنامه‌نویسی ویژوال یا بصری گرفته شده است. مایکروسافت از کلمه Windows Forms برای نشان دادن هر پنجره در یک برنامه استفاده می‌کند. برنامه ویژوال استودیو اجازه ایجاد هر چه راحت تر برنامه‌های ویندوزی را به شما می‌دهد. برای ایجاد و طراحی فرم‌ها می‌توانید از حالت Design استفاده کنید.



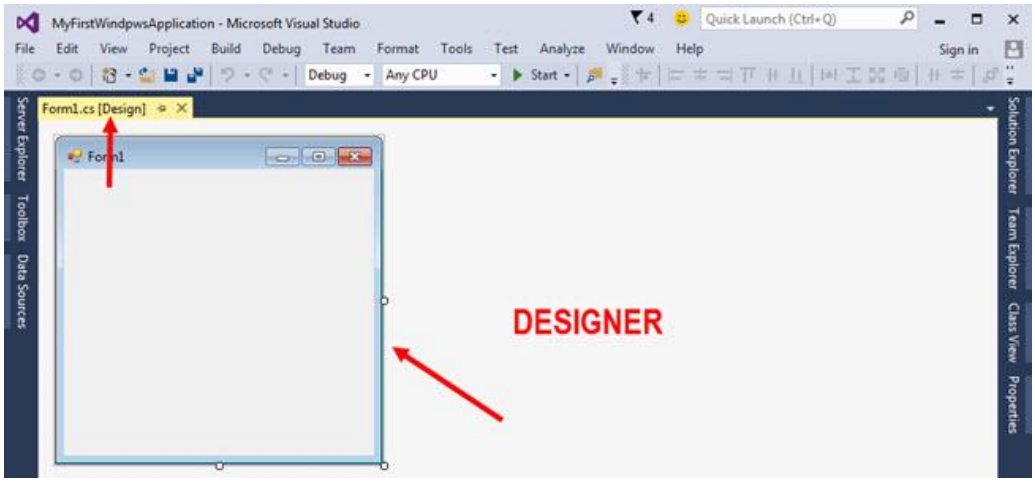
شکل بالا حالت طراحی (Design) در Visual Studio Community را نشان می‌دهد. در این شکل فرم ویندوزی و کنترل‌هایی که بر روی آن کشیده شده‌اند نشان داده شده است. در حالت طراحی، شما می‌توانید چگونگی به نظر رسیدن فرم را در حین اجرای برنامه، مشاهده کنید. کدهایی که باعث ایجاد و مقداردهی به کنترل‌ها می‌شوند از دید کاربر مخفی هستند، بنابراین شما می‌توانید بر روی کارکرد برنامه بیشتر تمرکز کنید. همچنین می‌توانید از ابزارهای ویژوال استودیو مانند، چپ چین یا راست چین کردن، تغییر اندازه و ... برای طراحی کنترل‌ها استفاده کرد.

ایجاد یک برنامه ویندوزی ساده

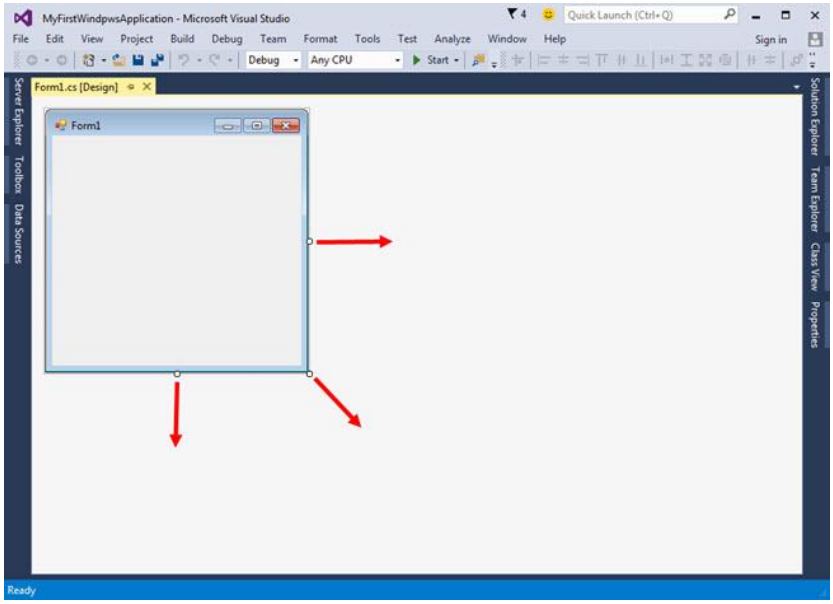
وقت آن رسیده است که برنامه‌نویسی ویژوال را تجربه کنید. به این نکته توجه کنید که این درس به شما نحوه ایجاد یک برنامه ویندوزی که در آن از یک اداره کننده رویداد (event-handling) استفاده شده است، را نشان می‌دهد. من هر مرحله را به صورت گام به گام انجام داده و به طور مختصر در مورد هر کدام توضیح می‌دهم. مفاهیم فرم، کنترل، کنترل کننده رویداد (event-handling) و برخی قسمت‌های Visual Studio در طراحی یک برنامه ویندوزی مورد استفاده قرار می‌گیرد و در مورد هر کدام از آنها در درس‌های مربوطه‌شان بحث خواهد شد. برنامه ویژوال سی‌شارپ را باز کنید و به مسیر File > New Project بروید. سپس از لیست ظاهر شده مانند شکل زیر گزینه Windows forms Application را انتخاب نمایید. یک Windows forms Application نوعی برنامه است که دارای یک رابط کاربر گرافیکی می‌باشد. نام پروژه را MyFirstWindowsApplication بگذارید.



بر روی دکمه OK کلیک کنید تا یک فرم خالی ظاهر شود. تب انتخاب شده نشان می‌دهد که شما در حال تماشای فایل Form1.cs از محیط طراحی هستید.

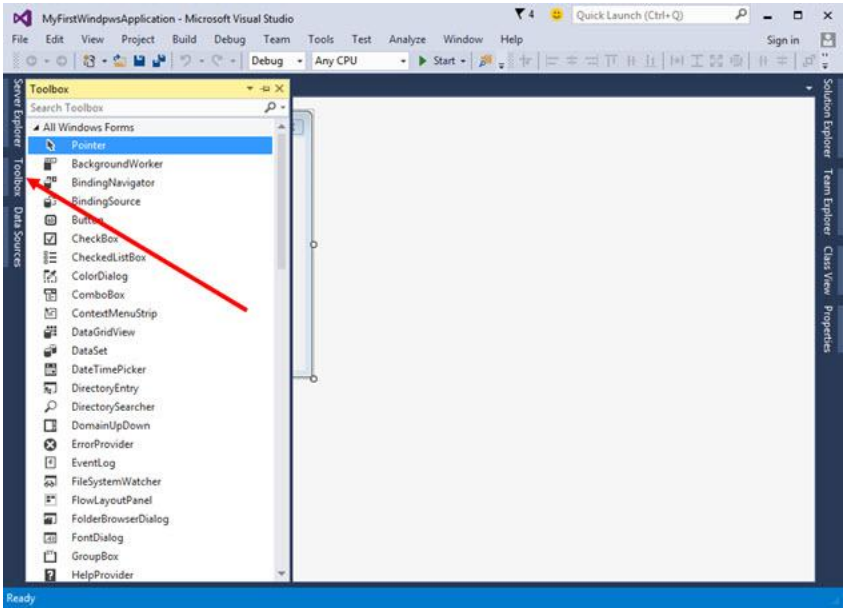


در حال حاضر بر روی کد فایلی که باعث افزوده شدن قابلیت به فرم می‌شود، تمرکز می‌کنیم. `form` را می‌توان در دو حالت مشاهده کرد، حالت طراحی و حالت کد. در حالت طراحی می‌توانید فرم، کنترل‌های ویژوال و غیر ویژوال را مشاهده نمایید. شکل زیر یک فرم را نشان می‌دهد، که شما می‌توانید با استفاده از دستگیره‌های کناری (که با فلش نشان داده شده است) در محیط طراحی اندازه آنرا تغییر دهید.

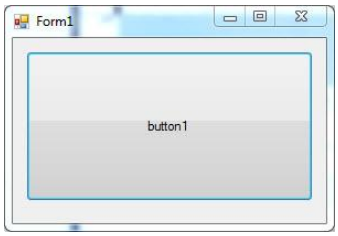


اضافه کردن کنترل‌ها به فرم

همه کنترل‌ها در جعبه ابزار یا `Toolbox` قرار دارند. `Toolbox` به صورت پیش فرض در سمت چپ محیط `IDE` قرار دارد. اگر جعبه ابزار را پیدا نکردید، می‌توانید از مسیر `View > Other Windows > Toolbox` به آن دسترسی یابید. برای نمایش جعبه ابزار کافیست، با ماوس بر روی آن لحظه‌ای توقف کرده و یا کلیک نمایید.

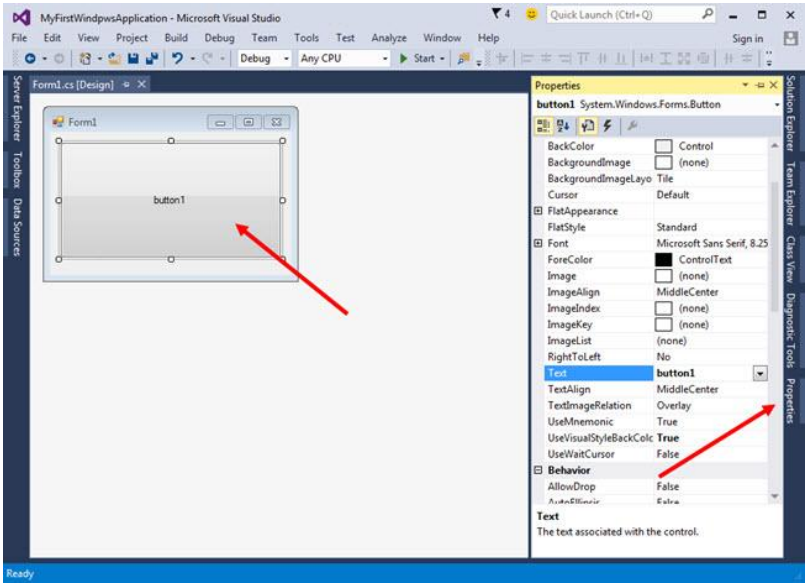


Toolbox به قسمت‌هایی تقسیم شده است، مثلاً کنترل‌های پر کاربرد در قسمت Common Controls قرار دارند. برای باز کردن یک دسته و نمایش کنترل‌های آن کافیست، بر روی نام دسته کلیک کنید. Toolbox در حالت پیش‌فرض مخفی می‌شود (خاصیت auto-hide). اگر شما این حالت را دوست ندارید، می‌توانید بر روی آیکون سنجاق شکل کنار دکمه close جعبه ابزار، کلیک کنید تا Toolbox ثابت بماند و مخفی نشود. برای اضافه کردن یک کنترل به فرم می‌توانید بر روی آن دوبار کلیک کنید تا به فرم اضافه شود. همچنین می‌توانید کنترل را از جعبه ابزار بر روی فرم کشیده و رها کنید (drag and drop). به این نکته توجه کنید که شما فقط می‌توانید کنترل‌ها را به ناحیه کاربر فرم اضافه کنید. ناحیه کاربر ناحیه خالی فرم است. برای حذف یک کنترل از روی فرم آن را انتخاب کنید و سپس دکمه Delete بر روی صفحه کلید را فشار دهید. یک کنترل دکمه (button) را به فرم اضافه کنید. بیشتر کنترل‌ها قابلیت تغییر اندازه را دارند. حال مکان و اندازه کنترل ذکر شده را مانند شکل زیر تغییر دهید.

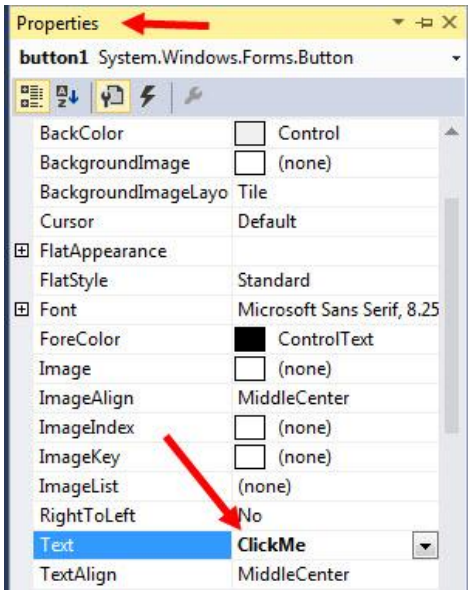


تغییر خاصیت کنترل‌ها

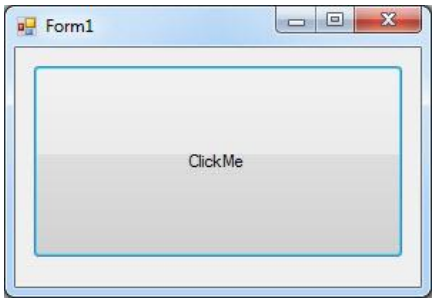
شما می‌توانید برخی از خواص فرم و کنترل‌ها را تغییر دهید. برای این کار از پنجره Properties برای مشاهده و تغییر مقادیر همه خواص در دسترس یک کنترل انتخاب شده در صفحه طراحی استفاده می‌کنیم. به این نکته توجه کنید که برخی از خواص در پنجره Properties نمایش داده نمی‌شوند و فقط از طریق کد می‌توان به آنها دست یافت. یک کنترل را با استفاده از کلیک کردن بر روی آن در صفحه طراحی انتخاب کنید. برای اینکه به شما نشان دهیم که چطور می‌توان خاصیت یک کنترل را تغییر داد، بر روی دکمه‌ی روی فرم کلیک کنید و سپس به پنجره Properties بروید.



خاصیت Text را پیدا کنید و آن را به Click Me تغییر دهید.



مشاهده می‌کنید که متن کنترل button تغییر می‌کند.



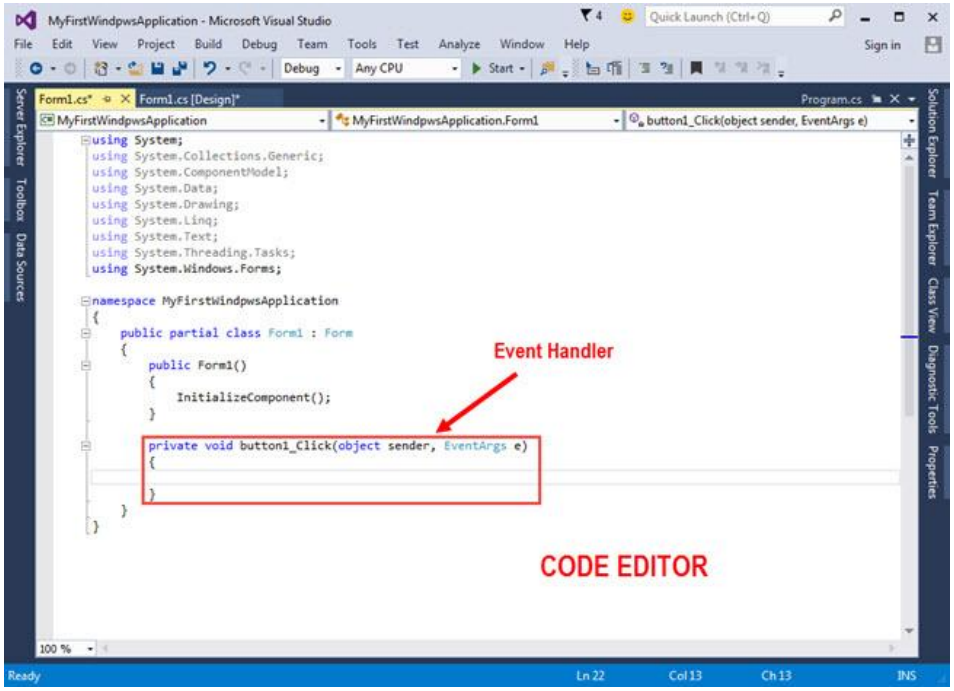
همچنین می‌توانید با کلیک بر روی فرم خاصیت Text آن را نیز تغییر دهید. به این نکته توجه کنید که برای انتخاب فرم باید بر روی یک منقطه خالی از آن کلیک کنید، نه یک کنترل که بر روی آن قرار دارد.

اضافه کردن یک کنترل کننده رویداد (Event Handler) به کنترل

در بخش آخر این آموزش قصد داریم که به شما نحوه اضافه کردن یک کنترل کننده رویداد را به کنترل، آموزش دهیم. کنترل کننده رویداد یا Event Handler، قسمتی از برنامه است که مسئولیت کنترل رویدادها را بر عهده دارد. رویداد وقتی به وقوع می‌پیوندد، که یک اتفاق معین رخ دهد. Event Handler هم برای کنترل کردن یک رویداد به کار می‌رود. در مورد جزییات کنترل کننده رویداد در یک بخش جداگانه بحث خواهیم کرد. هر کنترل یک رویداد پیش‌فرض مخصوص به خود دارد.

به عنوان مثال رویداد پیشفرض کنترل دکمه، Click و رویداد پیش‌فرض کنترل فرم، Load می‌باشد. کنترل کننده‌های رویداد، متدهایی هستند که به رویداد وابسته‌اند و وقتی اجرا می‌شوند که رویدادها رخ دهند. راه ساده برای اضافه کردن یک کنترل کننده رویداد، دابل کلیک کردن بر روی کنترل در محیط طراحی می‌باشد. ناگفته نماند که این کار باعث اضافه شدن کنترل کننده رویداد به رویداد پیش‌فرض می‌شود. برای روشن شدن

این مطلب بر روی دکمه button در محیط طراحی کلیک کنید. ویژوال استودیو به صورت خودکار یک کنترل کننده رویداد ایجاد کرده و آن را به رویداد Click کنترل می‌چسباند. همزمان با ایجاد کنترل کننده رویداد، شما وارد قسمت کد نویسی می‌شوید و نشانگر ماوس نیز در داخل کنترل کننده رویداد، قرار می‌گیرد. حال همه چیزی که شما نیاز دارید، نوشتن کدی است که هنگام وقوع رویداد اجرا می‌شود.

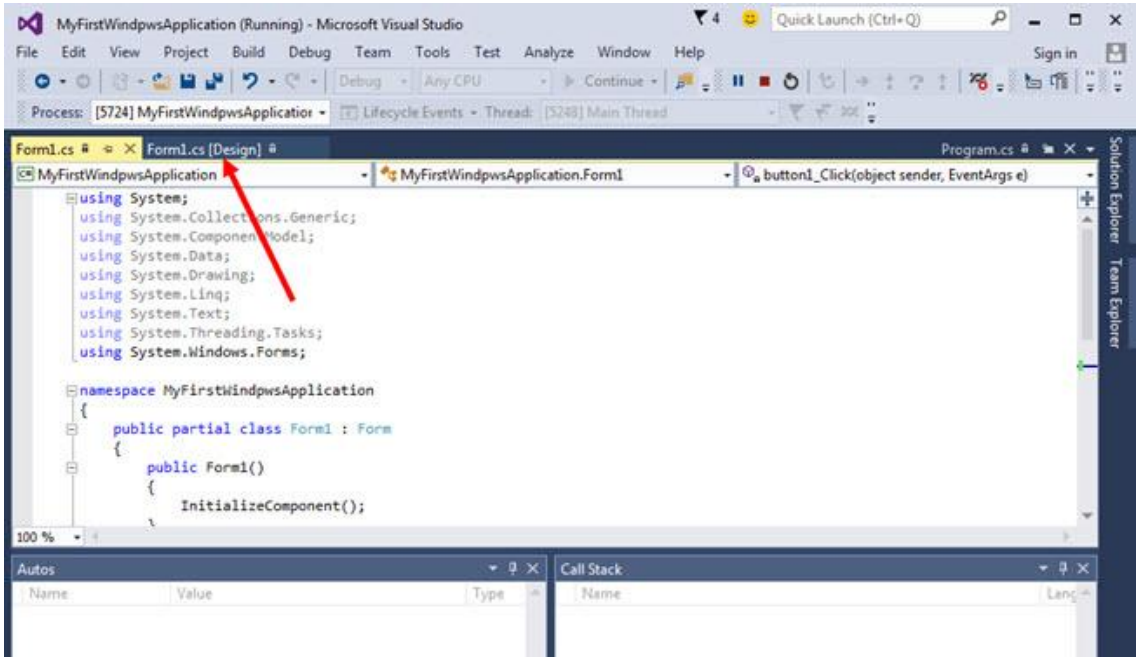


در مورد سایر قسمت‌های کد، در درس‌های آینده بحث خواهیم کرد. کد MessageBox.Show("You clicked the button!"); را در داخل کنترل کننده رویداد مربوط به رویداد Click به صورت زیر بنویسید.

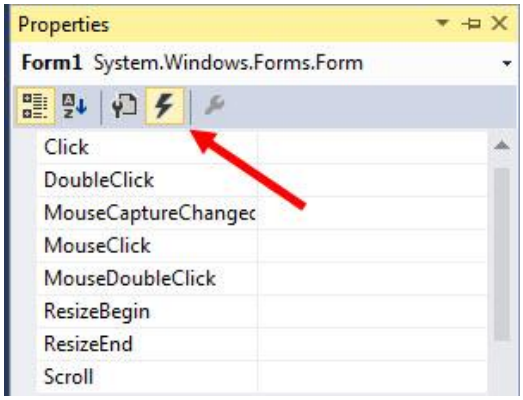
```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("You clicked the button!");
}
```

کلاس MessageBox به شما اجازه می‌دهد که یک جعبه متن را برای نمایش یک پیغام یا اطلاعات فوری به کاربر فراخوانی کنید. متد Show() باعث نمایش یک متن خاص در جعبه متن می‌شود. در مورد کلاس MessageBox در درس‌های آینده توضیح می‌دهیم. برنامه را اجرا کرده و بر روی دکمه کلیک کنید. با اجرای برنامه یک پنجره پیغام، که شامل پیامی است که به عنوان آرگومان به متد Show() ارسال کرده‌اید، نمایش داده خواهد شد.

راه دیگر برای اضافه کردن یک کنترل کننده رویداد به رویداد کنترل‌ها، مخصوصاً به رویدادهای غیر پیش‌فرض، استفاده از پنجره Properties است. برای روشن شدن این مطلب، اجازه دهید که رویداد Load را به فرم اضافه کنیم. برای برگشتن به محیط طراحی هم می‌توانیم بر روی تب Design کلیک کرده وهم از کلیدهای ترکیبی Shift + F7 استفاده کنیم.



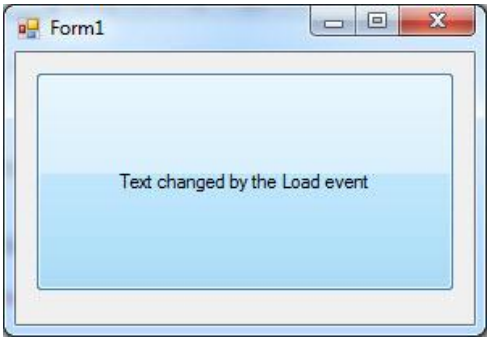
کنترل form انتخاب کرده و سپس از پنجره Properties رویدادهای آن را پیدا کنید. رویدادها در پنجره Properties به وسیله یک آیکون جرقه نمایش داده شده‌اند. اگر این آیکون را نیافتید مطمئن شوید که کنترل در محیط طراحی انتخاب شده باشد.



حال پنجره Properties لیستی از رویدادهای مربوط به کنترل انتخاب شده در محیط برنامه‌نویسی را نشان می‌دهد. رویداد Load مربوط به کنترل form را پیدا کنید. با کلیک بر روی منوی باز شونده (combo box) کنار آن، لیستی از متدهای موجود برای این رویداد نشان داده خواهد شد. سپس می‌توانید یک متد برای الحاق به این رویداد انتخاب کنید. همچنین می‌توان با دابل کلیک بر روی رویداد انتخاب شده در پنجره Properties یک کنترل کننده رویداد جدید، ایجاد کرد که در این صورت به صورت خودکار وارد محیط کدنویسی شده و کنترل کننده رویداد مناسب برای شما ایجاد می‌شود. کد پر رنگ شده زیر را در رویداد Load بنویسید.

```
private void Form1_Load(object sender, EventArgs e)
{
    button1.Text = "Text changed by the Load event";
}
```

این دستور خاصیت Text مربوط به دکمه واقع بر روی فرم را تغییر می‌دهد. رویداد Load فرم وقتی روی می‌دهد که بارگذاری فرم تمام شود. بنابراین با هر بار اجرای برنامه، متن داخل کنترل button تغییر می‌کند.



حال شما با موفقیت یک گرداننده رویداد را با استفاده از ابزارهای موجود در Visual Studio و Visual C# Express ایجاد کردید.

کنترل کننده رویداد (Event Handler)

رابط گرافیکی کاربر در دات نت و ویژوال سی‌شارپ از مکانیزم کنترل کننده رویداد برای کنترل رویدادها که در هنگام اجرای برنامه به وقوع می‌پیوندند، استفاده می‌کند. رویدادها، رفتارهایی یا اتفاقاتی هستند که هنگام اجرای برنامه به وقوع می‌پیوندند. کنترل رویداد فرایند نظارت بر وقوع یک رویداد مشخص می‌باشد. فرم‌های ویندوزی از کنترل کننده رویداد برای اضافه کردن یک قابلیت و پاسخ به کاربر استفاده می‌کنند. بدون کنترل کننده رویداد، فرم‌ها و رابط گرافیکی تا حد زیادی بدون استفاده هستند. در این درس فرض بر این است که شما بر مفاهیم delegates و events (رویداد) تسلط دارید. رویدادها با استفاده از یک delegate به عنوان یک نوع تعریف می‌شوند. Delegate ها آدرس متدها را در خود ذخیره می‌کنند. در مثال زیر یک delegate و یک event تعریف شده‌اند.

```
public delegate void SampleEventHandler(int);

public event SampleDelegate SampleEvent;
```

بر اساس تعریف بالا، delegate آدرس متدهایی را قبول می‌کند که دارای مقدار برگشتی نیستند و یک آرگومان ساده از نوعint قبول می‌کنند. سپس از این نوع delegate برای ایجاد event مان استفاده می‌کنیم. حال یک کنترل کننده رویداد به رویداد اضافه می‌کنیم. همانطور که قبلاً اشاره شد، هر رویداد دارای یک delegate مخصوص به خود است که این delegate دارای یک یا چند متد می‌باشد. کنترل کننده‌های رویداد، متدهایی هستند که امضای آنها همانند امضای این delegate است و هنگام وقوع رویداد اجرا می‌شوند. آنها (کنترل کننده‌های رویداد) چسبیده به یک event هستند و هنگامی که رویداد به وقوع می‌پیوندد اجرا می‌شوند. می‌توان چندین کنترل کننده‌های رویداد را به یک event متصل کرد تا هنگام وقوع رویداد اجرا شوند. برای این کار ابتدا باید کنترل کننده رویداد را ایجاد کنید. بعد از ایجاد، مطمئن شوید که امضای آن با امضای

delegate ی که رویداد از آن استفاده می‌کند، مطابق باشد. به عنوان مثال delegate مثال بالا دارای نوع برگشتی void بوده و یک پارامتر از نوع int دریافت می‌کند، پس کنترل کننده رویداد ما نیز باید دارای نوع برگشتی void بوده و یک پارامتر از نوع int دریافت کند (به این نکته توجه کنید که سطح دسترسی مهم نیست).

```
public void ShowMessage(int number)
{
    MessageBox.Show("Hello World");
}
```

سپس می‌توان با استفاده از عملگر += یک رویداد را متصل کرد:

```
SampleEvent += new SampleEventHandler(ShowMessage);
```

برای فعال کردن رویداد به همان روشی که متدها را فراخوانی می‌کردیم آن را صدا زده و آرگومانی را که لازم دارد به آن ارسال می‌کنیم.

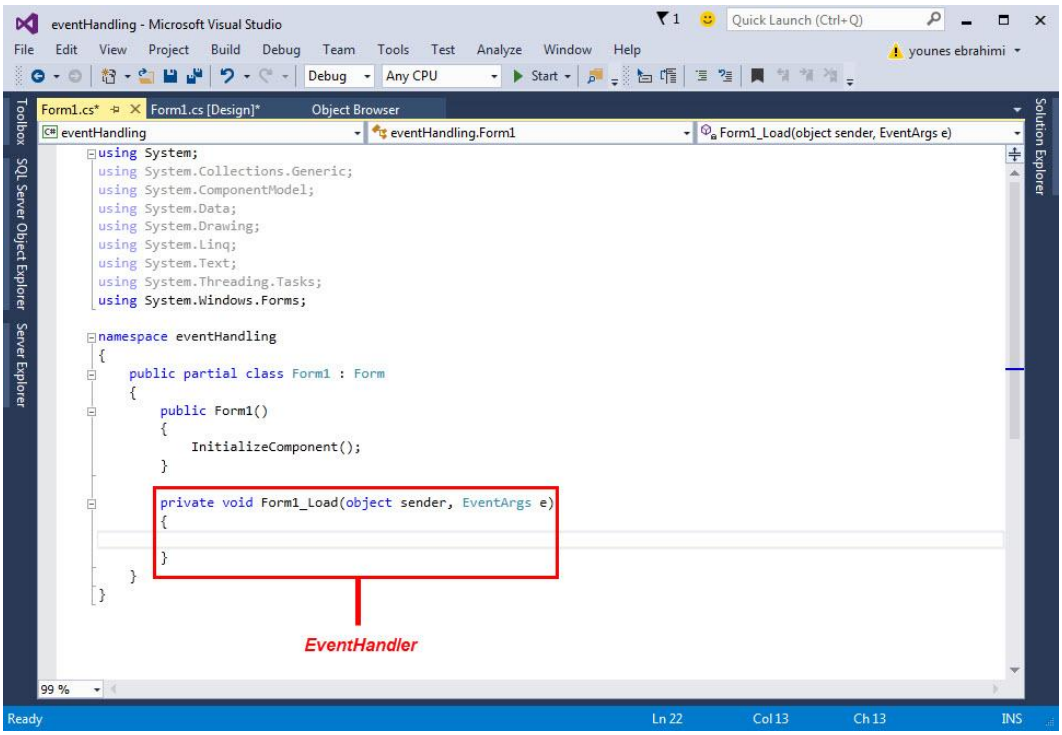
```
ShowMessage(3);
```

کنترل کننده رویداد در فرم‌های ویندوزی

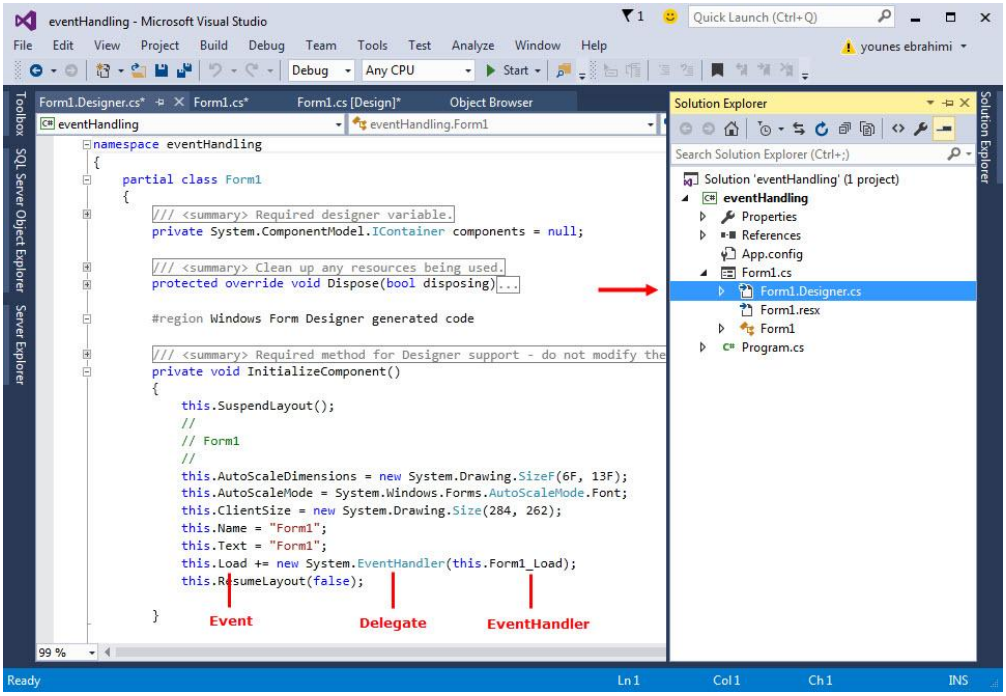
قبل از توضیح کنترل کننده رویداد در ویندوز فرم ابتدا ارتباط بین کنترل کننده رویداد، delegate و رویداد را دوباره یادآوری می‌کنیم. کنترل کننده رویداد، متدی است که در داخل delegate قرار می‌گیرد و به وسیله عملگر += به رویداد معرفی می‌شود. هنگامی که رویداد به وقوع می‌پیوندد، کدهای این متد اجرا می‌شوند. جمله بالا در کد زیر خلاصه می‌شود:

```
SampleEvent += new SampleDelegate(SampleEventHandler);
```

به این نکته هم اشاره کنیم که در سی‌شارپ تعداد زیادی event و delegate از پیش تعریف شده وجود دارد که ویژوال استودیو مسئولیت اداره آنها را به عهده دارد. برای شرح استفاده از رویدادها در فرم‌های ویندوزی، یک فرم ویندوزی جدید ایجاد کرده و نام آن را eventHandling بگذارید. بر روی فرم دوبار کلیک کرده تا ویژوال استودیو به صورت خودکار یک کنترل کننده رویداد ایجاد کرده و آن را به رویداد Load فرم متصل کند. به شکل زیر توجه کنید:



همانطور که در شکل بالا مشاهده می‌کنید، یک کنترل کننده رویداد به نام Form1_Load به طور خودکار ایجاد می‌شود، که دارای دو پارامتر است. در مورد این دو پارامتر در ادامه توضیح می‌دهیم. نکته ای که بهتر است در همین جا به آن اشاره کنیم، نحوه نامگذاری کنترل کننده‌های رویداد توسط ویژوال استودیو است. در ویژوال استودیو کنترل کننده‌های رویداد بر طبق یک قرار داد نامگذاری می‌شوند. هنگام نامگذاری آنها ابتدا نام کنترل (که در خاصیت Name کنترل مشخص شده است)، سپس علامت زیر خط و بعد از آن نام رویداد می‌آید مانند Form1_Load. می‌توان این قرارداد را زمانی که کنترل کننده رویداد به وسیله چند رویداد مورد استفاده قرار می‌گیرد، نادیده گرفت که در آینده توضیح داده خواهد شد. همانطور که گفتیم ویژوال استودیو به صورت خودکار یک کنترل کننده رویداد ایجاد کرده و آن را به رویداد Load فرم متصل می‌کند. برای مشاهده رویداد ایجاد شده و اتصال آن به کنترل کننده رویداد، کافی است که در پنجره Solution Explorer بر روی Form1.Designer.cs دوبار کلیک کنید. به شکل زیر توجه کنید:



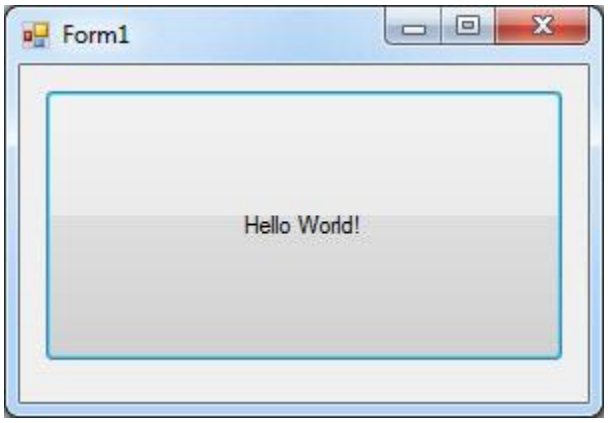
همانطور که در شکل بالا مشاهده می‌کنید، رویداد Load دارای یک delegate از نوع EventHandler است که کنترل کننده رویداد Form1_Loadدر داخل آن قرار می‌گیرد. ویژوال استودیو با استفاده از عملگر += این کنترل کننده را که قرار است هنگام وقوع رویداد Load اجرا شود را، به رویداد معرفی می‌کند. بیشتر رویدادهای کنترل‌ها یک delegate از نوع System.eventHandler دارند. مثال زیر تعریفی از دلیگیت eventHandler می‌باشد:

```
public delegate void EventHandler(object sender, EventArgs e)
```

همانطور که مشاهده می‌کنید، delegate مثال بالا دارای مقدار برگشتی نیست و دارای دو پارامتر است، یکی object و دیگری یک نمونه از eventArgs.

پارامتر object sender

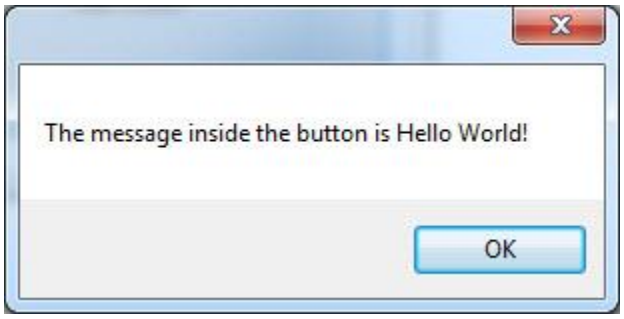
object sender، نشان دهنده کنترلی است که رویداد را فعال می‌کند و ویژوال استودیو از این آرگومان برای تشخیص کنترلی که رویداد مربوط به آن است، استفاده می‌کند (بعداً توضیح داده می‌شود). از آن جایی که نوع این پارامتر object است، هر کنترلی می‌تواند منبع ارسال رویداد باشد. چون هر کنترل یک شیء است که از کلاس پایه object مشتق می‌شود. از آنجایی که ارسال کننده رویداد (کنترل) به نوع object تبدیل می‌شود، برای استفاده مجدد از خصوصیات کنترل باید آن را با استفاده از عمل cast تبدیل کنیم (تولید کنیم). برای روشن شدن مطلب به مثال زیر توجه کنید. ابتدا یک کنترل button را روی فرم قرار دهید و سپس خاصیت Text آن را به "Hello World!" تغییر دهید.



بر روی کنترل دوبار کلیک کنید تا یک کنترل کننده رویداد برای رویداد Click آن ایجاد شود. مانند رویداد Load، رویداد Click نیز یک کنترل کننده رویداد دارد. کد زیر را در داخل کنترل کننده رویداد بنویسید:

```
private void button1_Click(object sender, EventArgs e)
{
    Button source = (Button)sender;
    MessageBox.Show("The message inside the button is " + source.Text);
}
```

برنامه را اجرا کرده و بر روی دکمه کلیک کنید. یک جعبه متن نمایش داده خواهد شد. این متن هنگام وقوع رویداد Click نمایش داده می‌شود.



خط اول کنترل کننده رویداد شیء sender را با استفاده از عمل cast به یک دکمه تبدیل می‌کند، بنابراین می‌توانیم به خاصیت Text دکمه دسترسی داشته باشیم. سپس متد Show() از کلاس MessageBox را برای نشان دادن مقدار خاصیت Text دکمه که به رویداد فرستاده شده است، فراخوانی می‌کنیم.

پارامتر eventArgs

این پارامتر نمونه ای از کلاس eventArgs می‌باشد و می‌توان آن را به عنوان آرگومان رویداد در نظر گرفت که حاوی داده‌هایی در مورد رویدادی که اتفاق افتاده است، می‌باشد. eventArgs، در اصل یک کلاس پایه است و دارای اعضای مفیدی نمی‌باشد. این پارامتر در برخی از کنترل کننده‌های رویداد متفاوت است. مثلاً در رویداد Click این آرگومان EventArgs و در رویداد MouseClick به صورت MouseEventArgs می‌باشد. همانطور که گفتیم، کلاس پایه eventArgs، هیچ خاصیتی که شما بتوانید در کنترل کننده رویداد خود از آن استفاده کنید ندارد:

```
private void button1_Click(object sender, EventArgs e)
{
    e.
}
```

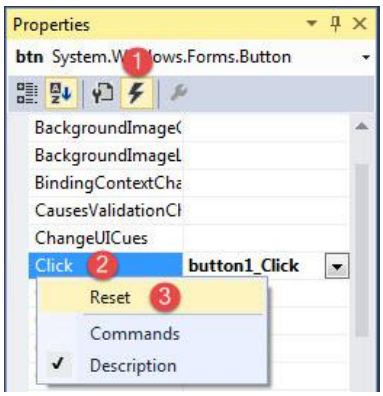
Equals
GetHashCode
GetType
ToString

اما MouseEventArgs دارای خواص مفیدی از جمله اینکه کدام دکمه ماوس فشار داده شده، تعداد کلیک‌ها، مقدار چرخش ماوس و مختصات مکانی که رویداد کلیک در آن اتفاق افتاده است، می‌باشد:

```
private void button1_MouseClick(object sender, MouseEventArgs e)
{
    e.
}
```

Clicks
Delta
Equals
GetHashCode
GetType
Location
ToString
X
Y

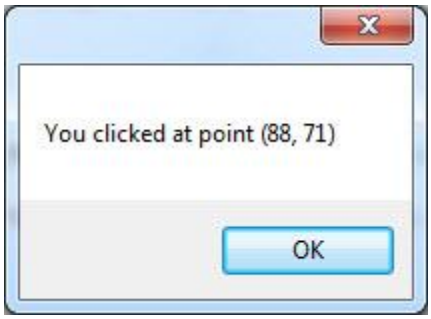
رویداد MouseClick کنترل button، نسخه بهتری از رویداد Click می‌باشد. از آنجایی که این رویداد (MouseClick) رویداد پیشفرض نیست، برای دسترسی به آن باید از بخش events از پنجره Properties اقدام نمایید. مطمئن شوید که کنترل button در قسمت طراحی فرم انتخاب شده است. بر روی آیکون events (که به شکل جرقه است) کلیک کنید. ابتدا کنترل کننده رویداد Click را حذف کنید تا با این رویداد (MouseClick) تداخل پیدا نکند. برای حذف رویداد Click از پنجره Properties بر روی آیکون جرقه شکل کلیک کرده و از لیست رویدادها، رویداد Click را پیدا کرده و سپس بر روی آن راست کلیک و سپس دکمه Reset را بزنید:



سپس رویداد MouseClick را یافته و بر روی آن دوبار کلیک کرده و کد زیر را در داخل کنترل کننده رویداد MouseClick آن بنویسید:

```
private void button1_MouseClick(object sender, MouseEventArgs e)
{
    MessageBox.Show("You clicked at point (" + e.X + ", " + e.Y + ")");
}
```

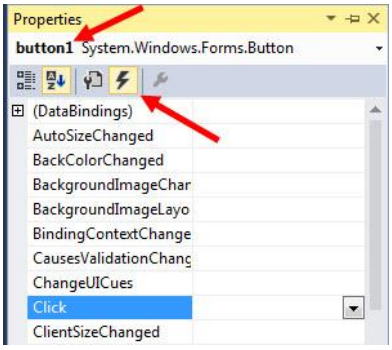
کد بالا با استفاده از پارامتر event argument به مختصات X و Y نقطه ای که با ماوس بر روی آن کلیک شده است، دست می‌یابد. خروجی زیر بستگی به این دارد که شما بر روی چه نقطه ای از دکمه کلیک کنید.



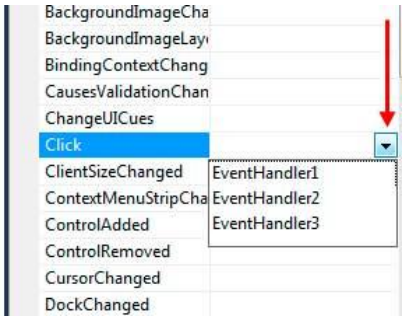
انواع زیادی event argument وجود دارد که هر کدام خواص مفیدی برای رویدادی که اتفاق می‌افتد، پیشنهاد می‌دهند.

استفاده از پنجره خواص برای اضافه کردن کنترل کننده رویداد

همانطور که می دانید صرف نظر از دوبار کلیک کردن بر روی کنترل‌ها برای ایجاد کنترل کننده رویداد می‌توانیم از پنجره Properties برای این کار نیز استفاده کنیم. بر روی آیکونی که شبیه به یک جرقه است کلیک کنید تا وارد بخش رویدادهای (Events) پنجره Properties شوید.



از منوی باز شونده ای که با فلش قرمز در شکل بالا نشان داده شده است برای انتخاب کنترل‌ها در محیط طراحی استفاده می‌شود. استفاده از این منو زمانی مفید است که شما بخواهید یک کنترل غیر قابل رویت و یا بسیار کوچک را انتخاب کنید. رویداد مورد نظر را پیدا کنید. برای اضافه کردن کنترل کننده رویداد به آن می‌توانید بر روی نام رویداد دوبار کلیک کنید تا کنترل کننده رویداد مربوط به آن ایجاد شود. همچنین اگر از قبل کنترل کننده‌های رویداد دیگری نیز ایجاد کرده‌اید، می‌توانید مانند شکل زیر به آنها دسترسی داشته باشید.



به این نکته توجه کنید که نام برخی از رویدادها در داخل نام رویدادهای پنجره Properties نیست. رویداد MouseWheel یکی از این رویدادهاست. برای دسترسی به چنین رویدادهایی می‌توانید از محیط کدنویسی و قابلیت IntelliSense استفاده کنید. البته این کار نیاز به کمی مهارت و یا حتی جستجو در بین سایت‌های مختلف دارد. ولی قواعد کلی همان است که در بالا اشاره شد. به کد زیر توجه کنید:

```
using System.Windows.Forms;

namespace eventHandling
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            this.button1.MouseWheel += new MouseEventHandler(this.button1_MouseWheel);
        }

        private void button1_MouseWheel(object sender, MouseEventArgs e)
        {
            this.button1.Width += e.Delta;
        }
    }
}
```

MouseWheel رویداد مربوط به حرکت دکمه وسط ماوس به طرف بالا و پایین است. همانطور که در کد بالا مشاهده می‌کنید کافیست که در قسمت سازنده form1 این رویداد را به مثلاً دکمه اضافه کنیم. سپس یک کنترل کننده رویداد (button1_MouseWheel) به آن وصل کنیم. کدهای بدنه کنترل کننده رویداد باعث می‌شوند که با چرخاندن دکمه وسط ماوس به سمت بالا و پایین عرض دکمه‌ی روی فرم کم و زیاد شود.

استفاده از یک کنترل کننده رویداد برای چندین رویداد

به این نکته توجه کنید که می‌توان از یک کنترل کننده رویداد برای چندین رویداد که دارای نماینده (delegate) یکسانی هستند، استفاده کرد. برای انجام این کار لازم است که یک کنترل کننده رویداد با یک امضای مناسب ایجاد کنید. مثلاً رویدادهای click و Load دارای امضاهای یکسانی هستند، پس می‌توان یک کنترل کننده رویداد را به این دو رویداد اختصاص داد. یک دکمه بر روی فرم قرار داده و سپس با زدن دکمه F7به محیط کدنویسی رفته و کد زیر را بنویسید:

```
private void myEventHandler(object sender, EventArgs e)
{
    MessageBox.Show("This is myEventHandler!");
}
```

بعد از ایجاد کنترل کننده رویداد بالا (myEventHandler)، با استفاده از عملگر += آن را به رویداد مورد نظرتان اختصاص دهید. فرض کنید که می‌خواهید قبل از اینکه فرم بالا بیاید و یا بر روی دکمه کلیک شود، این کنترل کننده رویداد اجرا شود. برای این منظور کنترل کننده رویداد را به صورت زیر به رویدادهای Load فرم و Click دکمه اختصاص دهید:

```
using System;
using System.Windows.Forms;

namespace eventHandling
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            this.Load += new EventHandler(myEventHandler);
            button1.Click += new EventHandler(myEventHandler);
        }

        private void myEventHandler(object sender, EventArgs e)
        {
            MessageBox.Show("This is myEventHandler!");
        }
    }
}
```

قبلاً ذکر شد که می‌توان قرارداد نامگذاری کنترل کننده‌های رویداد را زمانی که توسط چندین رویداد مورد استفاده قرار می‌گیرد، نادیده گرفت، این نکته را در کد بالا مشاهده می‌کنید. به این نکته توجه کنید که این کنترل کننده رویداد (myEventHandler) را برای هر رویدادی (مثلاً FormClosing) نمی‌توان مورد استفاده قرار داد. چون کنترل کننده‌های رویدادی که با اجرای این رویداد به وقوع می‌پیوندند در داخل Delegate ی قرار می‌گیرند که کنترل کننده‌های رویدادی با امضای زیر را قبول می‌کند:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
```

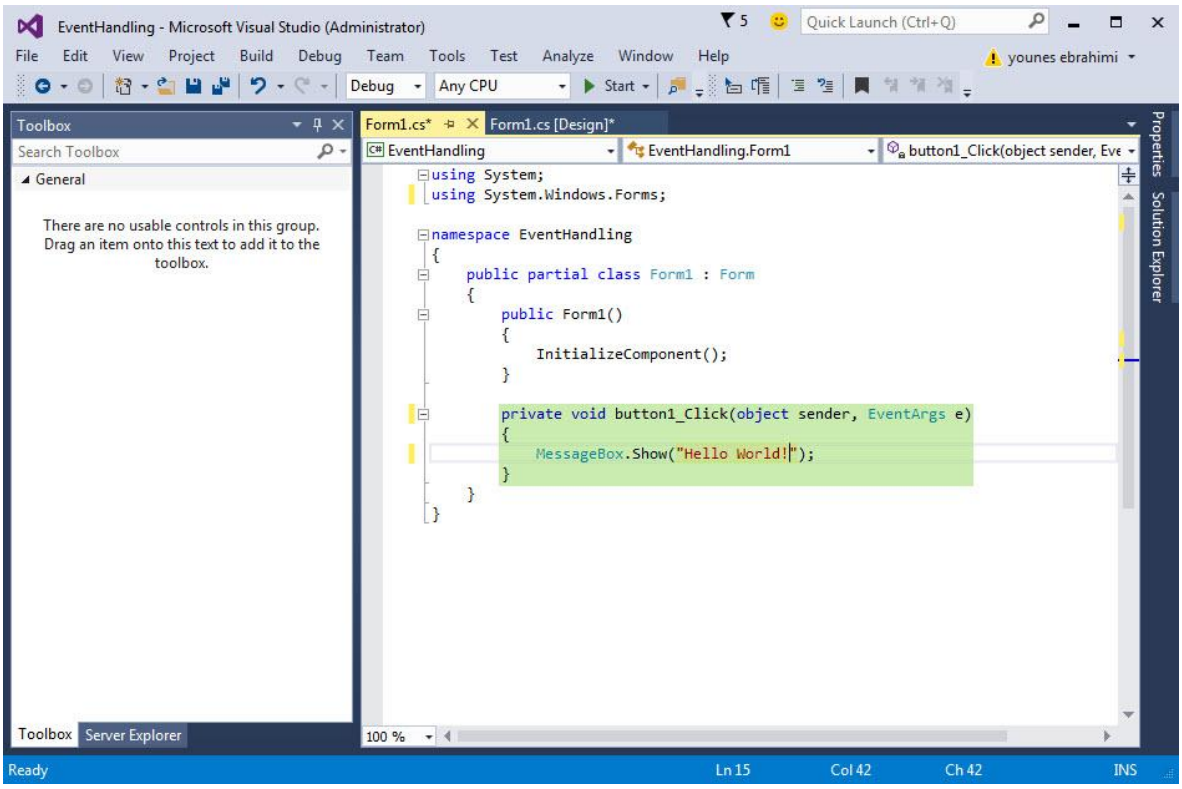
در حالیکه کنترل کننده رویداد ما دارای امضای زیر است:

```
private void myEventHandler(object sender, EventArgs e)
```

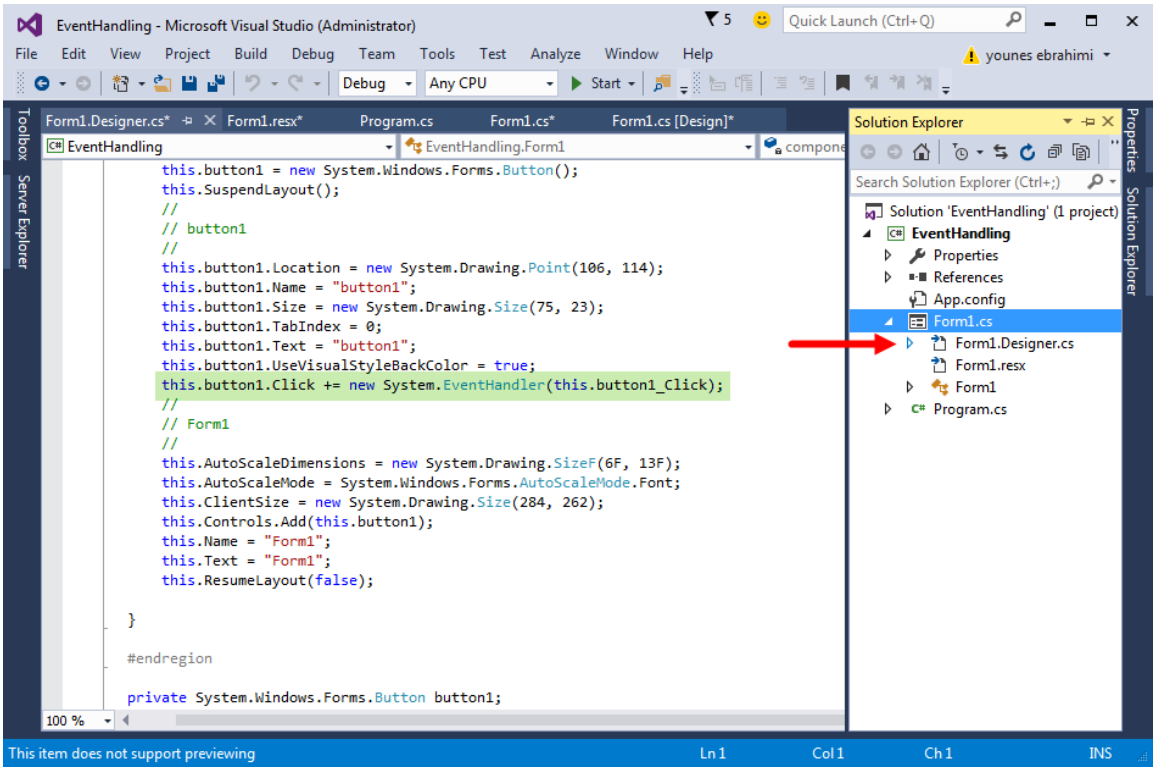
و در آخر به این نکته خیلی مهم توجه کنید که با کپی کردن یک کنترل کننده رویداد در محیط کدنویسی، کدهای داخل آن اجرا نمی‌شوند. برای روشن شدن مطلب یک برنامه ویندوزی ایجاد کرده و یک دکمه بر روی فرم قرار دهید. حال با زدن دکمه F7 به محیط کدنویسی رفته و کنترل کننده رویداد زیر را در آن کپی کنید:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World!");
}
```

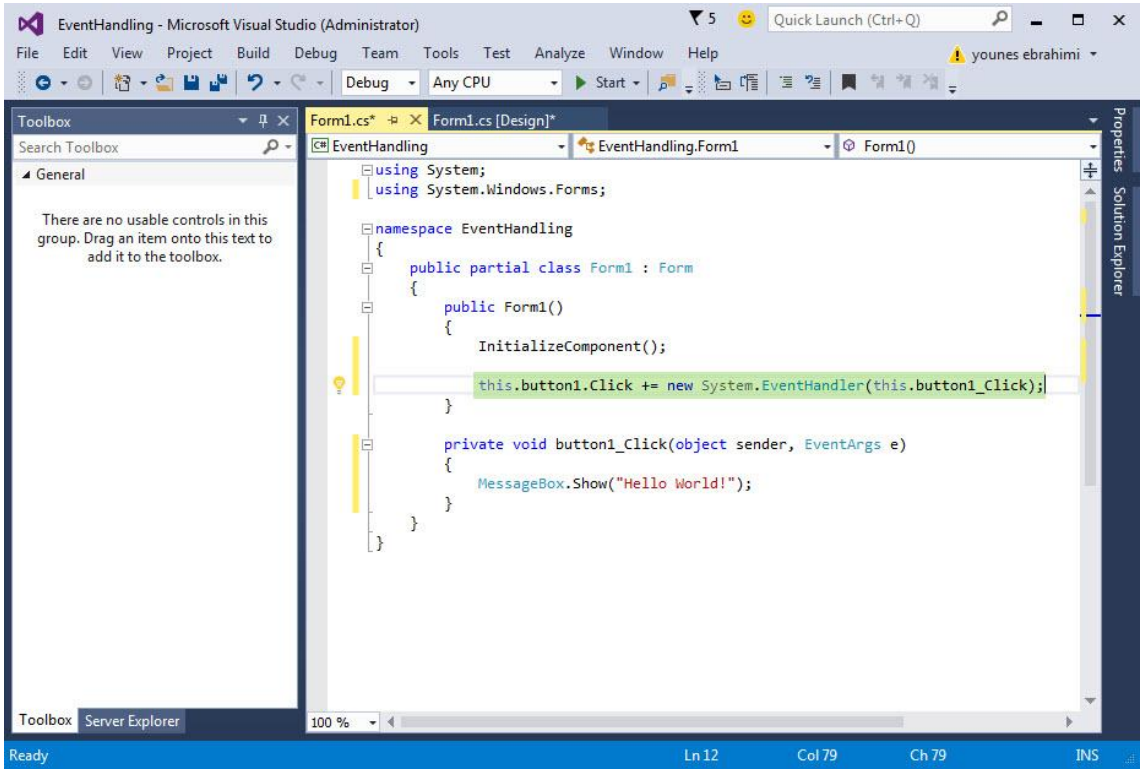
به صورت زیر



همانطور که از کد بالا پیداست، این کد برای رویداد کلیک دکمه نوشته شده است، به طوریکه اگر بر روی دکمه کلیک شود یک پیغام نمایش داده شود. حال با زدن دکمه F5 برنامه را اجرا کرده و بر روی دکمه کلیک کنید. مشاهده می‌کنید که هیچ پیغامی نمایش داده نمی‌شود. دلیل این امر روشن است و آن این است که ما این کنترل کننده رویداد را به رویداد Click وصل نکرده‌ایم. برای این منظور دو راه وجود دارد. اولین راه این است که برنامه را متوقف کرده و بر روی دکمه دو بار کلیک کنید تا ویژوال استودیو به طور خودکار این کار را انجام دهد. در این حالت اگر به صورت زیر بر روی فایل Form1.Designer.cs دو بار کلیک کنید مشاهده می‌کنید که کنترل کننده رویداد به رویداد Click متصل شده است:



حال اگر برنامه اجرا و بر روی دکمه کلیک کنید، مشاهده می‌کنید که پیغام نمایش داده می‌شود. راه دوم هم این است که خودتان به صورت دستی این کنترل کننده رویداد را به رویداد `Click` وصل کنید:

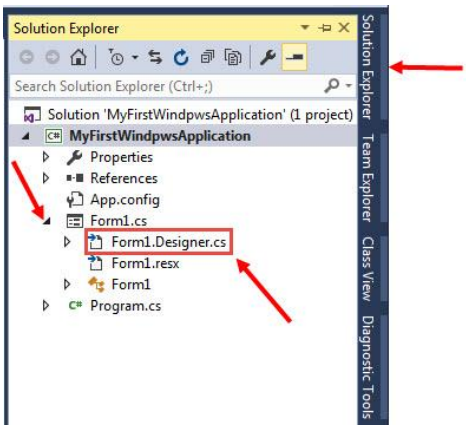


جدا کردن محیط طراحی از محیط کدنویسی

وقتی یک فرم ویندوزی جدید ایجاد می‌کنیم، یک کلاس که از System.Windows.Forms.Form ارث بری می‌کند تولید می‌شود. این کلاس به دو فایل جدا و وابسته به هم تقسیم می‌شود و یک کلاس تکه تکه (Partial class) را به وجود می‌آورد. کلاس‌های تکه تکه (Partial classes) به شما اجازه تعریف جداگانه یک کلاس در فایل‌های مختلف در داخل یک پروژه و فضای نام یکسان را می‌دهند. با معرفی کلاس‌های Partial در .NET 3.5 ویژوال استودیو قدرت جداسازی محیط طراحی از محیط کد نویسی را کسب کرد. این قابلیت به برنامه نویس اجازه می‌دهد که بر روی بخش کدنویسی برنامه تمرکز کند. به عنوان مثال نام فایل‌های یک فرم به نام FormName در محیط‌های کدنویسی و طراحی به صورت زیر است:

```
FormName.cs
FormName.Designer.cs
```

یک فایل سومی هم با پسوند regx. وجود دارد که در آینده درباره آن توضیح داده خواهد شد. بعد از ایجاد یک فرم، اضافه کردن کنترل و دستکاری خواص همه کدها، همه در یک فایل تا حدودی مخفی، با پسوند Designer.cs. نوشته می‌شوند. برای مشاهده این فایل می‌توانید از Solution Explorer استفاده کنید.



با دوبار کلیک بر روی این فایل، به شما اجازه مشاهده همه کدهای ایجاد شده توسط ویژوال استودیو داده می‌شود. این کدها شامل متدهایی برای مرتب سازی و مقداردهی به کنترل‌ها می‌باشند. شما می‌توانید کنترل‌هایی را که تعریف شده‌اند و خواصی مانند محل و متن کنترل، که شما در پنجره‌ی Properties به آنها مقادیری را اختصاص داده‌اید، را مشاهده نمایید. همچنین قسمت Event handler را که به رویدادهای کنترل‌ها وصل شده است، را مشاهده می‌کنید. اگر شما نمی‌توانید قسمت کد را ببینید، به طور پیش فرض مخفی است و به قسمت Windows Forms Designer generated code متصل است. برای نمایش آن بر روی آیکن + کلیک کنید تا قسمت کد نمایش داده شود. در آینده مشاهده خواهید کرد که مقدار دهی کنترل‌ها و خواص و رویدادهای مربوط به آنها در داخل متدی به نام InitializeComponent() صورت می‌گیرد. این متد در داخل سازنده فرم واقع در کدهای اصلی کلاس فرم فراخوانی می‌شود. کدهای نوشته شده در فایل Designer از چشم شما مخفی هستند چون Visual Studio می‌خواهد که شما به جای نوشتن کد به صورت دستی، از پنجره Properties و محیط طراحی برای ایجاد یک برنامه ویندوزی استفاده نمایید.

کلاس MessageBox

System.Windows.Forms.MessageBox یک کلاس استاتیک است که از آن برای نشان دادن یک پیغام فوری، اطلاعات و یا یک هشدار به کاربران استفاده می‌شود. برای نشان دادن یک پیغام به راحتی می‌توان از متد ()Show کلاس MessageBox استفاده نمایید. ساده‌ترین حالت متد ()Show این است که یک رشته متنی را به عنوان آرگومان قبول می‌کند و آن را نمایش می‌دهد.

```
MessageBox.Show("Hello World!");
```

شما همچنین می‌توانید به راحتی و با استفاده از یکی دیگر از سربارگذاری‌های متد ()Show یک عنوان برای جعبه پیامتان بگذارید.

```
MessageBox.Show("Hello World!", "A Message");
```

اگر نخواهید از دکمه OK پیش‌فرض جعبه متن استفاده کنید می‌توانید با استفاده از System.Windows.Forms.MessageBoxButtons آنرا تغییر دهید.

```
MessageBox.Show("Hello World!", "A Message", MessageBoxButtons.OKCancel);
```

جدول زیر اعضای MessageBoxButtons را نشان می‌دهد:

عضو	دکمه‌هایی که نمایش می‌دهد
AbortRetryIgnore	Abort, Retry, Ignore
OK	OK
OKCancel	OK, Cancel
RetryCancel	Retry, Cancel
YesNo	Yes, No
YesNoCancel	Yes, No, Cancel

متد Show() یک مقدار را از System.Windows.Forms.DialogResult بر می‌گرداند. تشخیص اینکه چه دکمه ای توسط شما در جعبه متن فشار داده می‌شود، مفید است. به عنوان مثال اگر بر روی دکمه Yes در جعبه پیام کلیک کنید، متد Show() مقدار DialogResult.Yes را بر می‌گرداند.

```
DialogResult result;
result = MessageBox.Show("What is your choice?");

if (result == DialogResult.Yes)
{
    //You pressed the Yes button
}
if (result == DialogResult.No)
{
    //You pressed the No button
}
```

لطفاً به این نکته توجه کنید که کلاس Form یک خاصیت DialogResult دارد و آن را با System.Windows.Forms.DialogResult اشتباه نگیرید. می‌توان به جعبه پیام برای نشان دادن معنی و مفهوم آن یک آیکون اضافه کرد. می‌توانید این کار را با استفاده از نوع شمارشی MessageBoxIcon انجام دهید.

```
MessageBox.Show("Hello World!", "A Message", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

در جدول زیر انواع آیکون‌ها و کاربرد آنها در جعبه پیام نشان داده شده است:

آیکون	عضو	استفاده
	Asterisk Information	برای نشان دادن اطلاعات به کاربر
	Error Hand Stop	برای نشان دادن یک پیغام خطا
	Exclamation Warning	برای نشان دادن یک هشدار
	Question	برای سؤال کردن از کاربر

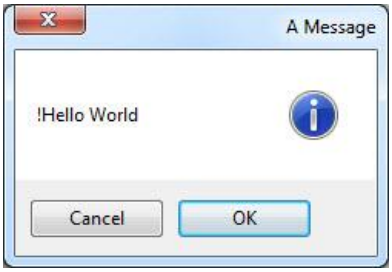
اگر بخواهید که جعبه متن هیچ گونه آیکونی نداشته باشد می‌توانید از MessageBoxIcon.None استفاده کنید. نوع شمارشی MessageBoxDefaultButton دکمه پیشفرضی را که هنگام فشرده شدن دکمه Enter باید عمل کند را تعیین می‌کند و فقط دارای سه عضو است Button1، Button2 و Button3. به عنوان مثال در جعبه متنی که دارای دکمه‌های OK و Cancel است استفاده از MessageBoxDefaultButton.Button1 باعث می‌شود که دکمه OK به صورت پیشفرض درآید. یعنی هنگامی که جعبه متن نشان داده شد، با زدن دکمه Enter، دکمه OK فشرده می‌شود:

```
MessageBox.Show(
                "Hello World!",
                "A Message",
                MessageBoxButtons.OKCancel,
                MessageBoxIcon.Information,
                MessageBoxDefaultButton.Button1
                );
```

همانطور که در کد بالا مشاهده می‌کنید، رنگ دکمه OK نسبت به دکمه Cancel متفاوت است که نشان دهنده پیشفرض بودن این دکمه است.

متد Show() دارای پارامتر دیگری از نوع شمارشی MessageBoxOptions است که دارای مقادیر مختلفی است. یکی از این مقادیر RtlReading بوده که از آن برای راست به چپ کردن پیغام استفاده می‌شود:

```
MessageBox.Show(
                "Hello World!",
                "A Message",
                MessageBoxButtons.OKCancel,
                MessageBoxIcon.Information,
                MessageBoxDefaultButton.Button1,
                MessageBoxOptions.RtlReading
                );
```



کنترل‌ها

کنترل‌ها، اجزای بصری هستند که به وسیله‌ی آنها محیط‌های گرافیکی (GUI) ساخته می‌شود. هرچیزی که شما در یک محیط گرافیکی می‌بینید یک کنترل است، حتی محیط فرم نیز یک کنترل است. کنترل‌ها در دسته‌های مختلفی در قسمت نوار ابزار قرار دارند. بیشتر کنترل‌ها از کلاس پایه System.Windows.Forms.Control ارث بری می‌کنند که دارای خواص، متدها و رویدادهای متفاوتی برای آنها می‌باشد.

خواص کنترل‌ها

در قسمت زیر بعضی از مشخصه‌های مفید کلاس Control را مشاهده می‌کنید.

خاصیت	توضیح

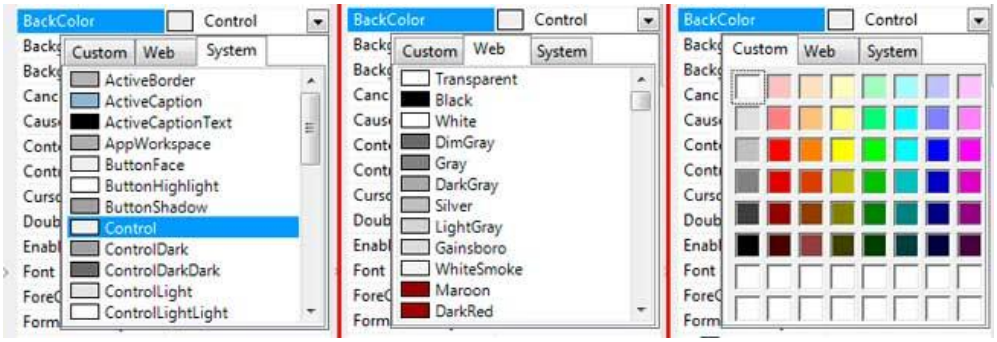
Anchor	این گزینه مشخص می‌کند که کنترل‌ها وقتی که سایز فرم تغییر می‌کند چگونه محل و سایزشان تغییر کند.
AutoSize	وقتی که مقدار آن برابر با True باشد، سایز کنترل به صورت اتوماتیک براساس محتوای آن تغییر می‌کند.
BackColor	رنگ قسمت زمینه‌ی کنترل را مشخص می‌کند.
BackgroundImage	به شما اجازه می‌دهد که یک عکس به زمینه‌ی کنترل خود اضافه کنید.
BackgroundImageLayout	حالت عکس زمینه‌ی کنترل را مشخص می‌کند.
ContextMenuStrip	به شما اجازه می‌دهد که یک توضیح را به کنترل خود در مورد آن اضافه کنید.
Controls	مجموعه‌ای از کنترل‌های مرتبط با کنترل مربوطه را نشان می‌دهد.
Dock	کنترل را به یکی از لبه‌های پنجره می‌چسباند.
Enabled	این قسمت مشخص می‌کند که کنترل فعال است یا نه. اگر مقدار آن False باشد کنترل غیرفعال می‌شود.
ForeColor	رنگ پیش زمینه‌ی کنترل و همچنین رنگ قلم کنترل را مشخص می‌کند.
Height	طول کنترل را بر مبنای پیکسل مشخص می‌کند.
Location	محل کنترل بر اساس در بر دارنده‌ی آن را مشخص می‌کند.
Locked	مشخص می‌کند که کنترل در حالت Design می‌تواند جابجا شود و یا تغییر سایز دهد یا نه.
MaximumSize	حداکثر اندازه‌ای که کنترل می‌تواند داشته باشد را، مشخص می‌کند.
Margin	فاصله‌ی بین لبه‌های کنترل مربوطه و لبه‌ی کنترل دیگر را مشخص می‌کند.
MinimumSize	کمترین سایز کنترل را مشخص می‌کند.
Name	نام کنترل را مشخص می‌کند. این نام برای اشاره به کنترل در زمان کد نویسی استفاده می‌شود.
Parent	والد کنترل را مشخص می‌کند.
Right	فاصله‌ی بین ضلع راست کنترل و ضلع چپ در بردارنده ی آن را مشخص می‌کند.
Size	سایز کنترل را مشخص می‌کند. این بخش شامل طول و عرض کنترل است.
TabIndex	با اختصاص یک عدد یه این خاصیت مشخص می‌کنیم که با چند بار زدن دکمه Tab، کنترل فوکوس بگیرد.
TabStop	مشخص می‌کند که یک کنترل می‌تواند توسط دکمه‌ی Tab مورد دستیابی قرار گیرد یا خیر.
Text	متنی که در داخل کنترل قرار می‌گیرد.

TextAlign	نحوه‌ی قرار گیری متن در داخل کنترل را مشخص می‌کند.
Top	فاصله‌ی بین ضلع بالایی کنترل و ضلع بالایی در بر دارنده‌ی آن را مشخص می‌کند.
Visible	نمایان بودن و نمایان نبودن کنترل را مشخص می‌کند.
Width	عرض کنترل براساس پیکسل را مشخص می‌کند.

مهم‌ترین خاصیت در جدول بالا خاصیت Name است. این مشخصه به شما اجازه می‌دهد که از آن نام برای ارجاع به کنترل در قسمت کد نویسی استفاده کنید. در این قسمت به بحث در مورد بعضی از مشخصه‌هایی که در بیشتر کنترل‌ها وجود دارند می‌پردازیم.

تغییر رنگ زمینه‌ی کنترل

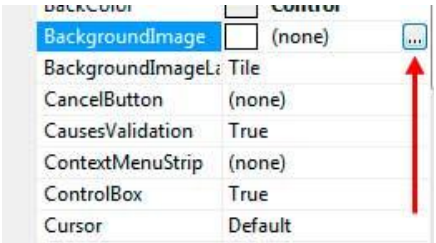
ما از مشخصه‌ی BackColor کنترل برای تغییر رنگ زمینه‌ی آن استفاده می‌کنیم. مشخصه‌ی BackColor را در پنجره‌ی Properties پیدا کنید و بر روی نوار کرکره‌ای کنار آن کلیک کنید. شما یک پنجره با سه سربرگ مشاهده می‌کنید. هر سربرگ حاوی رنگ‌های مختلفی است.



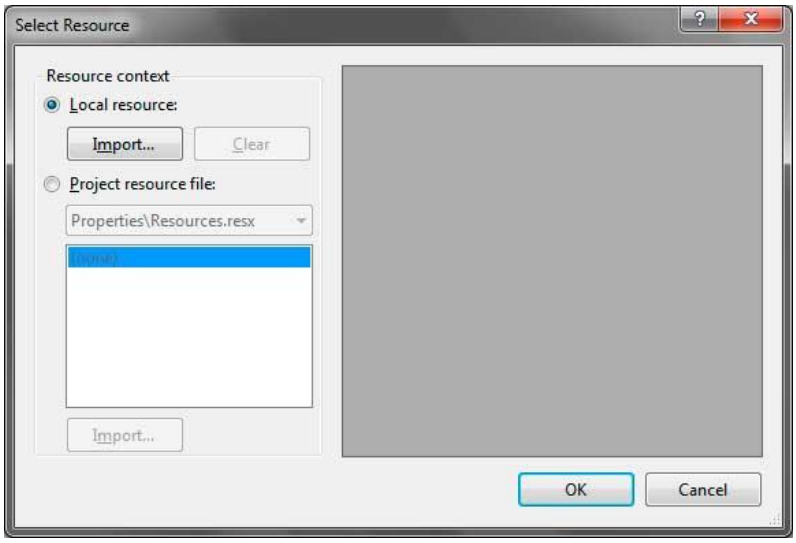
سربرگ System شامل رنگ‌هایی است که سیستم عامل به طور پیش فرض برای رنگ آمیزی کنترل‌ها از آنها استفاده می‌کند. سربرگ Web شامل رنگ‌هایی است که برای استفاده در صفحات وب مناسب هستند. سربرگ Custom شامل رنگ‌های بسیار زیادی است. در این سر برگ شما می‌توانید مقادیر RGB آن را در نوارها مربوطه وارد کنید.

اضافه کردن یک عکس به زمینه‌ی کنترل

شما می‌توانید عکس پس زمینه‌ی یک کنترل را به وسیله‌ی مشخصه‌ی BackgroundImage تغییر دهید. به عنوان مثال ما می‌خواهیم عکس زمینه‌ی یک فرم را تغییر دهیم. برای اینکار ابتدا فرم را انتخاب کرده و سپس به پنجره‌ی Properties رفته و مشخصه‌ی BackgroundImage را انتخاب می‌کنیم.



در این قسمت پنجره‌ای به ما نمایش داده می‌شود که به ما اجازه می‌دهد آدرس تصویر خود را مشخص کنیم. برای اینکار ابتدا قسمت Local resource را انتخاب کرده و سپس بر روی دکمه‌ی Import کلیک می‌کنیم.



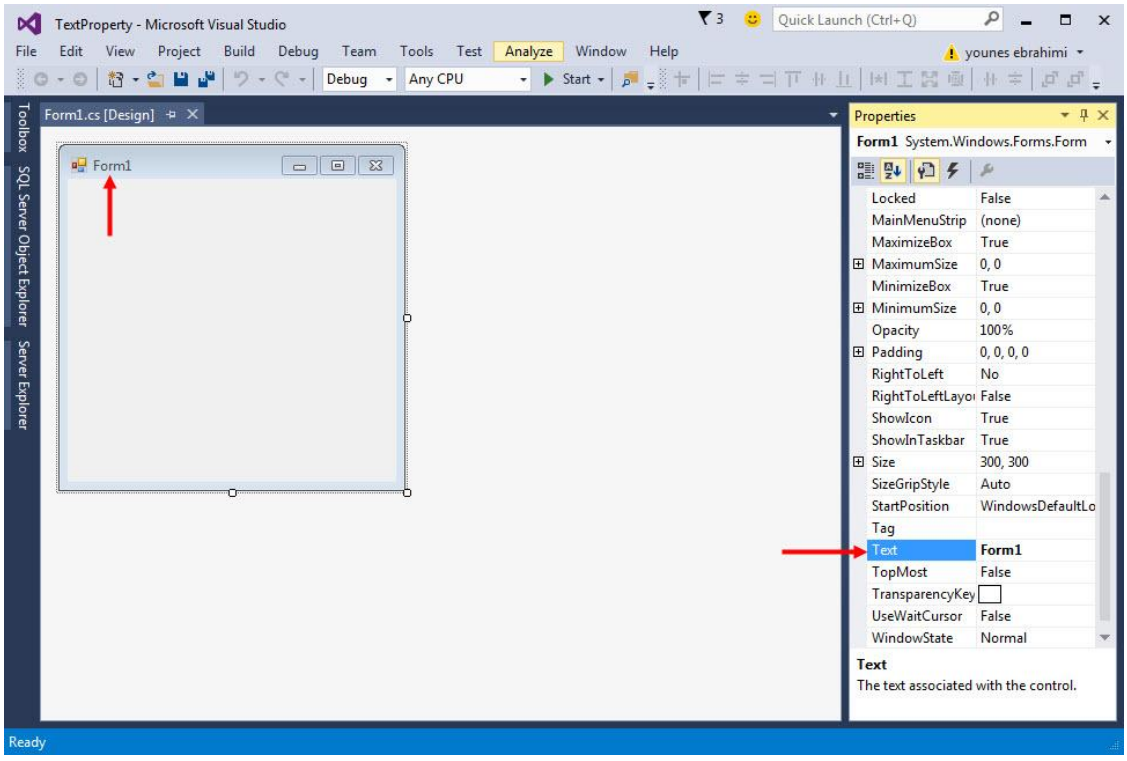
پس از اینکه عکس را انتخاب کردید دکمه Ok را برای تأیید بزنید. حالا تصویر مورد نظر ما بر روی فرم نمایش داده می‌شود. حالت قرارگیری و سایز تصویر ممکن است چیزی نباشد که شما انتظار داشته‌اید. برای تغییر آن یک مشخصه به نام BackgroundImageLayout وجود دارد. این مشخصه مقادیری از نوع شمارشی System.Windows.Forms.ImageLayout را می‌پذیرد. این مقادیر به شرح زیر هستند:

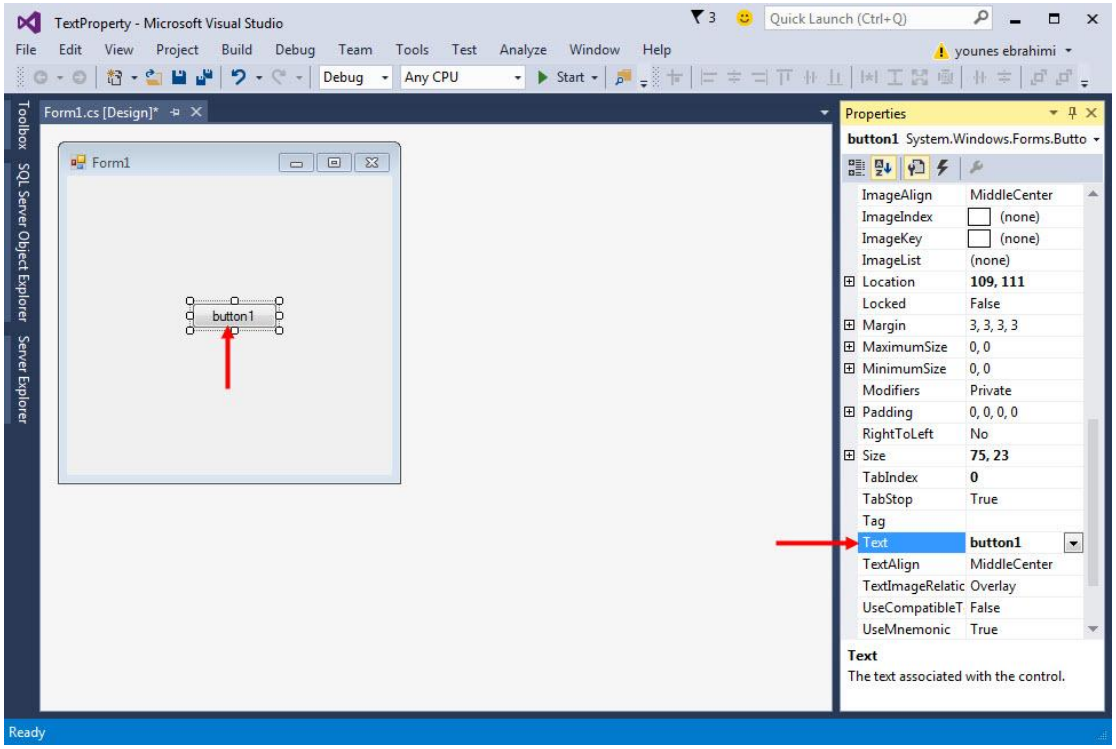
مقدار	توضیح
None	عکس بر اساس قسمت بالای چپ خود در محل قرار می‌گیرد و هیچ تغییر سایزی در آن صورت نمی‌گیرد.
Tile	اگر عکس از محیط کنترل کوچک‌تر باشد، عکس به طور متناوب بر روی کنترل تکرار می‌شود.
Center	عکس در وسط کنترل قرار می‌گیرد.
Stretch	عکس در حد اندازه کنترل کوچک می‌شود و کل کنترل را در بر می‌گیرد.
Zoom	عکس در حد کنترل تغییر سایز می‌دهد بدون اینکه نسبت طول و عرض آن تغییر کند.

در بیشتر مواقع Stretch به خوبی عمل می‌کند. در فرم پایین عکس پس زمینه‌ی فرم در حالت Stretch قرار دارد.

خاصیت Text

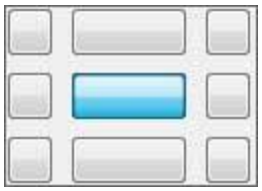
خاصیت Text یک متن را در داخل کنترل تعریف می‌کند. متن خاصیت Text، در کنترل‌های گوناگون نمایش مختلفی دارد. به عنوان مثال، متن خاصیت Text کنترل فرم در عنوان (Caption bar) آن قرار می‌گیرد. متن خاصیت Text یک دکمه، در داخل آن دکمه نمایش داده می‌شود. متن خاصیت Text یک جعبه متن، در داخل آن جعبه متن نمایش داده می‌شود:



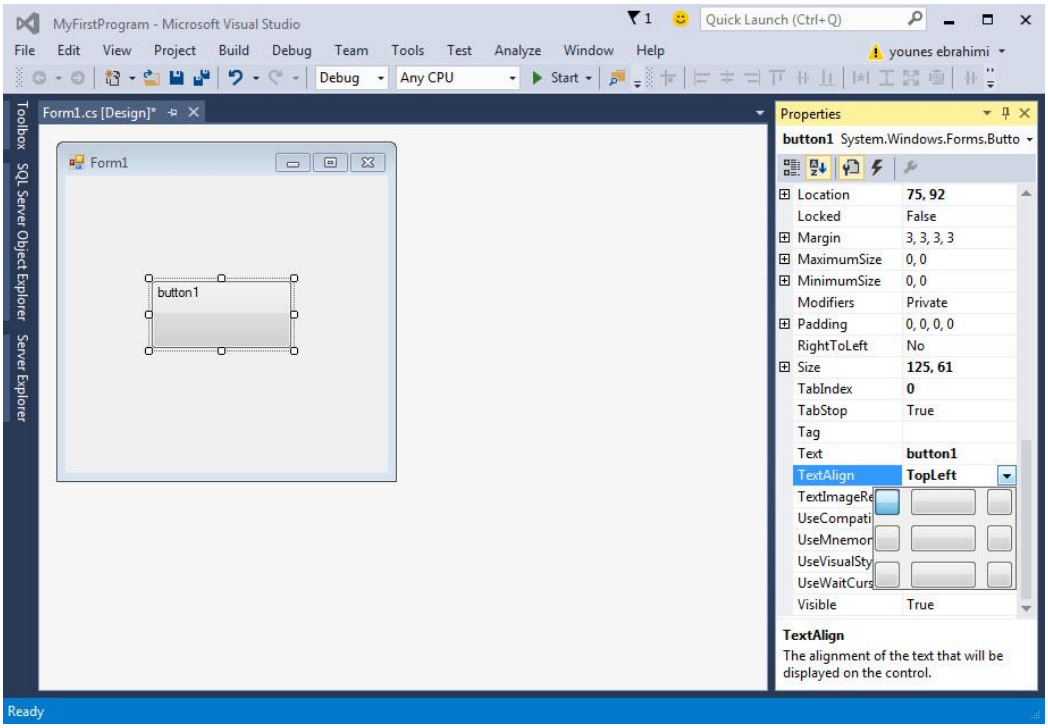


خاصیت TextAlign

ما می‌توانیم از خاصیت `TextAlign` برای تغییر مکان متن در داخل کنترل، استفاده کنیم. اگر شما بر روی مشخصه‌ی `TextAlign` کلیک کنید یک نوار کرکره ای را در پنجره `Properties` مشاهده می‌کنید، که به آسانی به شما اجازه می‌دهد که نحوه‌ی قرار گیری متن را در داخل کنترل، تنظیم کنید.

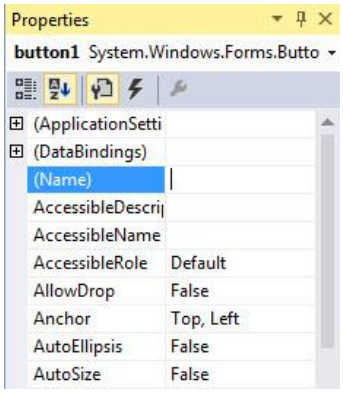


برای درک بهتر عملکرد این خاصیت، یک دکمه بر روی فرم قرار داده و اندازه آن را کمی بزرگ کنید. حال بر روی قسمت‌های مختلف خاصیت `TextAlign` کلیک کرده و تأثیر آن را بر نوشته داخل کنترل دکمه مشاهده کنید:

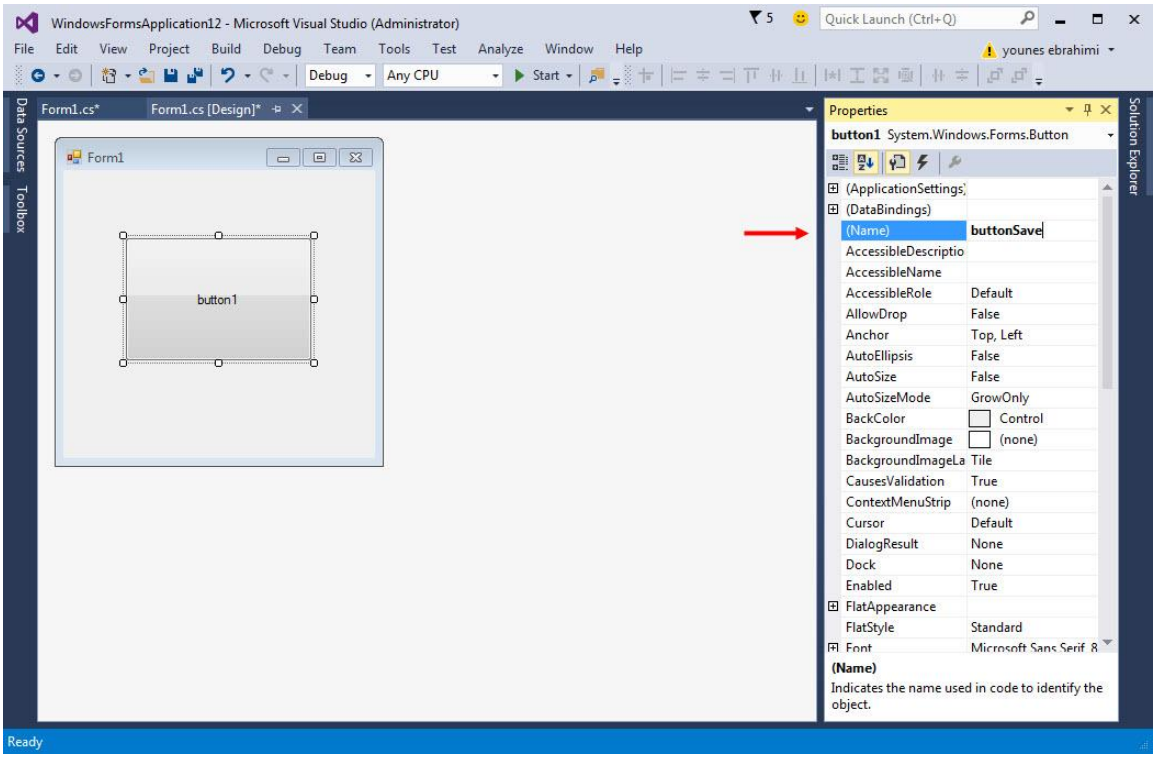


خاصیت Name

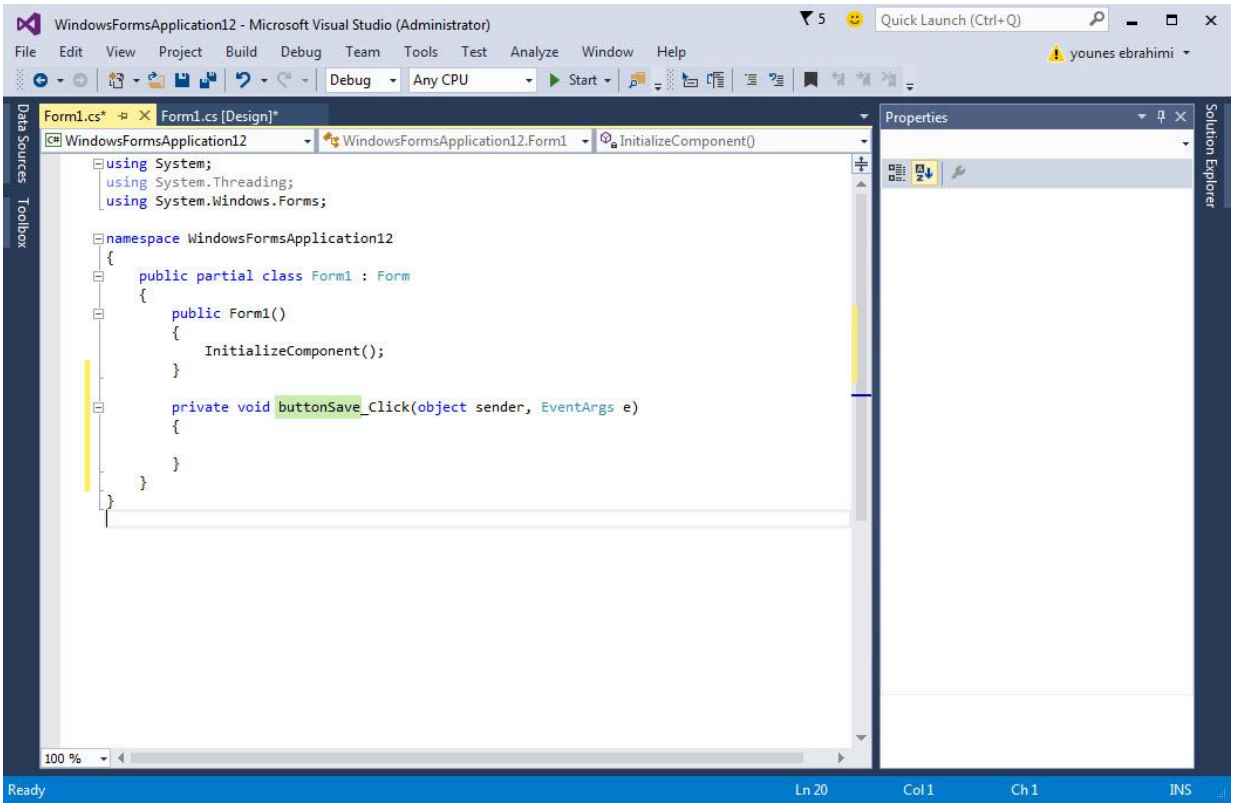
از خاصیت `Name` برای نامگذاری کنترل‌ها استفاده می‌شود:



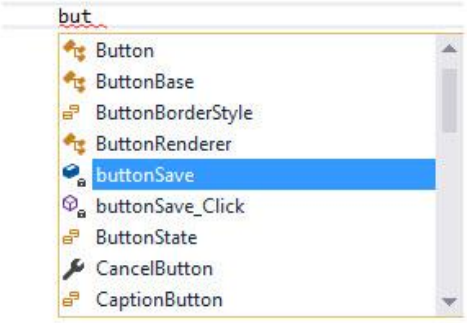
نامی که به کنترل اختصاص می‌دهیم در کد نویسی به درد ما می‌خورد. فرض کنید که یک دکمه قرار است عملیات ذخیره اطلاعات در دیتابیس را انجام دهد. برای این منظور خاصیت `Name` آن را به `buttonSave` تغییر می‌دهیم تا در محیط کدنویسی آن را راحت تر شناسایی کنیم:



حال اگر بر روی آن دوبار کلیک کنید، مشاهده می‌کنید که نام دکمه همان نامی است که در خاصیت Name به آن اختصاص داده‌ایم:



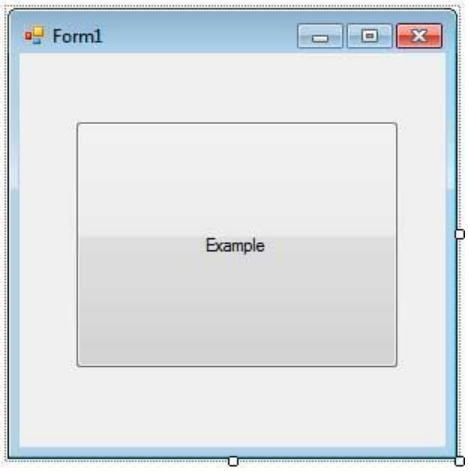
و اگر در محیط کدنویسی به دنبال آن بگردیم، پیدا کردن این کنترل بسیار راحت است:



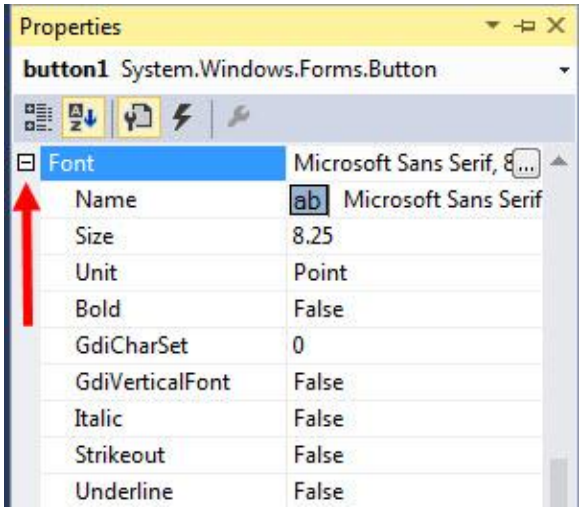
شاید در مورد یک یا چند مورد کنترل، نامگذاری زیاد کارا نباشد ولی وقتی که تعداد کنترل‌ها زیاد می‌شود بهتر است که همه آنها را نامگذاری کنید تا در موقع کدنویسی گیج نشوید.

تغییر فونت یک کنترل

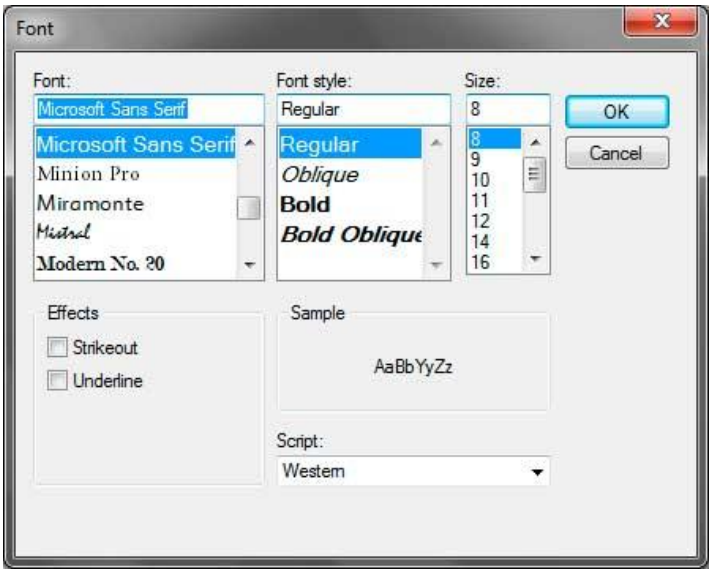
ما می‌توانیم نوع، رنگ، سایز، و حالت فونت یک کنترل را به وسیله‌ی خواص `ForeColor` و `Font` تغییر دهیم. به عنوان مثال یک دکمه بر روی فرم قرار می‌دهیم.



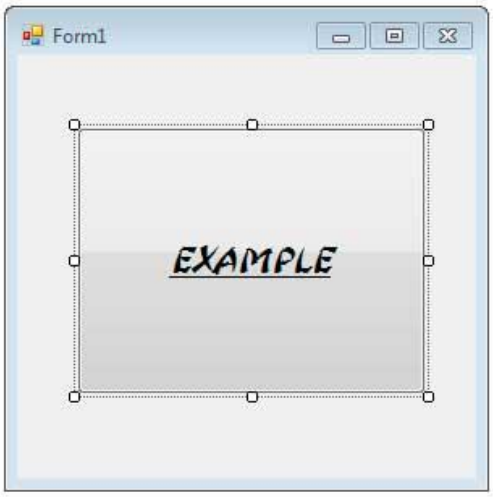
برای تغییر نوع فونت این کنترل، ابتدا می‌بایست خاصیت `Font` را در پنجره‌ی `Properties` پیدا کنیم. سپس در سمت چپ این خاصیت یک علامت فلش را مشاهده می‌کنیم. بر روی آن کلیک کرده تا تمامی زیر مجموعه‌های آنرا مشاهده کنیم.



قسمت Name که نوع فونت و قسمت Size اندازه‌ی فونت را مشخص می‌کند. قسمت Bold برای ضخیم کردن متن و قسمت‌های Italic و Strikeout برای کشیدن خط زیر متن کنترل به کار می‌روند. همچنین، شما می‌توانید از دکمه‌های سمت راست خاصیت Font، برای باز کردن پنجره‌ی فونت استفاده کنید.

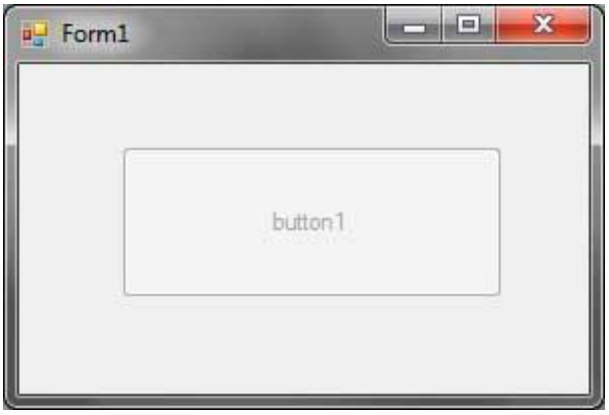


در اینجا شما می‌توانید نوع فونت، حالت و سایز آنرا به طور دقیق انتخاب کنید. همچنین در این پنجره شما می‌توانید یک پیش نمایش از نوع فونت خود را مشاهده کنید. به وسیله‌ی این‌ها شما می‌توانید فونت کنترل خود را سفارشی کنید.



فعال کردن و غیر فعال کردن یک کنترل

می‌توان از خاصیت Enabled برای فعال کردن یا غیر فعال کردن یک کنترل استفاده کنیم. از مقدار False برای غیر فعال کردن و از مقدار True برای فعال کردن یک کنترل استفاده می‌شود. وقتی یک کنترل را غیر فعال می‌شود، ظاهر آن تغییر می‌کند. رأی مثال، وقتی یک دکمه را غیر فعال می‌کنیم (خاصیت Enabled آنرا برابر False قرار می‌دهیم)، رنگ و ظاهر آن تغییر می‌کند. به این نکته توجه کنید که این تغییرات در هنگام اجرای برنامه قابل مشاهده هستند، نه در محیط طراحی.



وقتی که یک کنترل غیر فعال است، هیچ فوکوس و رویدادی را قبول نمی‌کند. برای مثال، اگر یک دکمه غیر فعال باشد، شما نمی‌توانید بر روی آن کلیک کنید.

مخفی کردن یک کنترل

شما می‌توانید یک کنترل را به وسیله‌ی اختصاص مقدار False به خاصیت Visible آن، به طور موقت مخفی کنید. خاصیت Visible یک خاصیت بولی است که دو مقدار True و False را قبول می‌کند. مقدار False آن، کنترل را مخفی می‌کند و مقدار True کنترل را نمایش می‌دهد. به این نکته توجه کنید که، اثر این خاصیت فقط در هنگام اجرای برنامه قابل مشاهده است و در حالت طراح قابل مشاهده نیست! شما حتی

وقتی که مقدار این خاصیت را برابر False قرار بدهید می‌توانید آنرا در حالت طراحی انتخاب کنید. در درس‌های آینده به طور مفصل در مورد خواص و ویژگی‌های کنترل‌ها به بحث می‌پردازیم.

رویدادهای کنترل

در جدول پایین شما برخی از رویدادهای مفید که بین بیشتر کنترل‌های رایج هستند را مشاهده می‌کنید.

رویداد	توضیحات
BackColorChanged	وقتی که رنگ پس زمینه تغییر می‌کند این رویداد اتفاق می‌افتد.
BackgroundImageChanged	وقتی که یک تصویر پس زمینه اضافه می‌شود یا تغییر می‌کند این رویداد رخ می‌دهد.
Click	این رویداد وقتی رخ می‌دهد که بر روی کنترل با دکمه‌ی سمت چپ موس کلیک شود.
DoubleClick	زمانی که شما بر روی یک کنترل دابل کلیک می‌کنید، رخ می‌دهد. به این نکته توجه داشته باشید که این رویداد می‌تواند به وسیله‌ی دستوراتی که کاربر نوشته نیز رخ دهد مثل ترکیب دو کلید (کلید میانبر).
DragDrop	زمانی که عملیات Drag & Drop بر روی کنترل انجام می‌شود، این رویداد رخ می‌دهد.
EnabledChanged	زمانی که یک کنترل حالت Enable و یا Disable را به خود می‌گیرد (به یکی از این دو حالت تغییر می‌کند) این رویداد رخ می‌دهد.
Enter	زمانی که مکان نما به کنترل وارد شود (بر روی آن قرار گیرد).
FontChanged	زمانی که خاصیت font کنترل تغییر می‌کند، این رویداد رخ می‌دهد.
ForeColorChanged	زمانی رخ می‌دهد که رنگ متن کنترل تغییر پیدا کند.
GotFocus	زمانی رخ می‌دهد که Focus روی کنترل قرار گیرد.
KeyDown	این رویداد زمانی رخ می‌دهد که focus بر روی کنترل باشد و کلیدی از صفحه کلید فشرده شود.
KeyPress	این رویداد زمانی رخ می‌دهد که focus بر روی کنترل باشد و کلیدی از روی صفحه کلید فشرده و رها شود. این رویداد قبل از رویداد Keydown رخ می‌دهد.
KeyUp	این رویداد زمانی رخ می‌دهد که focus بر روی کنترل باشد و کلیدی که از روی صفحه کلید فشرده شده، رها شود. این رویداد بعد از دو رویداد قبلی رخ می‌دهد.
Leave	این رویداد وقتی که مکان نما کنترل را ترک کند رخ می‌دهد.
LostFocus	وقتی که کنترل Focus خود را از دست بدهد این رویداد رخ می‌دهد.

MouseClick	یک حالت پیشرفته تر از رویداد کلیک است. کلیک کردن می‌تواند شامل فشردن کلیدهای صفحه‌ی کلید نیز باشد. ولی اگر شما نیاز به اطلاعاتی نظیر، تعداد کلیک بر روی کنترل، چرخیدن دکمه چرخنده‌ی ماوس و ... باشید، باید از این رویداد استفاده کنید
MouseDoubleClick	یک حالت پیشرفته تر از رویداد DoubleClick است.
MouseDown	این رویداد زمانی رخ می‌دهد که دکمه ای از ماوس در داخل کنترل پایین نگه داشته شود.
MouseEnter	زمانی که مکان نما به یک کنترل وارد می‌شود این رویداد رخ می‌دهد.
MouseHover	این رویداد زمانی رخ می‌دهد که مکان نما روی یک کنترل منتظر بماند. (حالت مکان نما به Wait یا همان ساعت شنی تغییر یابد.)
MouseLeave	این رویداد زمانی رخ می‌دهد که مکان نما کنترل را ترک کند.
MouseMove	این رویداد زمانی رخ می‌دهد که مکان نما وقتی در داخل محدوده‌ی کنترل است، حرکت کند.
MouseUp	این رویداد زمانی رخ می‌دهد که دکمه ای از ماوس را که قبلاً فشرده‌ایم، در داخل کنترل رها کنیم.
MouseWheel	این رویداد زمانی رخ می‌دهد که بر روی کنترل Focus باشد و دکمه چرخنده ماوس حرکت کند.
Move	این رویداد زمانی رخ می‌دهد که کنترل حرکت کند.
Paint	این رویداد زمانی رخ می‌دهد که کنترلی رسم شود.
ParentChanged	این رویداد زمانی رخ می‌دهد که خاصیت Parent کنترل تغییر کند.
Resize	این رویداد زمانی رخ می‌دهد که کنترل تغییر سایز دهد.
TextChanged	وقتی که خاصیت Text کنترل تغییر پیدا کند این رویداد رخ می‌دهد.
Validated	وقتی رخ می‌دهد که اعتبار سنجی کنترل پایان یابد.
Validating	وقتی رخ می‌دهد که کنترل اعتبار سنجی می‌شود.
VisibleChanged	وقتی که خاصیت نمایش کنترل تغییر پیدا کند.

در مورد بیشتر رویدادهای بالا در بخش‌های آینده صحبت خواهیم کرد.

متدهای کنترل

متدهای زیر پرکابردترین متدهای کلاس کنترل هستند.

متد	توضیح
CreateControl	یک کنترل را ایجاد می‌کند و به کنترل جاری اضافه می‌کند.
FindForm	فرمی را که کنترل بر روی آن قرار دارد را بازیابی می‌کند.
Focus	Focus را بر روی کنترل جاری قرار می‌دهد.
Hide	کنترل را مخفی می‌کند.
Refresh	نوسازی کنترل را انجام می‌دهد.
Select	کنترل را فعال (انتخاب) می‌کند.
Show	کنترل مخفی شده را نمایش می‌دهد.
Update	محتویات کنترل را به روز می‌کند.

همچنین کلاس کنترل به شما متدهایی را عرضه می‌کند که به شما اجازه می‌دهند به صورت دستی رویدادهای کنترل را راه اندازی کنید. برخی از متدها با کلمه‌ی On و سپس نام یک رویداد شروع می‌شوند. برای مثال، رویداد OnClick وقتی که رویداد Click فراخوانی می‌شود اتفاق می‌افتد.

نامگذاری کنترل‌ها

همیشه کنترل‌ها را نامگذاری کنید. ما از خاصیت Name برای نامگذاری کنترل‌ها استفاده می‌کنیم. نامگذاری کنترل‌ها از قواعد نامگذاری متغیرها استفاده می‌کند، برای مثال فاصله‌ها، کاراکترهای ویژه، و استفاده از کلمات کلیدی در نامگذاری کنترل‌ها ممنوع است. یک سری قرارداد در نامگذاری کنترل‌ها وجود دارد که در ادامه به آنها اشاره می‌کنیم.

شما می‌توانید بسته به نوع استفاده‌ای که از کنترل می‌کنید آنرا نامگذاری نمایید. برای مثال، وقتی که قرار است در داخل یک جعبه متن نام یک شخص یا کاربر وارد شود، می‌توانید نام آن جعبه متن را برای خوانایی بیشتر firstname قرار دهید. اما در نامگذاری کنترل‌ها بهتر است که نام واقعی آن کنترل قبل از نامی که برای آن انتخاب کرده‌ایم بیاید. برای مثال به جای نامگذاری جعبه متن به firstname، می‌توانید از TextBoxFirstName استفاده کنید. با توجه به این قرار دادها، ما می‌توانیم در داخل Intellisense ی که با آن کار می‌کنیم هر کنترلی را شناسایی کنیم.

تکنیک دیگری که برای نامگذاری کنترل‌ها به کار می‌رود، خلاصه کردن نام آنها است. برای مثال، به جای استفاده از کلمه‌ی TextBoxFirstName، شما می‌توانید از txtFirstName استفاده کنید. کلمه‌ی txt فرم کوتاه شده‌ی کلمه‌ی textbox است. تعداد زیادی کلمه‌ی کوتاه شده برای هر کنترل وجود دارد و شما حتی می‌توانید از کلمه‌ی کوتاه شده‌ای که خود برای آن کنترل انتخاب کرده‌اید استفاده کنید. البته کلمه‌ی کوتاه شده باید گویا باشد. به طوری که برای کسانی که به کد شما نگاه می‌کنند واضح باشد.

تکنیک دیگری که برای نامگذاری کنترل‌ها بکار می‌رود، جابجا کردن نام اصلی کنترل با نامی که ما برای آن انتخاب کرده‌ایم و همچنین استفاده از روش کوهان شتری (Camel Casing) است. برای مثال یک جعبه متن داریم که نام کاربر یا شخصی را از ورودی می‌گیرد، می‌توانید آنرا به صورت firstNameTextBox نامگذاری کنید و یا یک دکمه دارید که عملیات محاسباتی را انجام می‌دهد، می‌توانید نام آنرا calculateButton بگذارید. در درس‌های آینده از تکنیک سوم (Camel Casing) استفاده می‌کنیم.

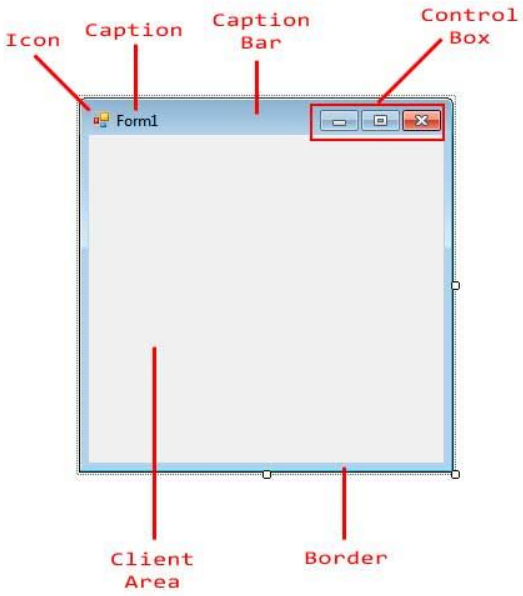
نیازی به این نیست که شما تمامی کلمات اختصاری کنترل‌ها را حفظ کنید. وقتی که شما یک کنترل را ایجاد می‌کنید، تنها کافیست عدد پسوندی آنرا حذف کنید و به جای آن، نامی را با توجه به کاربرد کنترل به آن اضافه کنید. وقتی که شما در حال تایپ کردن هستید و قصد پیدا کردن کنترل خاصی را دارید کافیست براحتی نام کنترل را تایپ کرده تا پنجره‌ی intellisence نام آن کنترل و تمامی کنترل‌های مشابه آنرا را برای شما نمایش دهد. برای مثال، یک جعبه متن و تمامی کنترل‌های جعبه متن در intellisence برنامه نمایش داده می‌شوند، فقط قسمتی که مربوط به نامی که شما برای آن کنترل انتخاب می‌کنید ممکن است طولانی باشد، که شما با توجه به قواعدی که پیش‌تر ذکر شد می‌توانید نسبت به نامگذاری و کوتاه کردن نام کنترل‌ها اقدام کنید. در زیر مثال‌هایی از نامگذاری کنترل‌ها بسته به کاربرد آنها ذکر شده است:

نام پیشنهادی	کاربرد
buttonConfirm, confirmButton, btnConfirm	کلیدی که برای نشان دادن یک پیغام به کار می‌رود.
textBoxAddress, addressTextBox, txtAddress	یک جعبه متن که یک ایمیل را از کاربر دریافت می‌کند.
formPersonalInformation, personalInformationForm, frmPersonalInformation	یک فرم که اطلاعات شخصی را از کاربر دریافت می‌کند.
comboBoxProducts, productsComboBox, cmbProducts	یک ComboBox که لیستی از محصولات را نمایش می‌دهد.
radioButtonMale, maleRadioButton, radMale	یک RadioButton که نشان دهنده‌ی مذکر بودن یک کاربر است.
menuItemSave, saveMenuItem, mnuSave	یک MenuItem برای ذخیره کردن یک فایل.
checkBoxSubscribe, subscribeCheckBox, chkSubscribe	یک CheckBox برای فرستادن یک نامه.

لازم نیست که تمامی کنترل‌های یک فرم را نام گذاری کنید. کنترل‌هایی که هیچوقت در کدنویسی مورد استفاده قرار نمی‌گیرند، می‌توانند از نام پیش فرض خود استفاده کنند. برای مثال، اکثر Lableها صرفاً برای نامگذاری بکار می‌روند و در کد نویسی مورد استفاده قرار نمی‌گیرند. لذا لازم نیست خاصیت Name آنها را تغییر دهیم.

ویندوز فرم

ویندوز فرم‌ها (یا همان فرم‌ها)، پنجره‌هایی هستند که شما در برنامه‌های تحت ویندوز می‌بینید. شما می‌توانید در یک برنامه بیش از یک فرم داشته باشید. هر فرم از خواص و متدهای کلاس System.Windows.Forms.Form ارث می‌برد. فضای نام System.Windows.Forms.Form تمامی اجزایی را که شما برای ساخت فرم‌ها و کنترل‌ها نیاز دارید را در بر دارد. در شکل زیر شما نام قسمت‌های مختلف فرم را مشاهده می‌کنید.



در قسمت بالا، نوار عنوان (Caption Bar) قرار دارد. نوار عنوان از یک آیکن (Icon)، یک عنوان (Caption)، و یک جعبه‌ی کنترل (Control Box) تشکیل شده است. جعبه‌ی کنترل (Control Box)، دکمه‌های کوچکنمایی (Minimizing)، بزرگنمایی (Maxmizing)، و بستن (Closing) را در بر دارد. قسمت داخلی یا همان Client Area مکانی است که ما کنترل‌ها را در آن قرار می‌دهیم. قسمت حاشیه یا Border که شامل قسمت Caption Bar نیز می‌شود، به شما اجازه می‌دهد که سایز فرم را تغییر دهید. در جدول زیر برخی از خواص مفید کلاس فرم را مشاهده می‌کنید.

خاصیت	توضیح
AcceptButton	نام یک دکمه در این خصوصیت تعیین می‌شود و هرگاه دکمه‌ی Enter فشار داده شود، آن دکمه عمل خواهد کرد.
CancelButton	در این خصوصیت نیز مانند خصوصیت قبل نام یک دکمه در آن قرار می‌گیرد و با زدن دکمه Esc آن دکمه عمل می‌کند.
ControlBox	این خصوصیت مشخص می‌کند که قسمت جعبه‌ی کنترل (Control Box) نمایش داده شود یا خیر. جعبه‌ی کنترل (Control Box) قسمتی از فرم است که دکمه‌های کوچکنمایی، بزرگنمایی و بستن فرم در آن قرار می‌گیرند.
DesktopBounds	به وسیله‌ی این خاصیت می‌توانید به سایز و محل قرارگیری فرم در میز کار دسترسی داشته باشید.

Font	نوع قلمی که در فرم مورد استفاده قرار می‌گیرد. تمامی کنترل‌ها از این خاصیت استفاده می‌کند.
FormBorderStyle	این خاصیت نوع حاشیه‌ی فرم را مشخص می‌کند.
HelpButton	یک دکمه‌ی Help قبل از دکمه‌ی Close بالای فرم قرار می‌دهد (دکمه‌های Minimize و Maximize غیر فعال می‌شوند).
Icon	این خاصیت Icon فرم را مشخص می‌کند (آیکنی که در قسمت Caption Bar وجود دارد).
Location	مختصات فرم را در صفحه‌ی نمایش مشخص می‌کند.
MainMenuStrip	یک منو را مشخص می‌کند تا در فرم به عنوان منوی اصلی استفاده شود.
MaximizeBox	مشخص می‌کند که دکمه‌ی Maximize در بالای فرم نمایش داده شود یا خیر.
MinimizeBox	مشخص می‌کند که دکمه‌ی Maximize در بالای فرم نمایش داده شود یا خیر.
Name	نام فرم که به عنوان مرجع برای استفاده در محیط کد نویسی انتخاب می‌شود.
ShowIcon	مشخص می‌کند که آیکن فرم نمایش داده شود یا خیر.
Size	سایز فرم را مشخص می‌کند.
StartPosition	این خاصیت محل نمایش فرم در صفحه را مشخص می‌کند.
Text	این خاصیت متنی را که در قسمت Caption Bar قرار می‌گیرد را مشخص می‌کند.

در این قسمت برخی از متدهای مهم کلاس فرم را مشاهده می‌کنید.

متد	توضیح
Activate	Focus را به فرم منتقل و فرم را فعال می‌کند.
CenterToScreen	فرم را به وسط صفحه منتقل می‌کند.
Close	فرم را می‌بندد.
Hide	فرم جاری را پنهان می‌کند.
OnLoad	وقتی که رویداد Load رخ می‌دهد، این متد اجرا می‌شود.
Show	فرم را نمایش می‌دهد (آشکار می‌سازد).

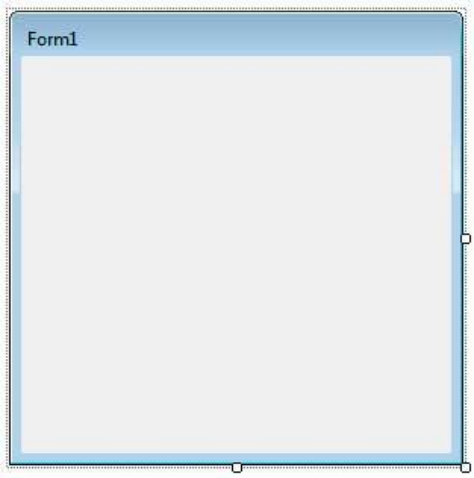
در این قسمت رویدادهای مربوط به فرم را مشاهده می‌کنید.

رویداد	توضیح
Activated	زمانی رخ می‌دهد که فرم فعال باشد.
Click	زمانی رخ می‌دهد که بر روی فرم کلیک می‌شود.
Deactivated	این رویداد زمانی رخ می‌دهد که فرم حالت Focus خود را از دست بدهد.
FormClosed	زمانی رخ می‌دهد که فرم بسته شود.
FormClosing	زمانی رخ می‌دهد که فرم در حال بسته شدن است. به شما اجازه می‌دهد برای بسته شدن فرم مکث کنید.
HelpButtonClicked	زمانی رخ می‌دهد که بر روی دکمه‌ی Help کلیک شود.
KeyPress	این رویداد زمانی رخ می‌دهد که یک کلید بر روی صفحه کلید فشرده شود.
Load	زمانی رخ می‌دهد که بارگذاری فرم تمام شده است ولی هنوز نمایش داده نشده است.
MenuComplete	زمانی که یک منو از فرم حالت Focus خود را از دست می‌دهد.
MenuStart	زمانی که بر روی یک منو Focus می‌شود.
ResizeBegin	زمانی رخ می‌دهد که فرم شروع به تغییر سایز می‌کند.
ResizeEnd	زمانی رخ می‌دهد که تغییر سایز فرم تمام می‌شود.
Shown	زمانی رخ می‌دهد که فرم برای اولین بار نمایش داده می‌شود.

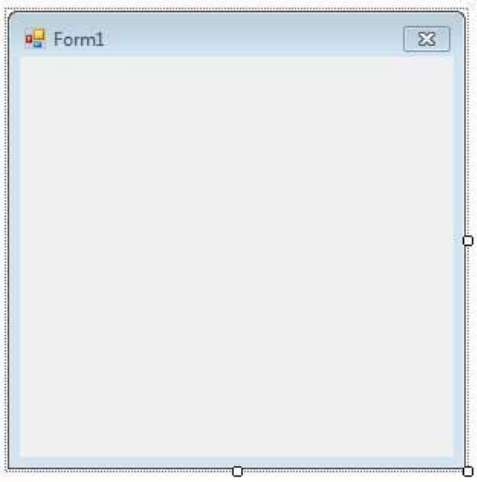
کلاس Form، فرزند کلاس اصلی System.Windows.Forms.Control است، بنابراین متدها و خواص تعریف شده‌ی این کلاس توسط کلاس Form ارث بری می‌شوند.

تغییر دادن جعبه‌ی کنترل (Control Box)

از خاصیت Control Box برای پنهان کردن و یا نمایش دادن قسمت جعبه‌ی کنترل (Control Box) استفاده می‌شود. این خاصیت وقتی که شما می‌خواهید دکمه‌های Minimize و Maximize کنترل را غیر فعال کنید و یا از طریق کدنویسی فرم را ببنید، بسیار کارآمد می‌شود. در تصویر زیر مقدار خاصیت ControlBox برابر False قرار گرفته شده است.



اگر شما می‌خواهید یکی از دکمه‌های Minimize یا Maximize را غیر فعال کنید، باید از خواص MinimizeBox و MaximizeBox استفاده کنید و مقدار آنها را به False تغییر دهید.



در فرم بالا دکمه‌های Maximize و Minimize مخفی شده‌اند. متاسفانه شما نمی‌توانید فقط دکمه Close را غیر فعال کنید.

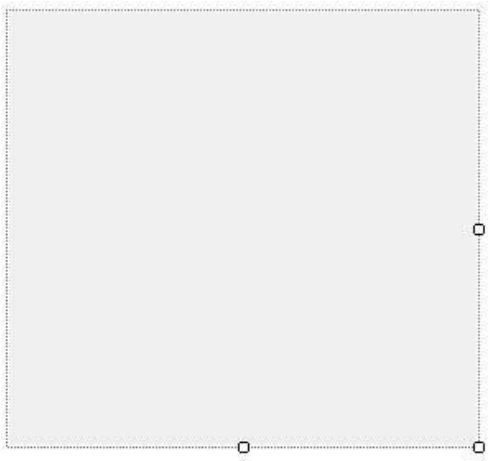
تغییر حالت حاشیه‌ی فرم

شما می‌توانید حالت حاشیه‌ی فرم را تغییر دهید. برای مثال، شما می‌خواهید که کاربر قادر به تغییر سایز فرم نباشد، در صورتیکه در حالت پیش‌فرض حاشیه‌ی فرم به کاربر اجازه‌ی اینکار را می‌دهد. می‌توانیم مقادیر مختلفی از نوع شمارشی System.Windows.Forms.FormBorderStyle را به خاصیت FormBorderStyle بدهیم.

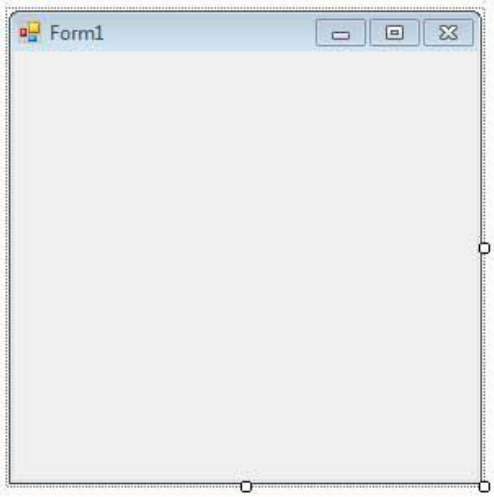
مقدار	توضیح
None	فرم هیچ حاشی‌های ندارد.
FixedSingle	فرم یک حاشیه‌ی خطی بدون قابلیت تغییر سایز را دارد.

Fixed3D	فرم یک حاشیه‌ی سه بعدی بدون قابلیت تغییر سایز را دارد.
FixedDialog	در این حالت یک خط ضخیم دور فرم قرار دارد و سایز فرم قابل تغییر نیست، همچنین دکمه‌های minimize و Maximize در آن وجود ندارند.
Sizable	حالت پیش‌فرض است. در این حالت سایز فرم قابل تغییر است.
FixedToolWindow	فرم دارای حاشیه ای است، که سایز آن غیر قابل تغییر می‌باشد و فقط دارای دکمه‌ی Close است. این حالت برای پنجره‌های ابزار بکار می‌رود. مانند پنجره‌ی ابزار در Office 2003
SizableToolWindow	دقیقاً مانند FixedToolWindow است ولی قابلیت تغییر سایز دارد.

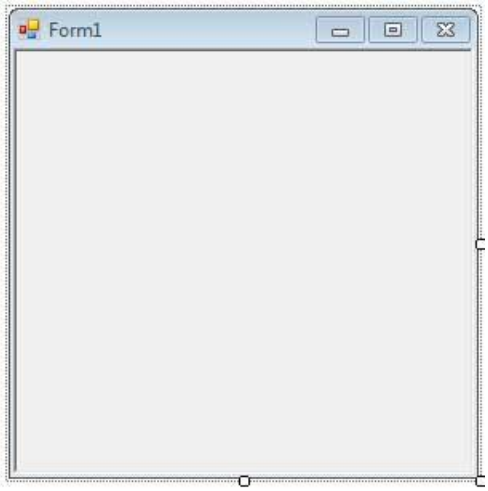
تصاویر زیر مربوط به حالات مختلف خاصیت FormBorderStyle است.



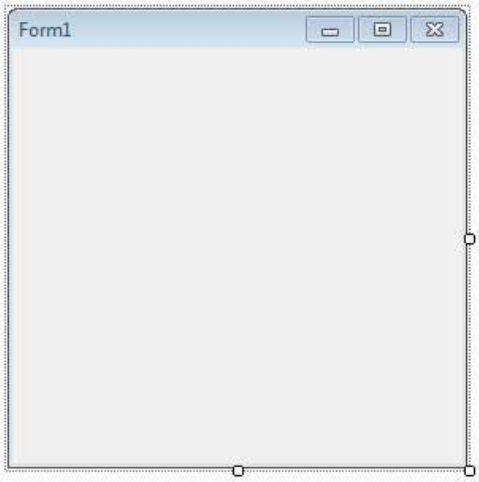
None



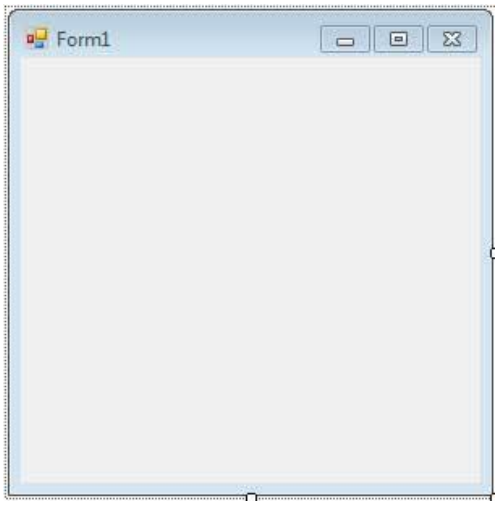
FixedSingle



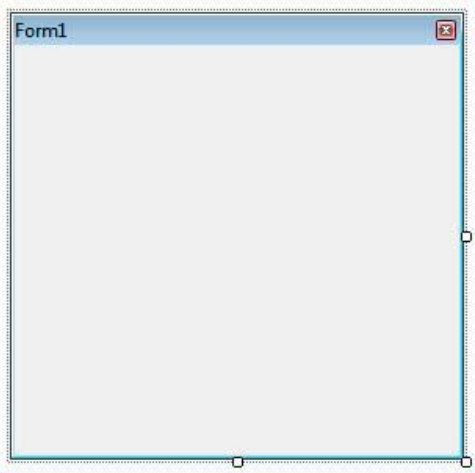
Fixed3D



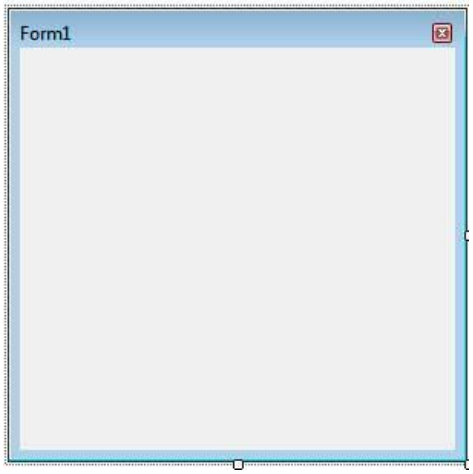
FixedDialog



Sizable



FixedToolWindow



SizableToolWindow

آیکن‌های فرم

از خاصیت Icon برای تغییر آیکن بالای فرم استفاده می‌شود. بر روی دکمه‌ی Browse در کنار خاصیت Icon در پنجره‌ی Properties کلیک و سپس فایلی را با پسوند ico. انتخاب کنید (ico. پسوند فایل‌های تصویری آیکن است). خاصیت ShowIcon به شما اجازه می‌دهد که آیکن را در نوارعنوان فرم (caption bar) پنهان یا آشکار کنید.

دکمه‌های Accept و Cancel

شما می‌توانید یک دکمه را به فرم اضافه کنید که یکی از دو نوع Accept یا Cancel باشد. برای این کار از خواص AcceptButton و CancelButton استفاده می‌شود. اگر دکمه‌ی شما از نوع Accept باشد، زمانی که فرم فعال باشد و کاربر دکمه‌ی Enter را فشار دهد آنگاه رویداد Click دکمه رخ می‌دهد. دکمه‌ی Cancel زمانی رخ می‌دهد که دکمه‌ی Escape بر روی صفحه کلید فشرده شود.

فقط به پنجره Properties رفته، خاصیت مورد نظر را پیدا کنید و بر روی نوار کرکره‌ای آن کلیک کنید تا نام تمامی دکمه‌های فرم برای شما نمایش داده شود. دکمه مورد نظر را انتخاب کنید. برای مثال، شما می‌خواهید یک فرم ورود (login) را ایجاد کنید. می‌توانید دکمه‌ی ورود را برای خاصیت AcceptButton انتخاب کنید. در اینصورت به آسانی، وقتی کاربر پسورد خود را وارد کرد، برای ورود می‌تواند کلید Enter را بفشارد. برای این خاصیت موارد زیادی وجود دارد، که در درس‌های آینده به آنها می‌پردازیم.

کنترل Button

وقتی که بر روی کنترل Button کلیک می‌کنیم دستوراتی اجرا می‌شود، و این معمول‌ترین استفاده‌ای است که از این کنترل می‌شود. دکمه‌ها به نوعی برای تأیید یا لغو یک عمل و یا باز کردن کادرهای محاوره‌ای به کار می‌روند. کنترل Button دارای خواصی است که در جدول زیر تعدادی از این آنها ذکر شده است:

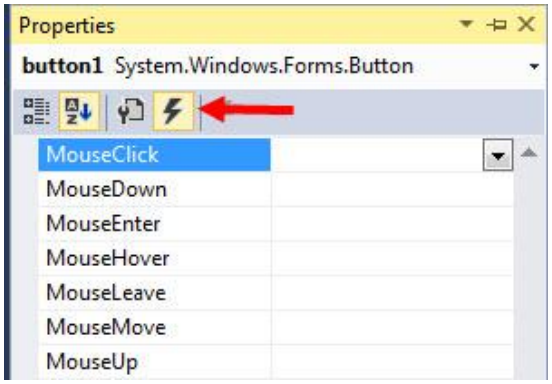
خواص	توضیح
AutoEllipsis	اگر طول یک متن برای Button انتخاب می‌کنیم زیاد باشد، تمام متن نمایش داده نشده و در انتهای آن سه نقطه (...) قرار داده می‌شود
AutoSize	اندازه کنترل را با متن داخل آن متناسب می‌کند
FlatStyle	برای تعیین شکل button به کار می‌رود. Popup باعث صاف شدن کنترل می‌شود. در این حالت اگر با ماوس بر روی کنترل توقف کنید، کنترل به حالت برجسته در می‌آید. Flat باعث صاف شدن کنترل می‌شود. در این حالت اگر نشانگر ماوس را بر روی کنترل ببریم، رنگ پس زمیه آن تغییر می‌کند.
Enabled	اگر مقدار این خاصیت false باشد بر روی button نمی‌توان کلیک کرد.
Image	می‌توان یک تصویر اختیاری به کنترل اختصاص داد.
ImageAlign	برای تراز کردن عکس اختصاص داده شده به button به کار می‌رود.
Text	عنوانی است که به کنترل اختصاص داده می‌شود.
Visible	مشخص می‌کند که آیا کنترل button در روی فرم قابل رویت باشد یا نه

فقط با ویرایش خواص یک button نمی‌توان از آن استفاده کرد. بلکه کنترل دکمه برای انجام برخی اعمال نیاز به واکنش به رویدادها دارد. در زیر معمول‌ترین رویدادهای کنترل button آمده است.

رویداد	توضیح

Click	وقتی روی می‌دهد که بر روی دکمه کلیک کنید.
Enter	وقتی روی می‌دهد که کنترل، کنترل فعال روی فرم باشد.
Leave	وقتی روی می‌دهد که ماوس کنترل را ترک کند.
LocationChanged	وقتی روی می‌دهد که مکان دکمه تغییر کند.
MouseDown	وقتی روی می‌دهد که نشانگر ماوس بر روی کنترل باشد و دکمه موس رو به پایین فشار داده شود.
MouseEnter	وقتی روی می‌دهد که ماوس وارد کنترل دکمه می‌شود.
MouseHover	وقتی روی می‌دهد که ماوس بر روی کنترل دکمه برای لحظاتی توقف کند.
MouseUp	وقتی روی می‌دهد که دکمه ماوس فشار داده شده و رها شود.
MouseLeave	وقتی روی می‌دهد که نشانگر ماوس دکمه را ترک می‌کند.

همانطور که قبلاً مشاهده کردید، رویداد پیشفرض کنترل دکمه رویداد Click می‌باشد. اجازه دهید که با ایجاد برنامه‌هایی با رویدادهای دیگر نیز آشنا شویم. یک فرم جدید ایجاد کرده و یک کنترل button را از toolbox به داخل آن درگ کنید. لازم به تغییر خاصیت text کنترل دکمه نمی‌باشد. در پنجره properties، خاصیت Name دکمه را یافته و آن را به buttonSample تغییر دهید. حال در هنگام کدنویسی می‌توانیم با استفاده از این نام به این کنترل مراجعه کنیم. برنامه ما رویدادهای MouseEnter و MouseLeave را شرح می‌دهد. برای دستیابی به این رویدادها کافیست که از پنجره properties بر روی دکمه ای که به صورت جرقه است، کلیک کنید.



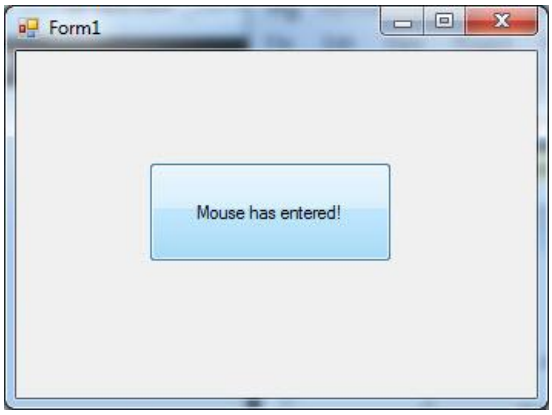
ابتدا رویداد MouseEnter را یافته و بر روی آن دوبار کلیک کنید. ویژوال استودیو یک کنترل کننده رویداد برای این رویداد ایجاد می‌کند. کد زیر را در داخل رویداد بنویسید.

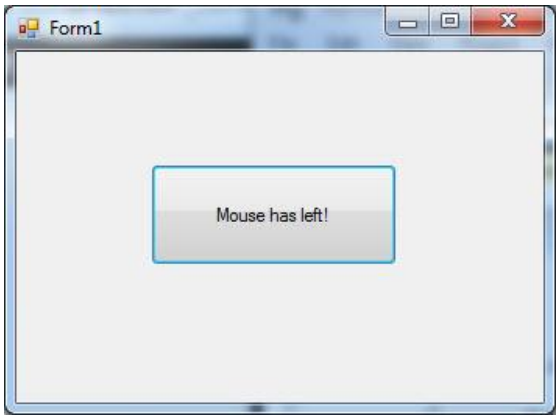
```
private void buttonSample_MouseEnter(object sender, EventArgs e)
{
   buttonSample.Text = "Mouse has entered!";
}
```

بعد از تایپ کد فوق به پنجره Properties برگشته و حال رویداد MouseLeave را انتخاب و بر روی آن دوبار کلیک کنید تا یک کنترل کننده رویداد نیز برای آن ایجاد شود. کد زیر را نیز داخل متد کنترل کننده رویداد بنویسید.

```
private void buttonSample_MouseLeave(object sender, EventArgs e)
{
   btnSample.Text = "Mouse has left!";
}
```

حال برنامه را اجرا کنید. مشاهده می‌کنید که با قرار گرفت اشاره گر ماوس بر روی کنترل button خاصیت text آن تغییر می‌کند. و از طرف دیگر وقتی که اشاره گر از کنترل دکمه دور می‌شود این خاصیت دوباره تغییر می‌کند.





کنترل ErrorProvider

کنترل ErrorProvider در مورد اشتباهات به کاربر هشدار می‌دهد. این اخطار ممکن است به خاطر خالی گذاشتن یک جعبه متن و یا اشتباه وارد کردن یک آدرس ایمیل باشد. در جداول زیر متدها، خاصیت‌ها و رویدادهای مهم این کنترل ذکر شده‌اند:

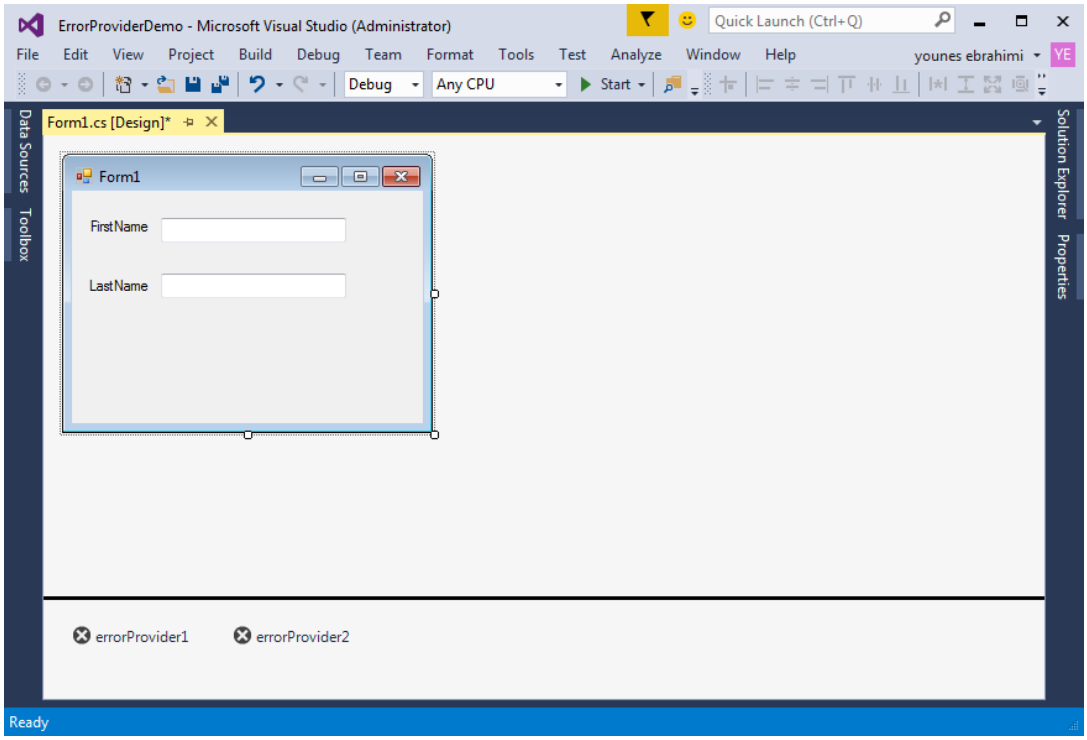
خاصیت	توضیح
BlinkRate	سرعت چشمک زدن آیکون ErrorProvider را مشخص می‌کند.

BlinkStyle	نحوه نمایش آیکون ErrorProvider را مشخص می‌کند. اینکه آیا این آیکون همیشه در حال چشمک زدن باشد، یا در صورت خطا نمایش داده شود و یا اصلاً نمایش داده نشود.
ContainerControl	کنترل والد، کنترل ErrorProvider را مشخص می‌کند.
Icon	آیکونی را که هنگام وقوع خطا باید نمایش داده شود را مشخص می‌کند. این آیکون زمانی نمایش داده می‌شود که برای خطا متنی مشخص کرده باشیم.
RightToLeft	برای راست به چپ کردن متن خطا به کار می‌رود.

متد	توضیح
GetError()	پیغام خطایی که برای کنترل مشخص کرده‌ایم را بر می‌گرداند.
GetIconAlignment()	محل قرار گیری آیکون خطا بسته به محل کنترل را بر می‌گرداند.
GetIconPadding()	فاصله آیکون خطا نسبت به کنترل والد را بر می‌گرداند.
SetError()	متن خطایی که در صورت بروز خطا قرار است به کاربر نمایش داده شود را مشخص می‌کند.
SetIconAlignment()	محل قرار گیری آیکون خطا نسبت به کنترل را مشخص می‌کند.
SetIconPadding()	فاصله بین آیکون خطا و کنترل مورد نظر را مشخص می‌کند.

رویداد	توضیح
RightToLeftChanged	وقتی روی می‌دهد که مقدار خاصیت RightToLeft تغییر کند.

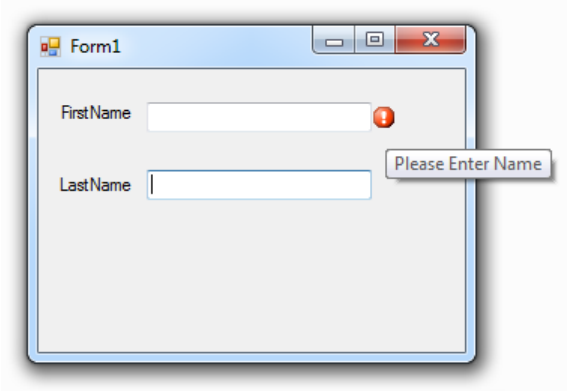
برای آشنایی با روش کار این کنترل، یک برنامه ویندوزی مانند شکل زیر ایجاد کنید و سپس دو جعبه متن و دو کنترل ErrorProvider بر روی آن قرار دهید:



لازم است که از رویدادهای اعتبارسنجی (validating) برای هر دو جعبه متن استفاده کنیم. بر روی جعبه‌های متن یکبار کلیک کرده و سپس از پنجره Properties به قسمت رویدادها رفته و بر روی رویداد validating دو بار کلیک کنید. کد زیر را در کنترل کننده رویداد validating جعبه متن FirstName ایجاد شده بنویسید:

```
private void textBox1_Validating(object sender, CancelEventArgs e)
{
    if (textBox1.Text == string.Empty)
    {
        errorProvider1.SetError(textBox1, "Please Enter Name");
    }
}
```

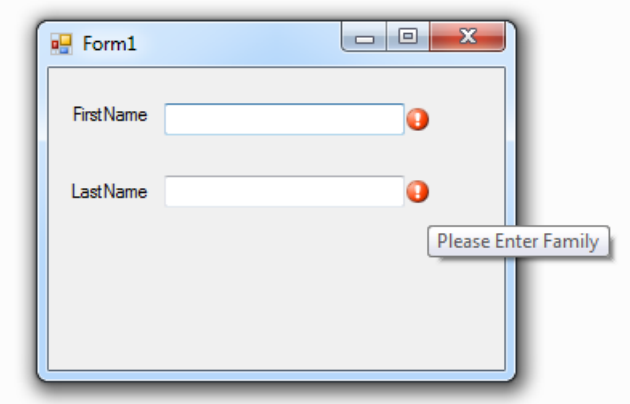
در کد بالا اعلام کرده‌ایم که اگر TextBox اول خالی بود، کنترل errorProvider1 ظاهر شده و به کاربر هشدار دهد. در کد بالا ما از متد SetError() برای این کار استفاده کرده‌ایم. این متد دو آرگومان قبول می‌کند، که آرگومان اول نام کنترلی است که errorProvider به آن می‌چسبد و آرگومان دوم پیغام هشداری است که با مکث بر روی آیکون errorProvider به شما نشان داده می‌شود. برای مشاهده عملکرد این کنترل برنامه را اجرا کرده و بدون اینکه در TextBox اول چیزی بنویسید، بر روی TextBox دوم کلیک کنید:



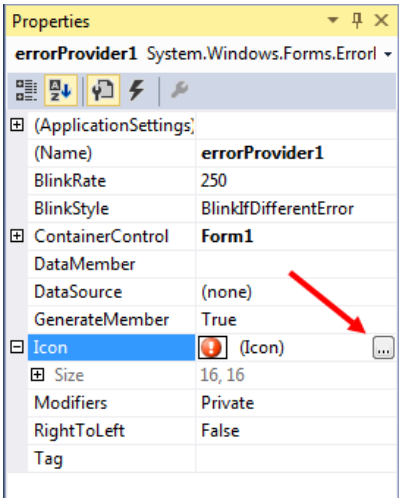
همانطور که در شکل بالا مشاهده می‌کنید با کلیک بر روی TextBox دوم کنترل errorProvider1 ظاهر شده و با مکث بر روی آیکون قرمز رنگ، پیام هشداری که در متد SetError() مشخص کرده‌اید به شما نمایش داده می‌شود. برای TextBox دوم هم به روشی مشابه می‌توان کد زیر را نوشت:

```
private void textBox2_Validating(object sender, CancelEventArgs e)
{
    if (textBox2.Text == string.Empty)
    {
        errorProvider2.SetError(textBox2, "Please Enter Family");
    }
}
```

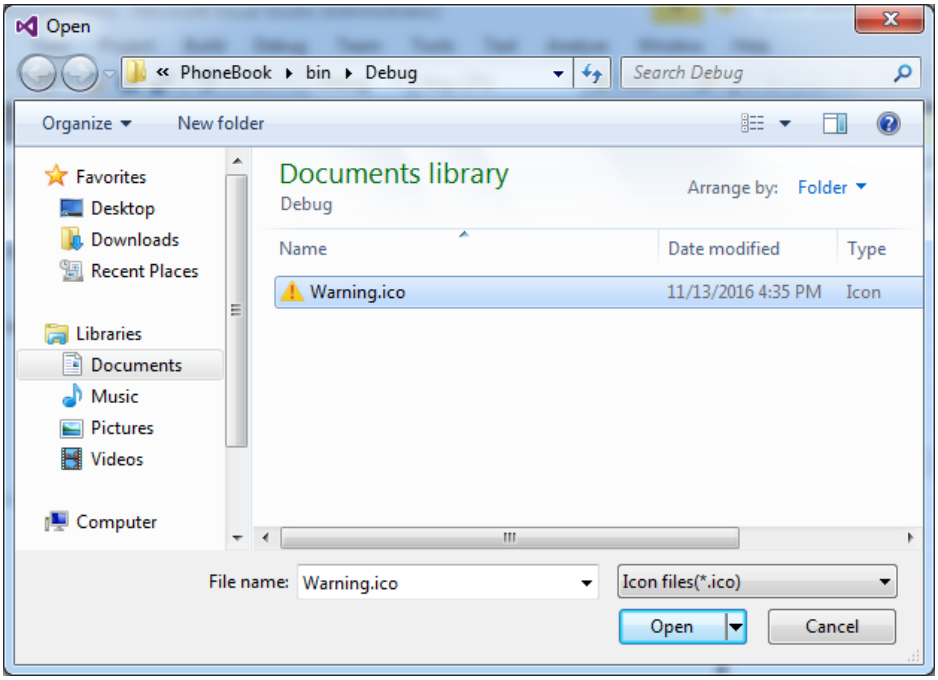
تنها تفاوت مربوط به اختصاص errorProvider2 به کنترل TextBox دوم است. حال برنامه را اجرا کرده و دو بار دکمه Tab را بزنید:



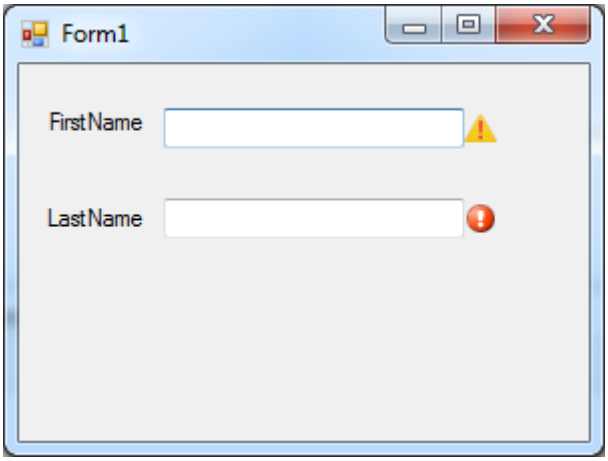
همانطور که احتمالاً تا کنون متوجه شده‌اید این کنترل دارای یک آیکون به صورت پیشفرض می‌باشد. اگر این آیکون رو دوست ندارید می‌توانید از یک آیکون دلخواه استفاده کنید. به عنوان مثال بر روی کنترل errorProvider1 کلیک کرده و سپس در در پنجره Properties بر روی خاصیت Icon مانند شکل زیر کلیک کنید:



با کلیک بر روی این خاصیت پنجره ای ظاهر می‌شود که از شما مسیر آیکون با پسوند `.ico` را می‌خواهد. آیکون را انتخاب کرده و سپس بر روی دکمه`Open` کلیک کنید:



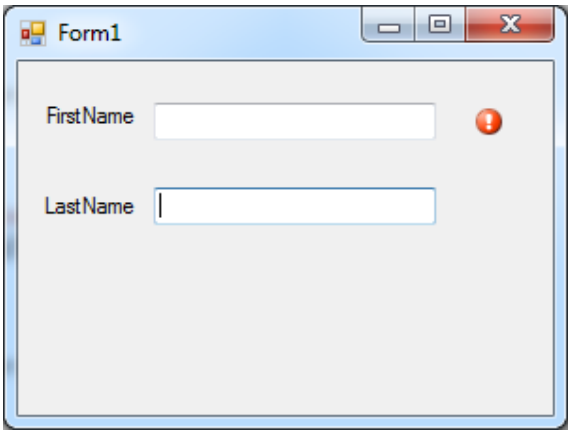
حال برنامه را اجرا کرده و با کلیک بر روی دکمه `Tab` نتیجه را مشاهده کنید:



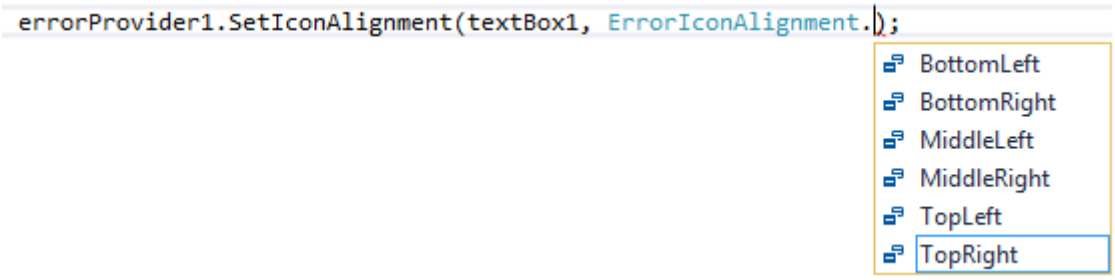
برای تنظیم فاصله آیکون از کنترل می‌توان از متد SetIconPadding() به صورت زیر استفاده کرد:

```
errorProvider1.SetIconPadding(textBox1, 20);
```

کد بالا، یک فاصله ۲۰ پیکسلی بین کنترل TextBox و آیکون ایجاد می‌کند:



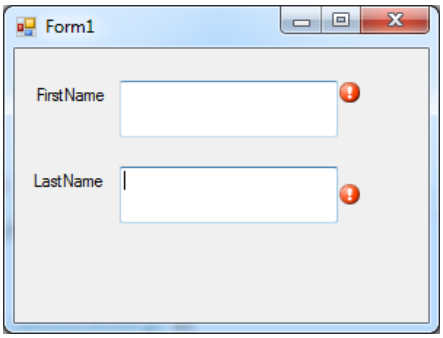
برای تنظیم مکان نمایش Icon هم می‌توان از متد SetIconAlignment() استفاده کرد. این متد دو آرگومان قبول می‌کند که اولی نام کنترل و دیگری یک نوع شمارشی است که مکان نمایش آیکون را مشخص می‌کند:



برای مشاهده عملکرد این متد، ارتفاع دو جعبه متن را تغییر داده و کدهای زیر را برای دو errorProvider بنویسید:

```
errorProvider1.SetIconAlignment(textBox1, ErrorIconAlignment.TopRight);
errorProvider2.SetIconAlignment(textBox2, ErrorIconAlignment.MiddleRight);
```

برنامه را اجرا و با زدن دکمه Tab نتیجه را مشاهده کنید:

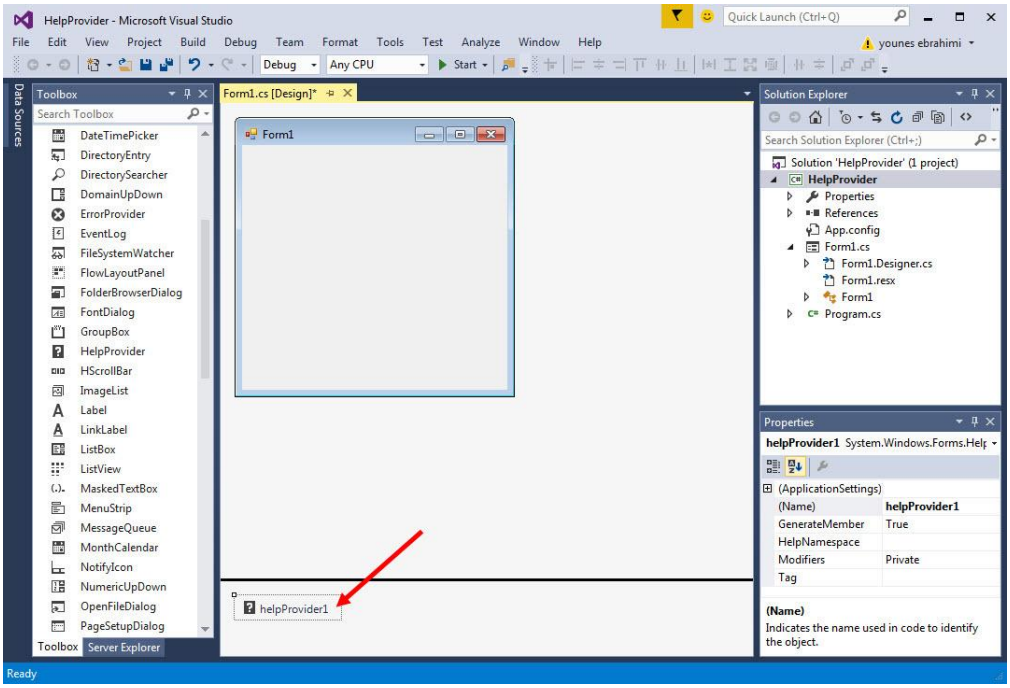


کنترل HelpProvider

از کنترل HelpProvider برای نمایش راهنمای آنلاین استفاده می‌شود. اگر کاربر دکمه F1 را بفشارد، این کنترل، اطلاعات فایل راهنما را نشان می‌دهد. این کنترل دارای خاصیتی به نام HelpNameSpace می‌باشد. این خاصیت نام فایلی را مشخص می‌کند که باید با زدن دکمه F1 نمایش داده شود. فرض کنید که یک فایل با نام help.html و محتویات زیر، در درایو E قرار داده‌ایم و این فایل همان فایل راهنمای ما است:

```
<h1>HelpProvider</h1>
<p>HelpProvider control provides popup or online help for a control.</p>
```

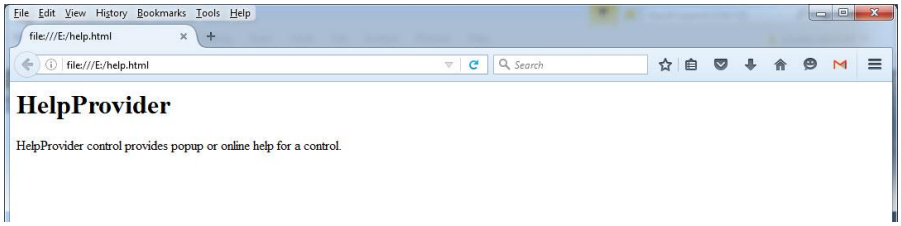
برای استفاده از این فایل ابتدا یک کنترل HelpProvider بر روی برنامه قرار دهید:



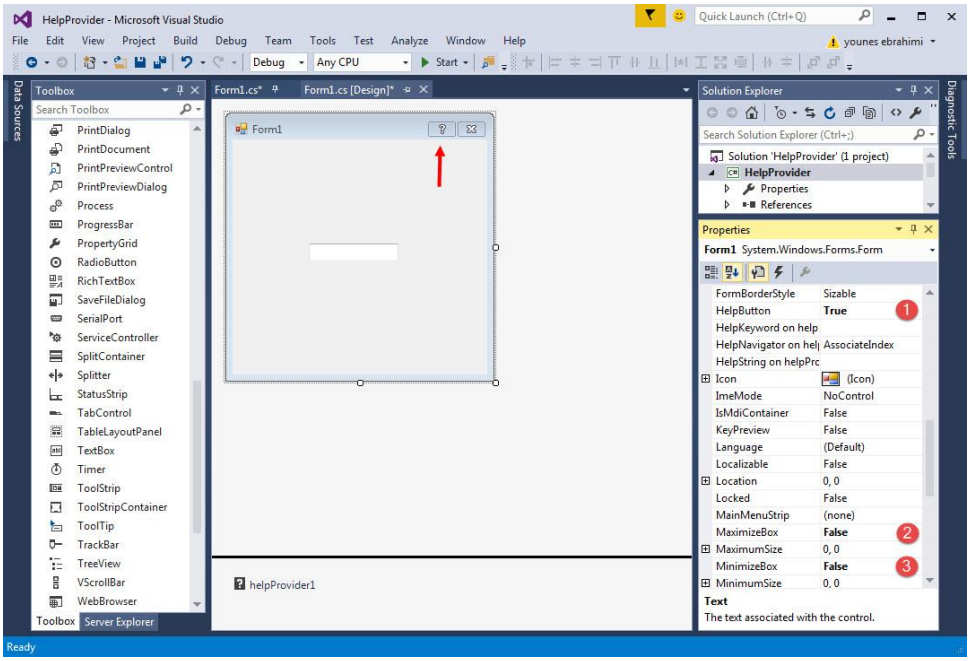
سپس با زدن دکمه F7 کد زیر را در سازنده کلاس برنامه‌تان بنویسید :

```
using System.Windows.Forms;

namespace HelpProvider
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            helpProvider1.HelpNamespace = @"E:\help.html";
            helpProvider1.SetShowHelp(this, true);
        }
    }
}
```

حال برنامه را اجرا کرده و دکمه F1 را بزنید :



همانطور که در شکل بالا مشاهده می‌کنید، با زدن دکمه F1 مرورگر باز شده و محتویات فایلی help.html را نمایش می‌دهد. گاهی اوقات ممکن است که بخواهید برای یک کنترل نیز یک متن توضیحی قرار دهید که نشان دهنده عملکرد آن باشد. برای این کار از متد SetHelpString()استفاده می‌شود. برای کار با این متد باید دکمه Help فعال باشد. برای فعال کردن دکمهHelp هم باید دکمه‌های Maximize و Minimaize غیر فعال شوند. یک کنترل textBox بر روی فرم قرار داده و تغییرات زیر را اعمال کنید :



سپس کد زیر را در ادامه کدهای بالا بنویسید :

```
helpProvider1.SetHelpString(textBox1, "Enter Your FirstName here...");
```

در کد بالا ما با استفاده از متد SetHelpString()، یک توضیح برای کنترل textBox نوشته‌ایم. برای نمایش این توضیح، ابتدا باید بر روی دکمه Help و سپس بر روی کنترل مورد نظر کلیک نمایید :

این کنترل دارای خاصیت‌ها و متدهای دیگری هم هست که در این درس به مهم‌ترین آنها اشاره شد.

کنترل Label

کنترل Label برای اضافه کردن یک متن به فرم بکار می‌رود. این متن می‌تواند حاوی یک پیام و یا توضیحی در باره عملکرد سایر کنترل‌ها باشد.

یک کنترل Lable را از ToolBox به فرم اضافه کنید. در حالت پیش‌فرض این کنترل دارای یک متن است. خواص زیر، معمول‌ترین خواص قابل تغییر این کنترل می‌باشند.

خاصیت	توضیح
AutoSize	اگر مقدار این خاصیت true باشد، اندازه کنترل بسته به متنی که به آن اختصاص می‌دهیم، تغییر می‌کند.
BorderStyle	کنترل را حاشیه دار می‌کند.
Font	برای تغییر فونت متن کنترل به کار می‌رود.
Text	متن کنترل را مشخص می‌کند.
TextAlign	جهت ترازبندی متن داخل کنترل به کار می‌رود.

خاصیت Text، مهم‌ترین خاصیت آن می‌باشد چون هدف اصلی از استفاده از این کنترل نشان دادن متن در فرم است.

```
Label1.Text = "Hello World!";
```

می‌توان خاصیت Font را با تغییر نوع فونت، اندازه فونت و برخی دیگر از خواص آن اصلاح کرد.

رویدادهایی هم برای این کنترل وجود دارد که اکثر مواقع به آن نیازی نیست و در نتیجه در مورد آنها بحث نمی‌کنیم.

کنترل TextBox

کنترل TextBox ابتدایی‌ترین وسیله برای ورود اطلاعات در یک فرم ویندوزی می‌باشد. این کار را با تایپ آنها (اطلاعات) در TextBox انجام می‌دهید. متنی که شما تایپ می‌کنید به وسیله خاصیت Text این کنترل قابل دسترسی است. در جدول زیر خاصیت‌های مفید این کنترل که شما می‌توانید از انها استفاده کنید آمده است:

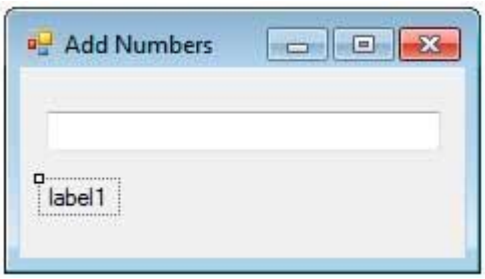
خاصیت	توضیح
AcceptsReturn	تعیین می‌کند که در TextBox متن به صورت چند خطی نوشته شود یا نه. که در این صورت با زدن دکمه Enter در داخل TextBox خط جدید ایجاد می‌شود.
AcceptsTab	در حالت چند خطی، این خاصیت باعث می‌شود که مکان نما با زدن دکمه tab به جای اینکه به کنترل بعدی برود به tab بعد برود.
Enabled	اگر این خاصیت در حالت false قرار بگیرد TextBox به صورت read-only در می‌آید و در نتیجه در حکم یک برچسب می‌شود.
Font	نوع فونتی که در جعبه متن مورد استفاده قرار می‌گیرد.
Lines	خطوط متن در یک TextBox چند خطی را نشان می‌دهد.
Multiline	اگر True باشد به شما اجازه تایپ چندین خط در TextBox را می‌دهد.
Text	متن داخل جعبه متن را نشان می‌دهد.
PasswordChar	کاراکترهایی را که توسط کاربر وارد می‌شوند، پنهان می‌کند.
ReadOnly	تعیین می‌کند که آیا متن قابل ویرایش باشد یا نه.
Visible	قابل رویت بودن یا نبودن TextBox را تعیین می‌کند
WordWrap	در TextBox چند خطی کاربرد دارد. باعث می‌شود وقتی که نشانگر ماوس به انتهای TextBox رسید به صورت اتوماتیک به خط بعد منتقل شود.

در مثال زیر نحوه استفاده از TextBox نشان داده شده است. یک پروژه ویندوزی ایجاد کنید. برنامه از شما می‌خواهد که دو عدد را وارد کنید و با زدن دکمه جمع آنها را با استفاده از یک برچسب (label) به شما نشان می‌دهد. دو textbox را روی فرم بکشید و نام اولی را textBoxFirstNumber و دومی را textBoxSecondNumber بگذارید. سپس در کنار هر textbox یک کنترل label که نشان دهنده هدف آنهاست (textbox ها) قرار دهید. یک کنترل lable دیگر برای نشان دادن حاصل جمع دو عدد در فرم قرار داده و نام آن را labelSum بگذارید. یک دکمه Button را هم اضافه کرده و نام آن را buttonAdd بگذارید. مکان قرارگیری و سایز کنترل‌ها را مانند شکل زیر تنظیم کنید:

بر روی دکمه add دو بار کلیک کرده و در رویداد کلیک مربوط به آن کد زیر را وارد نمایید:

```
private void buttonAdd_Click(object sender, EventArgs e)
{
    int num1 = Int32.Parse(textBoxFirstNumber.Text);
    int num2 = Int32.Parse(textBoxSecondNumber.Text);
    int sum = num1 + num2;

    labelSum.Text = "Sum = " + sum;
}
```

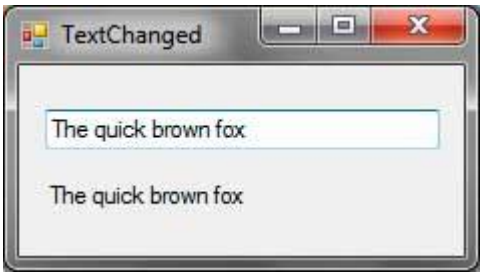
کدهای نوشته شده در داخل textBoxFirstNumber و textBoxSecondNumber با استفاده از متد Parse() کلاس Int32 به نوع صحیح تبدیل شده و در متغیرهای مربوطه ذخیره می‌شوند. ما به این تبدیل نیاز داشتیم چون خاصیت Text از نوع رشته است. سپس جمع آنها محاسبه می‌شود. و در نهایت مقدار جمع به خاصیت Text برچسب labelSum نسبت داده می‌شود. یکی از مفیدترین رویدادهای تکست باکس، رویداد TextChanged است. این رویداد زمانی رخ می‌دهد که متن داخل تکست باکس دستکاری شود (تغییر کند). برنامه‌ی زیر یک مثال از کاربرد این رویداد را نشان می‌دهد. یک برنامه ویندوزی ایجاد و یک تکست باکس (Text Box) و یک برچسب (Label) به آن اضافه کنید. می‌توانید نام آنها را تغییر ندهید.



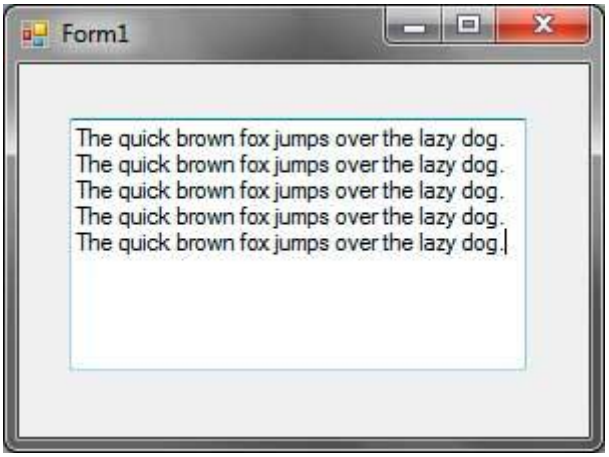
متن برچسب را که در داخل خاصیت Text آن قرارگرفته است را پاک کنید. اگر مقدار خاصیت AutoSize برچسب شما برابر با False باشد انتخاب کردن آن مشکل است. برای انتخاب کردن آن می‌توانید از نوار کرکره‌ای بالای پنجره‌ی خواص (Properties) استفاده کنید. رویداد پیش فرض کنترل تکست باکس (Text Box) رویداد TextChanged است. پس با دوبار کلیک کردن بر روی تکست باکس یک کنترل کننده رویداد (Event Handler) برای رویداد گفته شده ایجاد می‌کند. کد روبرو را به Event Handler اضافه کنید.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```

حالا برنامه را اجرا کرده و هر چیزی را که می‌خواهید درون تکست باکس بنویسید. مشاهده می‌کنید که هر چیزی که شما درون تکست باکس می‌نویسید دقیقاً به همان صورت بر روی بر چسب نوشته می‌شود.



زمانی که متن تکست باکس دستکاری می‌شود، Event Handler اجرا شده و هر متنی را که درون تکست باکس قرار دارد را دقیقاً به همان صورت درون برچسب (label1) قرار می‌دهد. به طور پیش فرض، هر تکست باکس فقط می‌تواند یک خط را درون خود جای دهد. برای اینکه چند خط درون تکست باکس قرار بگیرد، می‌توانید مقدار خاصیت Multiline آنرا برابر با True قرار دهید. وقتی که شما اینکار را انجام می‌دهید، باید توجه داشته باشید که، نمی‌توانید طول تکست باکس را در قسمت طراحی تغییر دهید.

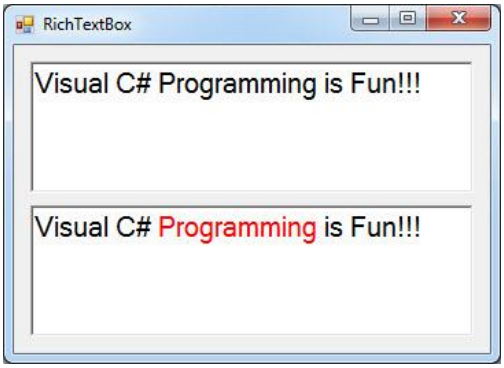


شما می‌توانید مقدار خاصیت WordWrap را برابر با True قرار دهید تا نشانگر ماوس زمانی که به انتهای قسمت راست تکست باکس رسید، به طور اتوماتیک به خط بعد برود. اگر مقدار آن برابر با False باشد، متن در همان خط ادامه پیدا می‌کند. شما می‌توانید مقدار خاصیت ScrollBar را به Horizontal (افقی)، Vertical (عمودی) و یا هر دو تغییر دهید. این کار باعث می‌شود که قسمت پیمایش به تکست باکس چند خطی اضافه شود. اگر تکست باکس شما می‌خواهد یک پسورد را بپذیرد، شما باید مقدار خاصیت PasswordChar را به چیزی مانند * تغییر دهید

(این کاراکتر بسته به سلیقه‌ی شما می‌تواند هر چیزی باشد). وقتی که کاربر یک حرف را وارد کند این کاراکتر (*) به او نمایش داده می‌شود. به شکل زیر دقت کنید.

کنترل RichTextBox

کنترل RichTextBox شبیه کنترل TextBox است با این تفاوت که این کنترل به شما اجازه می‌دهد که قالب قسمت‌های مختلف متن آنرا تغییر دهید. کنترل TextBox به طور کلی برای پذیرفتن اطلاعات ورودی از کاربر بکار می‌رود، در صورتیکه کنترل RichTextBox برای نمایش دادن متن‌های قالب بندی شده و ذخیره‌ی آن در قالب Rich Text Format (RTF) به کار می‌رود.



در تصویر بالا شما تفاوت بین TextBox (بالا) و RichTextBox (پایین) را مشاهده می‌کنید. اگرچه شما می‌توانید یک Text Box را قالب بندی کنید، ولی این قالب بندی به کل متن آن اعمال می‌شود. کنترل RichTextBox به شما اجازه می‌دهد که فقط قسمتی از کل متن داخل کنترل را قالب بندی کنید. شما می‌توانید از کنترل RichTextBox مانند کنترل TextBox برای گرفتن اطلاعات ورودی از کاربر استفاده کنید. هر دوی این کنترل‌ها از کلاس TextBoxBase برگرفته می‌شوند بنابراین، بیشتر خواص آنها مثل خاصیت Text مشترک هستند. در جدول زیر خواص کنترل RichTextBox را مشاهده می‌کنید.

خاصیت	توضیح
AcceptsTab	مشخص می‌کند زمانی که دکمه‌ی Tab فشرده شد Focus بر روی آن قرار گیرد یا خیر.
CanRedo	مشخص می‌کند که تمامی کارهایی که در داخل RichTextBox انجام شد می‌توانند دوباره اعمال شوند یا خیر.
CanUndo	مشخص می‌کند که کاربر می‌تواند عملیات انجام شده در داخل RichTextBox را Undo کند یا خیر.

DetectUrls	مشخص می‌کند که وقتی یک URL در RichTextBox تایپ شود، به طور خودکار قالب بندی شود یا خیر.
Lines	اطلاعات مربوط به کنترل RichTextBox را مشخص می‌کند. چنانچه بر روی دکمه‌ی کنار این خاصیت کلیک کنید، می‌توانید مقادیر آنرا را مشاهده و دستکاری کنید.
Modified	مشخص می‌کند که محتوای RichTextBox پس از آخرین تغییرات دستکاری شده است یا خیر.
Multiline	مشخص می‌کند که کنترل RichTextBox می‌تواند حالت چند خطی داشته باشد یا خیر (مقدار این خاصیت به طور پیش فرض برابر با True است).
ReadOnly	مشخص می‌کند که کنترل RichTextBox فقط خواندنی است یا خیر.
Rtf	متن کنترل RichTextBox را به همراه کدهای فرمت RichText (RTF) را در خود جای می‌دهد.
Scrollbars	برای تعیین نوع Scroll بکار می‌رود. این خاصیت نیز مانند خاصیت ScrollBars کنترل Text Box است.
SelectedText	متن داخل RichTextBox را در بر می‌گیرد.
SelectionBackColor	رنگ پشت زمینه‌ی متن انتخاب شده را مشخص می‌کند.
SelectionBullet	مشخص می‌کند که حالت گلوله ای به متن انتخاب شده اعمال شود یا خیر.
SelectionColor	رنگ متن انتخاب شده را مشخص می‌کند.
SelectionFont	نوع قلم متن انتخاب شده را مشخص می‌کند.
SelectionLength	تعداد کاراکترهای متن انتخاب شده را مشخص می‌کند.
SelectionRightIndent	فاصله بین ضلع راست کنترل RichTextBox و ضلع راست متن انتخاب شده یا محل درج جاری را مشخص می‌کند.
SelectionStart	محل شروع قسمت انتخاب شده یا محل درج را مشخص می‌کند.
ShowSelectionMargin	مشخص می‌کند که حاشیه‌ی انتخاب نمایش داده شود یا خیر.
Text	متنی را درون کنترل RichTextBox قرار می‌دهد.
WordWrap	متن درون کنترل RichTextBox را دسته بندی می‌کند.

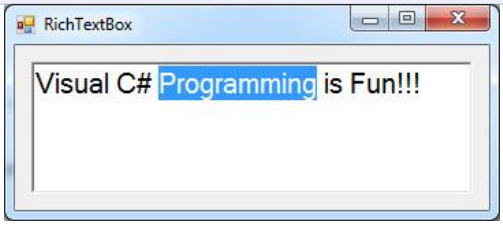
در این قسمت با رویدادهایی آشنا می‌شوید که از آنها می‌توانید برای کنترل RichTextBox استفاده کنید.

رویداد	توضیح
LinkClicked	زمانی رخ می‌دهد که بر روی یک لینک کلیک شود.
Protected	زمانی رخ می‌دهد که کاربر بخواهد یک متن حفاظت شده را دستکاری کند.
TextChanged	زمانی رخ می‌دهد که متن داخل RichTextBox دستکاری شود.
SelectionChanged	زمانی رخ می‌دهد که متن انتخاب شده تغییر کند.

تغییر دادن قالب متن انتخاب شده

تعداد زیادی از خواص کنترل RichTextBox برای متن انتخاب شده استفاده می‌شوند. برای مثال، خاصیت SelectedText متنی را که توسط کاربر انتخاب شده است را، مشخص می‌کند. همچنین شما می‌توانید از یک متد برای انتخاب یک متن استفاده کنید.

```
richTextBox1.Select(10, 11);
```

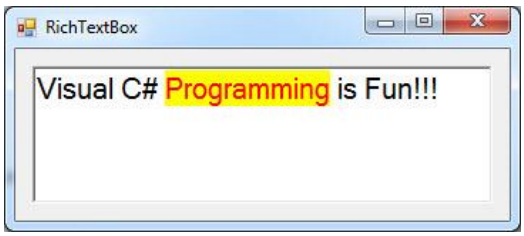


مقدار اول، نقطه‌ی شروع انتخاب و مقدار دوم تعداد کاراکترهایی را که باید از نقطه‌ی شروع انتخاب شوند را، مشخص می‌کند. وقتی ما یک متن را انتخاب می‌کنیم، می‌توانیم متن انتخاب شده را به وسیله‌ی تعداد زیادی از خواص که بر روی متن انتخاب شده کار می‌کنند، قالب بندی کنیم. برای مثال، شما می‌توانید برای تغییر رنگ متن انتخاب شده از خاصیت SelectionColor استفاده کنید. همچنین می‌توانید برای تغییر رنگ پس زمینه‌ی آن از خاصیت SelectionBackColor استفاده کنید.

```
richTextBox1.SelectionColor = Color.Red;
richTextBox1.SelectionBackColor = Color.Yellow;
```

با استفاده از متد DeselectAll() می‌توانید متن انتخاب شده را با کلیک بر روی هر جایی از فرم از حالت انتخاب خارج کنید.

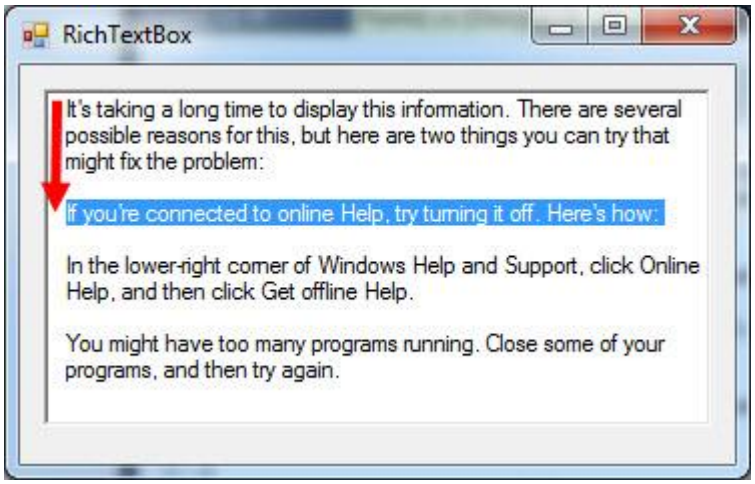
```
richTextBox1.DeselectAll();
```



به این نکته توجه کنید که، وقتی که هیچ متنی انتخاب نشده، تأثیر خواص Selection مثل SelectionColor، از نقطه‌ی درج شروع می‌شود. نقطه‌ی درج، محل چشمک زن و به شکل حرف I است که مشخص می‌کند درج متن از آنجا شروع خواهد شد. هر کاراکتر جدیدی که شما تایپ می‌کنید، قالب بندی جدید بر آنها اعمال می‌شود.

اضافه کردن یک حاشیه به متن انتخاب شده

حاشیه‌ی انتخاب، یک حاشیه‌ی کوچک است که در سمت چپ کنترل RichTextBox قرار می‌گیرد (به شکل توجه کنید).



حاشیه‌ی انتخاب (که با فلش مشخص شده است) برای انتخاب یک خط بکار می‌رود. برای مثال، در تصویر بالا خط سوم توسط کلیک کردن بر روی حاشیه انتخاب کناری آن، انتخاب شده است. ما می‌توانیم از خاصیت SelectionMargin برای نمایش دادن حاشیه‌ی انتخاب استفاده کنیم.

اضافه کردن Scroll (نوار پیمایش)

برای اضافه کردن یک نوار پیمایش، ما از خاصیت ScrollBars که مقادیری از نوع شمارش RichTextBoxScrollBars را قبول می‌کند استفاده می‌کنیم. این مقادیر شمارشی، شامل مقادیری از جمله Horizontal (افقی) است. این مقدار زمانی که طول خط ما زیاد باشد و مقدار خاصیت WordWrap نیز False باشد به کار می‌رود. مقدار بعدی Vertical (عمودی) است، و زمانی بکار می‌رود که تعداد خطوط متن از ارتفاع RichTextBox بیشتر باشد. مقدار Both نیز در مواقع لزوم به صورت اتوماتیک Scrollهای عمودی و افقی را اضافه می‌کند. وقتی مقدار خاصیت SelectionMargin برابر false باشد، شما لازم است که از مقادیر شمارشی دیگری مانند ForcedHorizontal، ForcedVertical و یا ForcedBoth استفاده نمایید. برای غیر فعال کردن نوار پیمایش به صورت کامل کافیست که مقدار خاصیت ScrollBars را برابر None قرار دهید.

تغییر ترازبندی متن انتخاب شده

می‌توانیم از خاصیت SelectionAlignment برای تغییر تراز بندی (Align) متن انتخاب شده استفاده کنیم. این خاصیت مقادیری از نوع شمارشی HorizontalAlignment دریافت می‌کند که شامل Left، Right و Center می‌باشد. مهم است که بدانید پاراگراف‌ها چگونه در

داخل کنترل ساخته می‌شوند. به طور کلی، وقتی شما اولین کاراکتر خود را تایپ می‌کنید، اولین پاراگراف در RichTextBox ساخته می‌شود. برای ساختن پاراگراف بعدی، شما باید دکمه‌ی Enter را بزنید.

Undo و Redo

شما می‌توانید از خواص CanUndo و CanRedo برای اینکه کاربر بتواند عملیات انجام شده را Undo و یا Redo کند، استفاده کنید. Undo کردن یعنی شما تغییراتی را که در اجزای کنترل RichTextBox به وجود آورده‌اید، را برگشت دهید، در صورتیکه Redo عملیات Undo را خنثی می‌کند و تغییراتی را که شما قبلاً در اجزاء RichTextBox به وجود آورده‌اید را، بر می‌گرداند. خواص UndoActionName و RedoActionName برای محدود کردن عملیات Undo و Redo به کارهای مشخصی به کار می‌روند. این خواص عملیات‌های زیر را به صورت رشته‌ای قبول می‌کنند.

عمل	توضیح
Typing	عملیات تایپ
Delete	عملیات حذف
DragDrop	عملیات Drag & Drop
Cut	عملیات برش (Cut)
Paste	عملیات Paste

برای مثال وقتی که شما کلمه‌ی "Delete" را به خاصیت UndoActionName نسبت می‌دهید. اینکار باعث می‌شود که کاربر فقط بتواند عملیات حذف کردن را برگشت (Undo) دهد.

```
richTextBox1.UndoActionName = "Delete";
```

به یک دلیل نامعلوم مایکروسافت برای این خواص نوع شمارشی ایجاد نکرده است. بنابراین شما باید از یک رشته برای انجام این اعمال، استفاده نمایید.

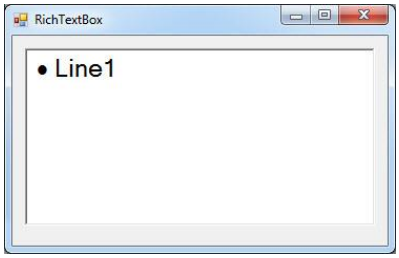
شناسایی URL

کنترل RichTextBox این قابلیت را داراست که، بتواند URL هایی را که در آن تایپ و یا Paste می‌شوند را تشخیص دهد. خاصیت DetectUrls به کنترل اجازه می‌دهد که URLها را تشخیص داده و یک لینک به آن صفحه ایجاد کند. مقدار True این خاصیت را فعال و مقدار False این خاصیت را غیر فعال می‌کند. در تصویر زیر، شما یک URL را مشاهده می‌کنید که نشان دهنده‌ی یک لینک است. و زمانی که شما نشانگر ماوس را بر روی آن ببرید، آیکن ماوس به شکل یک دست تغییر پیدا می‌کند.

کلیک کردن بر روی Link آنرا در داخل مرورگر شما باز می‌کند.

اضافه کردن بالت ها

شما می‌توانید در داخل RichTextBox گلوله‌هایی را به وجود آورید. شما می‌توانید از خاصیت SelectionBullet و قرار دادن مقدار True برای نشانه گذاری متن انتخاب شده استفاده کنید. خاصیت BulletIndent فاصله‌ی هر گلوله را با ابتدای متن مشخص می‌کند. برنامه‌ی زیر یک دکمه دارد که در ابتدای هر خط انتخاب شده یا پارگراف یا جایی که محل درج قرار گرفته است یک گلوله اضافه می‌کند.



فعال و غیر فعال کردن کلیدهای میانبر

خاصیت ShortcutsEnabled کلیدهای میانبری را که برای Cut و Paste کردن متن بکار می‌روند، فعال یا غیر فعال می‌کند. وقتی که مقدار این خاصیت برابر با True باشد، کلیدهای میانبر زیر توسط RichTextBox قبول می‌شوند.

میانبر	عملکرد
Ctrl + Z	بازگردانی
Ctrl + E	وسط چین کردن متن
Ctrl + C	کپی کردن متن
Ctrl + Y	برعکسUndo
Ctrl + X	بریدن قسمتی از متن
Ctrl + Backspace	تمامی کلماتی که در سمت چپ نشانگر موس قرار دارند را حذف می‌کند
Ctrl + V	کردنPaste

Ctrl + Delete	تمامی کلماتی را که در سمت راست نشانگر موس قرار دارند را حذف می‌کند.
Ctrl + A	همه‌ی متن را انتخاب می‌کند.
Ctrl + L	چپ چین کردن
Ctrl + R	راست چین کردن

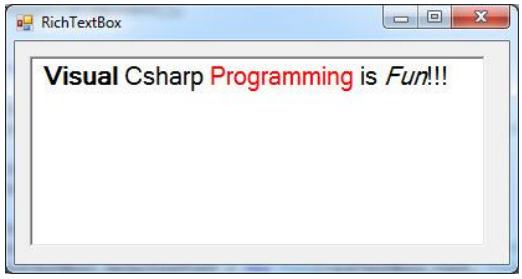
وقتی که مقدار این خاصیت را برابر با False قرار دهید، تمامی این کلیدهای میانبر در داخل RichTextBox غیر فعال می‌شوند.

حفاظت از متن

شما قابلیت محافظت از قسمتی از متن را با استفاده از خاصیت SelectionProtected را دارا هستید. وقتی که مقدار این خاصیت برابر با True باشد، قسمتی از متن را که انتخاب کرده‌اید محافظت می‌شود. متنی که محافظت شده باشد را، نمی‌توان دستکاری کرد. وقتی که کاربر می‌خواهد یک متن حفاظت شده را دستکاری کند، رویداد Protect اتفاق می‌افتد. شما می‌توانید بر روی این رویداد مانور دهید. برای مثال، می‌توانید یک پیغام را به کاربر نشان دهید مبنی بر اینکه او در حال تلاش برای دستکاری یک متن حفاظت شده است.

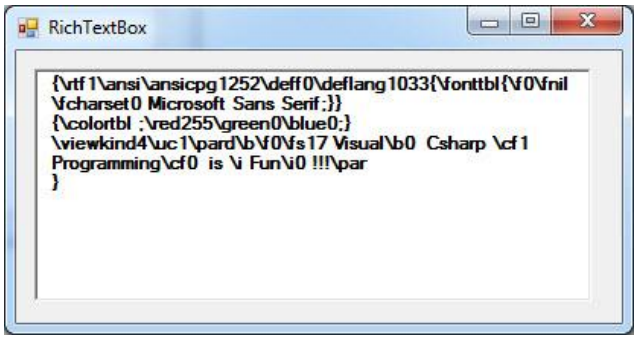
Rich Text Format (RTF)

Rich Text Format فایلی است که اطلاعات قالب (فرمت) متن شما را در خود ذخیره می‌کند. یک TextBox نمی‌تواند قالب (فرمت) متن‌ها را در خود ذخیره کند و فقط می‌تواند حاوی متن‌های ساده باشد. RichTextBox حاوی خواص Text و RTF است که خاصیت Text آن متن ساده و خاصیت RTF آن، متن و کدهای RTF ی را که برای قالب بندی آن به کار رفته است را در خود جای می‌دهد. تصویر زیر یک متن قالب بندی شده را به همراه کدهای RTF آن نشان می‌دهد.



کدی که متن موجود در شکل بالا را قالب بندی می‌کند به صورت زیر است:

```
richTextBox1.Select(0, 6);
richTextBox1.SelectionFont = new Font(richTextBox1.Font, FontStyle.Bold);

richTextBox1.Select(14, 11);
richTextBox1.SelectionColor = Color.Red;

richTextBox1.Select(29, 3);
richTextBox1.SelectionFont = new Font(richTextBox1.Font, FontStyle.Bold);
richTextBox1.SelectionFont = new Font(richTextBox1.Font, FontStyle.Italic);
```



و کدی که همین متن را به RTF تبدیل می‌کند به صورت زیر است:

```
richTextBox1.Text = richTextBox1.Rtf;
```

کدهای RTF به برنامه‌های دیگر (نظیر Word) اجازه می‌دهد که قالب‌ها را خوانده و به همان صورت، متن قالب بندی شده را به شما نمایش دهند. SelectRTF متن انتخاب شده را به همراه کد RTF آن انتخاب می‌کند. این به شما اجازه می‌دهد که، متن قالب بندی شده را به همراه قالب آن در هنگام Cut کردن و Paste کردن، با هم داشته باشید.

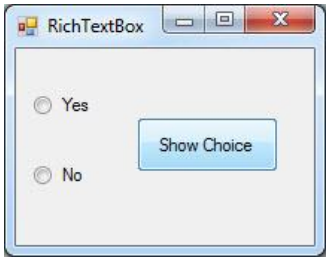
کنترل RadioButton

کنترل RadioButton دکمه ای است که دارای دو حالت خاموش و روشن می‌باشد. دکمه‌ی Radio یک دکمه‌ی دایره‌ای شکل به همراه یک برچسب است. شما با کلیک کردن بر روی دکمه‌ی Radio می‌توانید آنرا از حالت خاموش به روشن و یا بلعکس تغییر دهید. وقتی که یک دکمه‌ی Radio روشن باشد، یک نقطه در وسط آن قرار می‌گیرد، و زمانی که خاموش باشد، دایره‌ی آن خالی است. دکمه‌های Radio معمولاً زمانی استفاده می‌شوند که یک کاربر می‌بایست از بین چند گزینه یکی از آنها را انتخاب کند. برای مثال، زمانی که شما بخواهید جنسیت کاربر را مشخص کنید، می‌توانید از دو دکمه‌ی Radio با نام‌های مرد و زن استفاده کنید. وقتی که شما از دکمه‌های رادیویی استفاده کردید، فقط می‌توانید یکی از آنها را انتخاب کنید. در جدول زیر برخی از خواص کنترل Radio را مشاهده می‌کنید.

خاصیت	توضیح
Appearance	دکمه‌ی Radio می‌تواند همانند یک دکمه‌ی معمولی و یا یک دکمه‌ی دایره‌ای همراه با یک برچسب در کنار آن نمایش داده شود.
CheckAlign	نوع چینش دکمه را مشخص می‌کند. حالت پیش‌فرض MiddleLeft است، که دکمه را در سمت برچسب آن نمایش می‌دهد.
Checked	وقتی که مقدار آن برابر با True باشد دکمه‌ی Radio روشن می‌شود و یک نقطه در وسط دکمه‌ی آن نمایش داده می‌شود.
Text	متن برچسب دکمه را مشخص می‌کند.

دکمه‌ی Radio یک رویداد CheckChanged و Click دارد. رویداد CheckChanged زمانی رخ می‌دهد که، حالت دکمه تغییر پیدا کند. برای مثال وقتی که حالت دکمه‌ی Radio از خاموش به روشن تغییر کند، رویداد CheckChanged اجرا می‌شود. رویداد Click زمانی رخ می‌دهد که، بر روی دکمه Radio کلیک شود. به طور پیش فرض اگر بر روی دکمه Radio کلیک کنید، حالت آن تغییر می‌کند، که باعث می‌شود رویداد CheckChanged اجرا شود.

تفاوت بین رویداد Click و CheckChanged زمانی مشخص می‌شود که شما این حالت پیش فرض را تغییر دهید. در اینجا یک خاصیت در کنترل Radio به نام AutoCheck وجود دارد که اگر مقدار آن را به False تغییر کند، کلیک کردن بر روی دکمه‌ی Radio دیگر حالت آن را تغییر نمی‌دهد، اما باعث می‌شود که رویداد Click اتفاق بیافتد. شما می‌توانید به صورت دستی مقدار خاصیت Checked دکمه‌ی Radio را True قرار دهید. رویداد پیش‌فرض برای دکمه‌ی Radio رویداد CheckChanged است، بنابراین، دابل کلیک کردن بر روی این کنترل در حالت طراحی یک Event Handler برای رویداد گفته شده، درست می‌کند. مثال زیر کاربرد دکمه‌ی Radio را شرح می‌دهد. دو عدد دکمه‌ی Radio را بر روی فرم قرار دهید. نام آنها را radioButtonYes و radioButtonNo بگذارید. یک کنترل Button را نیز بسازید و نام آنرا buttonShow قرار دهید و مقدار خاصیت Text آنرا برابر Show Choice قرار دهید.



بر روی buttonShow دوبار کلیک کنید تا یک handler برای رویداد Click آن ساخته شود. کدهای روبرو را در قسمت کد آن وارد کنید.

```
private void buttonShow_Click(object sender, EventArgs e)
{
    if (radioButtonYes.Checked)
       MessageBox.Show("You choosed yes!");
    else
       MessageBox.Show("You choosed no!");
}
```

وقتی که شما بر روی buttonShow کلیک می‌کنید، برنامه تعیین می‌کند که کدام Radio انتخاب شده است. شما می‌توانید این کار را به وسیله‌ی خاصیت Checked انجام دهید. ما از یک عبارت شرطی if برای تعیین اینکه Radio انتخاب شده است یا خیر، استفاده می‌کنیم. اگر آن انتخاب نشده باشد، پس دکمه‌ی Radio ی دیگر انتخاب شده است. چون ما فقط دو دکمه‌ی Radio بر روی فرم دارم و نهایتاً یکی از آنها انتخاب می‌شود.

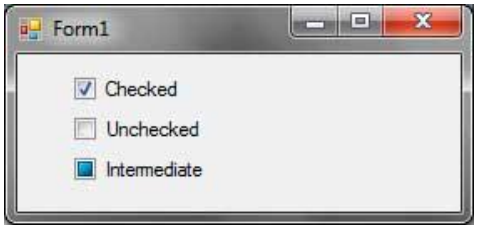
کنترل CheckBox

کنترل CheckBox (System.Windows.Forms.CheckBox) یک دکمه است که به شکل یک جعبه‌ی خالی به همراه یک برچسب در کنار آن نمایش داده می‌شود. در حالت عادی، زمانی که بر روی جعبه‌ی خالی کلیک شود، یک تیک در داخل جعبه نمایان می‌شود که به ما می‌گوید کنترل CheckBox در حالت Checked قرار دارد. برخلاف دکمه‌ی Radio که فقط اجازه‌ی انتخاب یکی از Radioهای فرم را به ما می‌داد، شما می‌توانید

چند عدد CheckBox و یا همه‌ی آنها را تیک بزنید. CheckBox نیز خواصی را شبیه به خواص Radio در خود جای داده است. خواص زیر منحصراً مربوط به کنترل CheckBox هستند.

خاصیت	توضیح
Checked	وقتی که CheckBox تیک خورده باشد مقدار آن True می‌شود.
CheckState	مشخص می‌کند که CheckBox آیا تیک خورده است یا خیر.
ThreeState	وقتی مقدار آن برابر با True باشد، CheckBox یک حالت دیگری را به نام InterMediate را قبول می‌کند.

بر خلاف دکمه‌ی Radio، کنترل CheckBox می‌تواند با دادن مقدار True به خاصیت ThreeState سه حالت داشته باشند. آن حالات Checked، Unchecked و Intermediate هستند. حالت میانی (Intermediate) نشان می‌دهد که checkbox بی اعتبار یا غیر قابل تشخیص است. تصویر زیر شکل این سه حالت مختلف را نشان می‌دهد.



شما با استفاده از خاصیت CheckState می‌توانید حالت این کنترل را در قسمت کد نویسی تعیین کنید. CheckState مقادیری از نوع شمارشی قبول می‌کند.

```
checkBox1.CheckState = CheckState.Checked;
checkBox2.CheckState = CheckState.Unchecked;
checkBox3.CheckState = CheckState.Intermediate;
```

اگر Checkbox فقط دو حالت On یا Off (تیک خورده و تیک نخورده) را قبول کند، آنگاه شما به سادگی می‌توانید از خاصیت Checked آن که دو مقدار True به معنی اینکه Checkbox تیک خورده و False به معنی اینکه CheckBox تیک نخورده است را قبول می‌کند، استفاده کنید. رویداد پیش‌فرض کنترل CheckBox، رویداد CheckChanged است. اما تفاوت بسیار کمی بین رویداد CheckChanged کنترل CheckBox و RadioButton وجود دارد. برای مثال اگر مقدار خاصیت ThreeState کنترل Checkbox را True قرار دهید، وقتی CheckBox حالتش را از Checked به InterMediate تغییر دهد، رویداد CheckChanged آن اتفاق نمی‌افتد. اگر می‌خواهید زمانی که CheckBox از Checked به Intermediate تغییر حالت می‌دهد، یک رویداد اتفاق بیوفتد، می‌توانید از رویداد CheckStateChanged به جای CheckChanged استفاده کنید. در مثال زیر نحوه‌ی کاربرد کنترل CheckBox را مشاهده می‌کنید. یک فرم را ساخته و یک Lable به آن اضافه کنید، سپس سه عدد CheckBox و نیز یک Button به آن اضافه کنید. متن کنترل‌ها و نحوه‌ی چینش آنها را مطابق تصویر قرار دهید.

خاصیت Name، چک باکس‌ها را به checkBoxSoap، checkBoxShampoo و checkBoxToothpaste تغییر دهید. مشخصه‌ی Name، دکمه را به buttonCheckOut تغییر دهید. نام کنترل label را لازم نیست که تغییر دهید، چون در کدنویسی از آن استفاده نمی‌شود. بر روی دکمه (Button) دوبار کلیک کرده و کدهای زیر را در داخل کنترل کننده‌ی رویداد (Event Handler) کپی کنید.

```
string items = String.Empty;

if (checkBoxSoap.Checked)
    items += "\n Soap";
if (checkBoxShampoo.Checked)
    items += "\n Shampoo";
if (checkBoxToothpaste.Checked)
    items += "\n Toothpaste.";

MessageBox.Show("You have bought: " + items);
```

برنامه را اجرا کرده و برخی از CheckBoxها را تیک بزنید. وقتی که شما یک CheckBox را تیک می‌زنید، و سپس بر روی دکمه Check Out کلیک می‌کنید، آیتم‌هایی که قبلاً تیک زده‌اید در یک پنجره نمایش داده می‌شوند. به شکل زیر توجه کنید.

در کد بالا، یک متغیر از نوع رشته‌ای تعریف کرده‌ایم و آنرا با یک رشته‌ی تهی مقدار دهی کرده‌ایم (با استفاده از خاصیت Empty از کلاس String). سپس چک می‌کنیم که کدام یک از چک باکس‌ها تیک خورده‌اند، هر کدام از آنها که تیک خورده‌اند، نامشان به وسیله‌ی عملگر =+ در رشته‌ای که قبلاً به صورت تهی تعریف کرده‌ایم قرار می‌گیرد. سپس به وسیله‌ی یک پیغام (Message Box) نتایج را در خروجی نمایش می‌دهیم.

کنترل ListBox

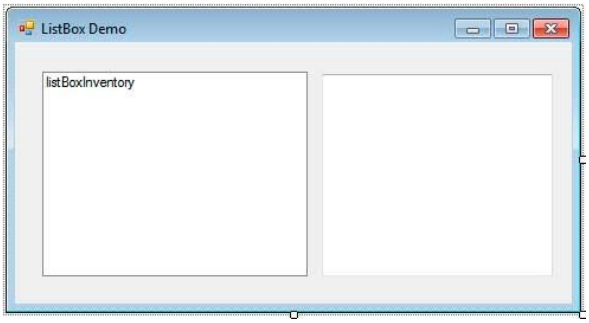
کنترل ListBox برای نمایش لیستی از رشته‌ها که قابل انتخاب هستند، استفاده می‌شود. به طور پیش فرض شما فقط می‌توانید یک آیتم را انتخاب کنید. کنترل ListBox، بهترین گزینه برای مواقعی است که شما می‌خواهید تعداد زیادی آیتم را نمایش دهید. در جدول زیر برخی از خواص کنترل ListBox را مشاهده می‌کنید.

خواص	توضیح
ColumnWidth	وقتی که مقدار خاصیت MultiColumn برابر با True باشد، این خاصیت پهنای هر ستون را مشخص می‌کند.
DataSource	منبع داده‌هایی را که در ListBox نمایش داده می‌شوند، را مشخص می‌کند
Items	حاوی آیتم‌هایی است که در ListBox نمایش داده می‌شوند.
MultiColumn	مشخص می‌کند که ListBox حالت چند ستونی را پشتیبانی می‌کند یا خیر.
SelectedIndex	اندیس آیتم انتخاب شده را مشخص می‌کند.
SelectedIndices	حاوی اندیس آیتم‌های انتخاب شده است.
SelectedItem	آیتم انتخاب شده را به عنوان یک شیء بر می‌گرداند.
SelectedItems	مجموعه‌ای از اشیاء آیتم‌های انتخاب شده را، بر می‌گرداند.
SelectionMode	تعداد آیتم‌هایی که شما در یک لحظه می‌توانید انتخاب کنید را مشخص می‌کند. SelectionMode.None- هیچ آیتمی را نمی‌توانید انتخاب کنید. SelectionMode.One- فقط یک آیتم را می‌توانید انتخاب کنید. SelectionMode.MultiSimple- شما می‌توانید چند آیتم را به سادگی با کلیک کردن بر روی آنها انتخاب کنید. SelectionMode.MultiExtended- شما می‌توانید آیتم‌های مختلفی را در با نگه داشتن کلیدهای Ctrl، Shift و همچنین کلیدهای مکان نما انتخاب کنید.
ScrollAlwaysVisible	این خاصیت بدون توجه به آیتم‌های انتخاب شده در ListBox، Scroll Bar را همیشه نمایش می‌دهد.
Sorted	آیتم‌های ListBox را براساس حروف الفبا و یا بر اساس ترتیب صعودی مشخص می‌کند.
Text	اگر از یک مقدار رشته‌ای استفاده شود، اولین آیتمی که با آن همخوانی داشته باشد، انتخاب می‌شود. این خاصیت، متن اولین آیتم انتخاب شده را بر می‌گرداند.

در جدول زیر متدهای مفیدی که شما می‌توانید آنها را بکار ببرید آورده شده است.

متدها	توضیح
ClearSelected	همه‌ی آیتم‌های انتخاب شده‌ی ListBox را از حالت انتخاب در می‌آورد.
FindString	اولین آیتم ListBox را که با رشته‌ی مشخص شده شروع شده باشد را پیدا می‌کند. جستجو از اندیس مشخص شده شروع می‌شود.
FindStringExact	اولین آیتمی از ListBox که دقیقاً برابر با رشته‌ی مورد نظر باشد را، انتخاب می‌کند.
GetSelected	به ما می‌گوید که آیتمی که اندیس مشخص شده را داراست، انتخاب شده است یا خیر.
SetSelected	آیتمی را که اندیس مشخص شده را داراست انتخاب و یا از حالت انتخاب در می‌آورد.

برای دستکاری آیتم‌های درون یک لیست باکس، ما از خاصیت Items که از نوع ObjectCollection است استفاده می‌کنیم. شما می‌توانید از مجموعه‌ای از متدها مثل Add()، Remove()، Clear() استفاده کنید. یک فرم را طراحی کرده و یک ListBox و TextBox به آن اضافه کنید. مقدار خاصیت Multiline کنترل TextBox را به True تغییر دهید. از طرح بندی زیر استفاده کنید.



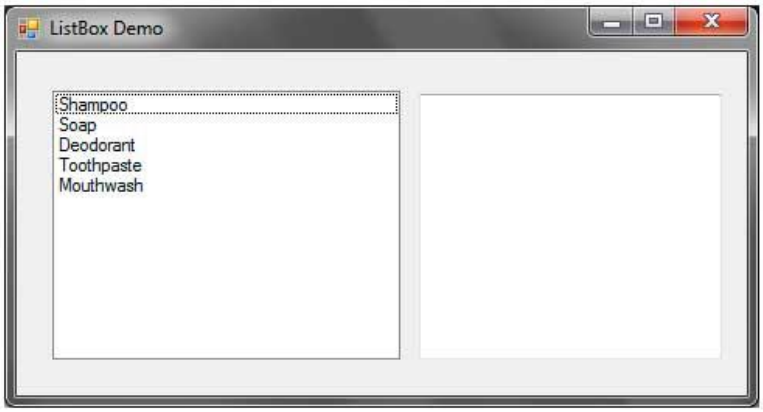
مقدار خاصیت Name کنترل ListBox را به listBoxInventory و کنترل TextBox را به textBoxDescription تغییر دهید. بر روی فرم دوبار کلیک کرده تا یک کنترل کننده رویداد Load (Event Handler)، به آن اضافه شود. کدهای زیر را به آن اضافه کنید.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsFormTutorial
{
    public partial class Form1 : Form
    {
        private Dictionary<string, string> products;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            products = new Dictionary<string, string>();
            products.Add("Shampoo", "Makes your hair beautiful and shiny.");
            products.Add("Soap", "Removes the dirt and germs on your body.");
            products.Add("Deodorant", "Prevents body odor.");
            products.Add("Toothpaste", "Used to clean your teeth.");
```

```
            products.Add("Mouthwash", "Fights bad breath.");

            foreach (KeyValuePair<string, string> product in products)
            {
                listBoxInventory.Items.Add(product.Key);
            }
        }
    }
}
```

ما مجموعه‌ای به نام Dictionary که یک کلید رشته‌ای و یک مقدار رشته‌ای دارد را ایجاد، در داخل کنترل کننده‌ی رویداد Load فرم، ما تعدادی محصول را به همراه توضیحاتی در مورد آنها به این مجموعه اضافه کرده‌ایم. با استفاده از حلقه‌ی Foreach، نام هر محصول را به خاصیت Items لیست باکس اضافه می‌کنیم.

به این نکته توجه کنید که هر آیتم در کلکسیون عمومی Dictionary از نوع KeyValuePair<TKey,TValue> می‌باشد. وقتی برنامه را اجرا می‌کنید، شما پنج محصول را در داخل لیست باکس مشاهده می‌کنید. به یاد داشته باشید، در صورتیکه ارتفاع لیست باکس شما برای نمایش آیتم‌های آن کافی نباشد، یک نوار Scroll عمودی در سمت راست آن دیده خواهد شد.



حالا یک کنترل کننده‌ی رویداد به رویداد SelectedIndexChanged لیست باکس اضافه می‌کنیم. رویداد SelectedIndexChanged زمانی رخ می‌دهد که، اندیس آیتم انتخاب شده، تغییر کند. این رویداد، رویداد پیش‌فرض لیست باکس است، بنابراین با دوبار کلیک کردن بر روی لیست باکس بطور اتوماتیک یک کنترل کننده‌ی رویداد برای رویداد SelectedIndexChanged ایجاد می‌شود. کد زیر را به آن اضافه کنید.

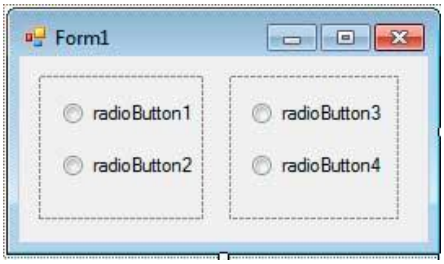
```
private void listBoxInventory_SelectedIndexChanged(object sender, EventArgs e)
{
    textBoxDescription.Text = products[listBoxInventory.Text];
}
```

حالا برنامه‌ی خود را اجرا کرده و یک محصول را انتخاب کنید. مشاهده می‌کنید که توضیحات مربوط به آن محصول در تکست باکس نمایش داده می‌شود.

به این نکته نیز توجه داشته باشید که همچنین می‌توانید از خاصیت Items برای اضافه کردن آیتم‌ها به لیست باکس استفاده کنید.

کنترل‌های Panel و GroupBox

کنترل Panel برای گروه بندی کنترل‌های فرم بکار می‌رود. یکی از کاربردهای خوب Panel وقتی است که، شما می‌خواهید دکمه‌های Radio را گروه بندی کنید. با توجه به اینکه فقط یکی از دکمه‌های Radio در فرم می‌تواند فعال باشد، شما می‌توانید با گروه بندی کردن آنها، بیش از یک دکمه‌ی Radio فعال داشته باشید. برای انجام اینکار، دو Panel را بر روی فرم قرار دهید. کنترل Panel در قسمت نوار ابزار (ToolBox) قرار دارد. سایز آنها را تغییر داده، و در داخل هر یک از آنها دو دکمه‌ی Radio قرار دهید. فرم شما باید فرمی شبیه به شکل زیر داشته باشد.



وقتی که کنترلی را در داخل یک Panel قرار می‌دهید، آن کنترل به فرزند Panel تبدیل می‌شود و Panel به والد کنترل تبدیل می‌شود. برای درک این مساله کافیست، Panel را حرکت دهید. هر کنترلی که در داخل Panel قرار دارد به همراه آن حرکت می‌کند. این ارتباط به والد و فرزند اجازه می‌دهد که، خواص عمومی بین آنها به اشتراک گذاشته شود.

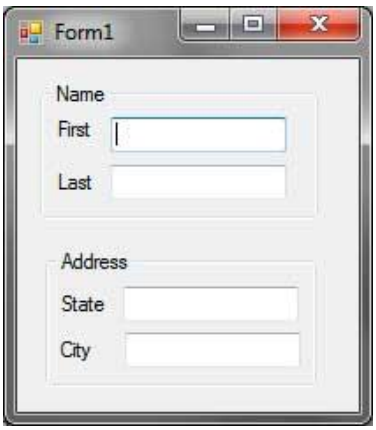
به عنوان مثال، اگر مقدار خاصیت Enable مربوط به Panel را برابر با False قرار دهیم، نه تنها خود Panel بلکه تمامی کنترل‌های داخل آن نیز به حالت غیر فعال در می‌آیند. همانطور که می‌دانید Panel در حالت طراحی به شکل خط تیره نمایش داده می‌شود. این خطوط در حالت اجرایی برنامه نمایش داده نمی‌شوند. اما شما این امکان را دارید که یک حاشیه به پنل اضافه کنید. در جدول زیر تعدادی از خواص پنل را مشاهده می‌کنید.

خاصیت	توضیح
BorderStyle	نوع حاشیه‌ی پنل را مشخص می‌کند.
Controls	مجموعه‌ی کنترل‌های داخل پنل را نشان می‌دهد.

Enabled	به شما اجازه می‌دهد که پنل را فعال یا غیر فعال کنید.

حالا برنامه را اجرا کنید. مشاهده می‌کنید که می‌توانید از دسته‌های مختلف یک دکمه‌ی Radio را انتخاب کنید.

کنترل GroupBox شبیه کنترل panel است با این تفاوت که، به شما اجازه می‌دهد که برای هر دسته یک عنوان قرار دهید. شما می‌توانید با استفاده از خاصیت Text کنترل GroupBox یک عنوان برای آن قرار دهید. همچنین کنترل GroupBox در دور آن به طور پیش‌فرض یک حاشیه قرار دارد.



کنترل ComboBox

کنترل ComboBox (نوار کرکره‌ای) به کاربر اجازه می‌دهد از بین گزینه‌های مختلف، یکی را انتخاب کند. کنترل ComboBox شبیه یک کنترل TextBox است که در سمت راست آن یک دکمه قرار دارد. وقتی بر روی دکمه‌ی آن کلیک شود، ComboBox یک نوار کرکره‌ای، که حاوی یک لیست از گزینه‌های مختلف است را، نمایش می‌دهد. کاربر می‌تواند از بین این گزینه‌ها یکی را انتخاب کند. مورد انتخاب شده به متن داخل ComboBox تبدیل می‌شود. در جدول زیر برخی از خواص این کنترل را مشاهده می‌کنید.

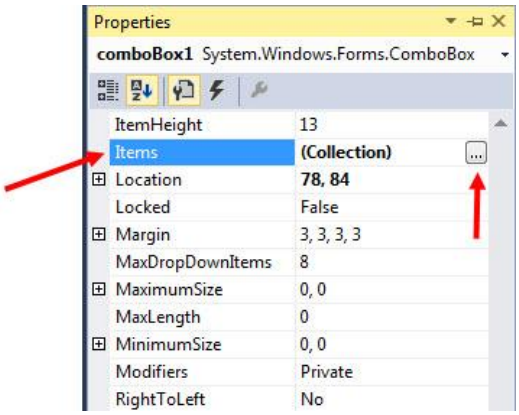
خصوصیت	توضیح
DataSource	لیست داده‌هایی که این کنترل برای گرفتن داده‌هایش از آنها استفاده می‌کند.
DropDownHeight	ارتفاع نوار کرکره‌ای بر حسب پیکسل را مشخص می‌کند.
FormatString	قالبی که نحوه‌ی نمایش مقادیر را مشخص می‌کند.

Items	می‌توان آیتم‌هایی را برای نمایش در درون ComboBox را در داخل این خاصیت قرارداد.
Sorted	مشخص می‌کند که آیتم‌های ComboBox مرتب شوند یا خیر.
Text	متن پیشفرضی است که در حالتی که هیچ آیتمی انتخاب نشده است نمایش داده می‌شود.
SelectedIndex	شماره (index) آیتم انتخاب شده را مشخص می‌کند. هر آیتم یک شماره دارد که از ۰ تا (۱ – تعداد آیتم‌ها) است. مقدار ۱، مشخص می‌کند که هیچ آیتمی انتخاب شده است.
SelectedItem	آیتم انتخاب شده را بر می‌گرداند.

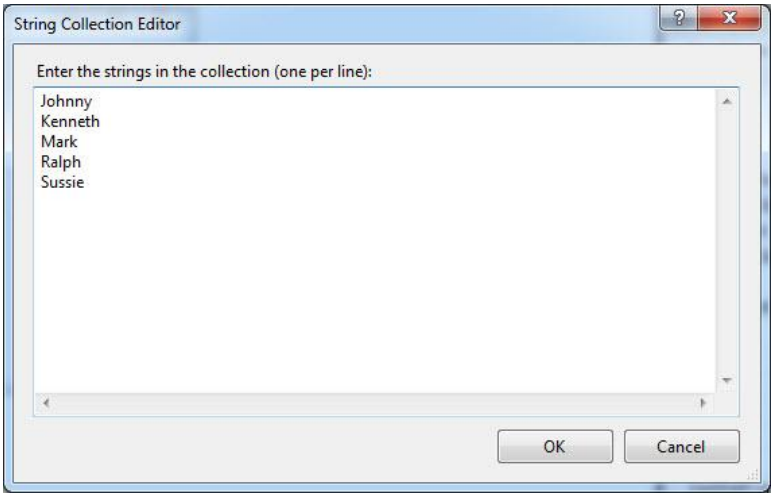
در جدول زیر رویدادهای مربوط به ComboBox را مشاهده می‌کنید. رویداد پیش فرض آن SelectedIndexChanged است.

رویداد	توضیح
Click	زمانی که بر روی یکی از اجزای آن کلیک شود این رویداد رخ می‌دهد.
DropDown	زمانی که نوار کرکره‌ای نمایش داده می‌شود این رویداد رخ می‌دهد.
DropDownClosed	زمانی که نوارکرکره‌ای بسته می‌شود این رویداد رخ می‌دهد.
SelectedIndexChanged	زمانی که مقدار خاصیت SelectedIndex تغییر می‌کند، این رویداد رخ می‌دهد.

در مثال زیر با کاربرد ComboBox آشنا می‌شوید. ابتدا در داخل فرم یک ComboBox قرار دهید. مقدار خاصیت Name آنرا به comboBoxNames تغییر دهید. به پنجره‌ی Properties رفته و خاصیت Items را پیدا کنید. یک دکمه را به همراه سه نقطه مشاهده می‌کنید. بر روی آن کلیک کرده تا پنجره String Collection Editor باز شود. همچنین شما می‌توانید بر روی Edit Items در قسمت پایین پنجره‌ی Properties کلیک کنید.



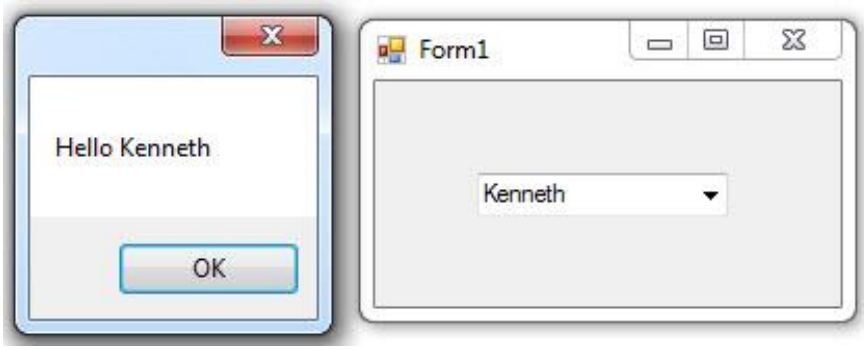
در پنجره‌ی StringCollectionEditor، نام‌هایی را که در تصویر زیر می‌بینید را، وارد کرده و بر روی OK کلیک کنید.



مقدار مشخصه‌ی Text کنترل ComboBox را به Choose a name تغییر دهید. بنابراین مقدار پیش فرض آن وقتی که هنوز آیتمی انتخاب نشده باشد "Choose a name" خواهد بود. بر روی ComboBox دوبار کلیک کنید. VS/VCE یک کنترل کننده‌ی رویداد (Event Handler) برای رویداد SelectedIndexChanged می‌سازد. کد زیر را در داخل کنترل کننده‌ی (Event Handler) رویداد وارد کنید.

```
string selectedName = comboBoxNames.SelectedItem.ToString();

MessageBox.Show("Hello " + selectedName);
```

برنامه را اجرا کرده و یک نام را انتخاب کنید. وقتی که شما نامی را انتخاب می‌کنید، برنامه بلافاصله یک پیغام را به شما نشان می‌دهد که در آن به نام انتخاب شده خوش آمدگویی می‌کند.



وقتی که شما یک نام را انتخاب می‌کنید، رویداد SelectedIndexChanged رخ می‌دهد. خاصیت SelectedItem تمامی داده‌های مربوط به آیتم انتخاب شده را، در بر دارد. از آنجایی که این خاصیت یک شیء (Object) را بر می‌گرداند، ما باید مقادیر آنرا به وسیله‌ی متد ToString() به رشته تبدیل کرده و در داخل یک متغیر String قرار دهیم (هرچند شما می‌توانید از خاصیت Text استفاده کنید ولی ما برای این مثال از SelectedItem استفاده کرده‌ایم). سپس ما یک متن که حاوی پیغام خروجی است را، نمایش می‌دهیم.

خاصیت SelectedIndex اندیس (Index) آیتم انتخاب شده را مشخص می‌کند. اندیس آیتم اول ۰ است و این اندیس برای آیتم‌های بعدی به اندازه‌ی یک واحد اضافه می‌شود. به طوریکه اندیس هر آیتم از آیتم قبلی خودش، یک واحد بیشتر است. اندیس آخرین آیتم (۱- تعداد آیتم‌ها)

است. بنابراین اگر ما ۱۰ آیتم داشته باشیم، اندیس آخرین آیتم ۹ است. اگر هیچ آیتمی انتخاب نشده باشد، اندیس آن برابر با ۱ خواهد بود. اگر شما می‌خواهید در زمان اجرای برنامه آیتم‌هایی را اضافه کنید، می‌توانید از خاصیت Items کنترل ComboBox استفاده کنید. خاصیت Items یک متد به نام Add() دارد که به وسیله‌ی آن می‌توانید به ComboBox آیتم‌های جدیدی را اضافه کنید.

```
string[] names = { "Johnny", "Kenneth", "Mark", "Ralph", "Sussie" };

//Add each names from the array to the combo box
foreach(string name in names)
{
    comboBoxNames.Items.Add(name);
}
```

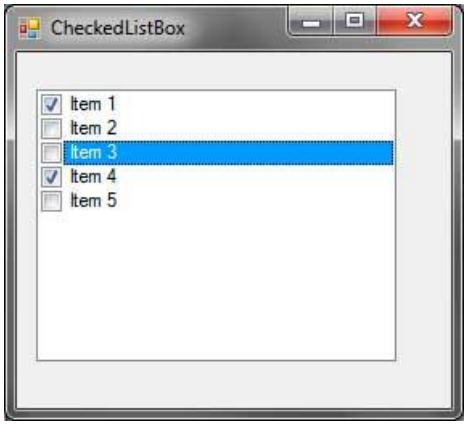
شما می‌توانید لیستی از نام‌ها را در یک آرایه‌ی رشته‌ای قرار دهید. سپس با استفاده از یک حلقه‌ی foreach آن نام‌ها را به ComboBox اضافه کنید. همچنین شما می‌توانید از خاصیت DataSource نیز استفاده کنید.

```
comboBoxNames.DataSource = names;
```

خاصیت DataSource می‌تواند یک مجموعه یا آرایه را قبول کرده و با آنها لیست آیتم‌های مربوط به خود را پر کند.

کنترل CheckedListBox

کنترل CheckedListBox شبیه کنترل ListBox است، با این تفاوت که در کنار هر آیتم از آن یک کنترل CheckBox نیز وجود دارد. شما می‌توانید هر آیتم را مانند ListBox انتخاب کنید، بعلاوه اینکه می‌توانید هر آیتم را توسط CheckBox کنار آن تیک بزنید.



خواص و متدهای کنترل ListBox در کنترل CheckedListBox نیز وجود دارند. اما کنترل CheckedListBox دارای خواصی منحصر بفرد نیز می‌باشد.

خاصیت	توضیح
CheckedItems	مجموعه‌ای از آیتم‌های تیک خورده است.
CheckOnClick	مشخص می‌کند که آیا جعبه کنار آیتمی که انتخاب شده است تیک بخورد یا نه؟

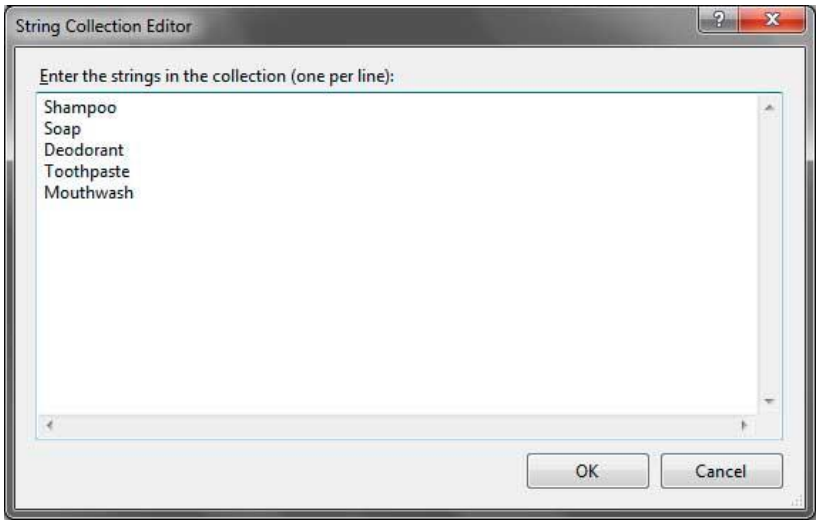
ThreeDCheckBoxes	حالت نمایش CheckBox را مشخص می‌کند، این خاصیت دو مقدار دارد. Flat: ظاهر چک باکس دو بعدی است. Normal: ظاهر چک باکس ۳ بعدی است.

در زیر متدهای منحصر به فرد کنترل CheckedListBox آمده است.

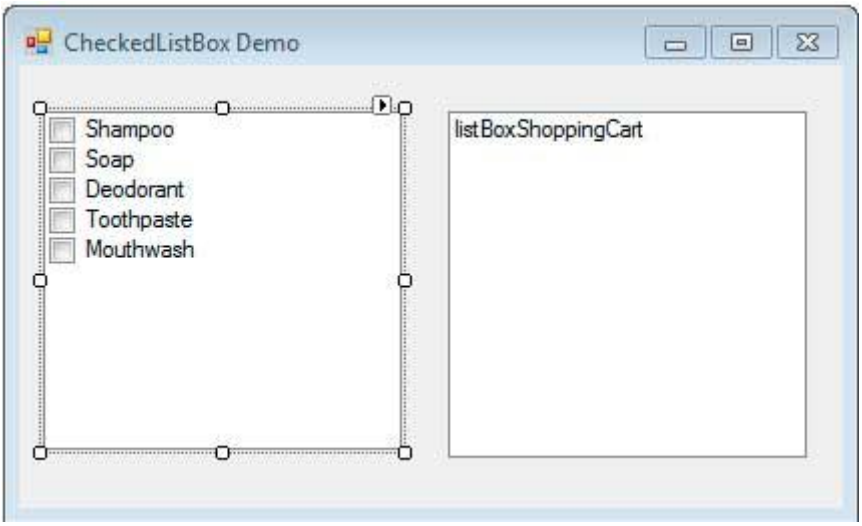
متد	توضیح
GetItemChecked	به ما می‌گوید که یک آیتمی با یک اندیس مشخص چک خورده است یا خیر.
GetItemCheckState	مقدار CheckState، آیتمی با یک اندیس مشخص را بر می‌گرداند.
SetItemChecked	آیتمی با اندیس مشخص را تیک می‌زند یا تیک آنرا بر می‌دارد.
SetItemCheckState	حالت چک، یک آیتم با اندیس مشخص را تعیین می‌کند.

حالا برنامه‌ای می‌سازیم که در آن یک کنترل CheckedListBox بکار رفته است. یک فرم را ساخته و یک کنترل CeckedListBox بر روی آن قرار دهید و مقدار خاصیت Name آنرا به checkedListBoxProducts تغییر دهید. حالا یک ListBox را به فرم اضافه کرده و مقدار خاصیت Name آنرا به listBoxShoppingCart تغییر دهید. فرم شما باید چیزی شبیه به این باشد.

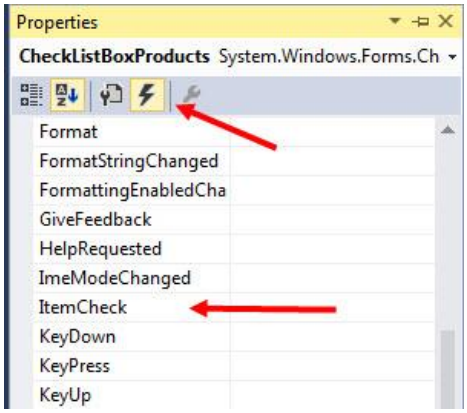
در پنجره‌ی خواص (Properties) خاصیت Items کنترل CheckedListBox را پیدا کنید. بر روی دکمه کنار آن کلیک کنید تا پنجره‌ی String Collection Editor باز شود. مقادیر زیر را به آن اضافه کنید.



بر روی OK کلیک کنید. حالا شما آیتم‌هایی را که به آن اضافه کرده‌اید را مشاهده می‌کنید.



رویداد پیش فرض CheckedListBox، رویداد SelectedIndexChanged است. که دقیقاً شبیه رویدادی با همین نام در ListBox است. حال ما می‌خواهیم آیتم‌های تیک خورده را به shopping cart اضافه کنیم. کنترل CheckListBox یک رویداد به نام ItemCheck دارد، زمانی این رویداد رخ می‌دهد که، حالت Check یکی از آیتم‌ها تغییر کند. از آنجایی که این رویداد، رویداد پیش‌فرض کنترل CheckedListBox نیست، ما نمی‌توانیم به سادگی با دوبار کلیک کردن بر روی این کنترل، این رویداد را ایجاد کنیم. برای اینکار ما باید به بخش Events در پنجره‌ی خواص (Properties) رفته و رویداد ItemCheck را پیدا کنیم.



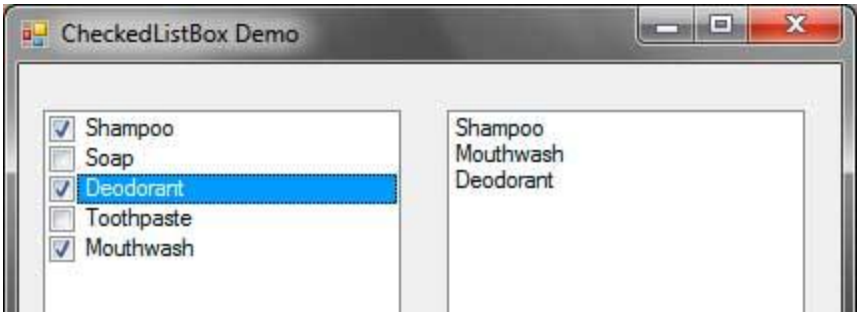
بر روی ItemCheck دوبار کلیک کرده تا یک کنترل کننده‌ی رویداد برای آن ساخته شود. حالا این کدها را به آن اضافه کنید.

```
private void checkedListBoxProducts_ItemCheck(object sender, ItemCheckEventArgs e)
{
    if (e.NewValue == CheckState.Checked)
        listBoxShoppingCart.Items.Add(checkedListBoxProducts.Items[e.Index]);
    else if (e.NewValue == CheckState.Unchecked)
        listBoxShoppingCart.Items.Remove(checkedListBoxProducts.Items[e.Index]);

}
```

در داخل کنترل کننده‌ی رویداد ItemCheck یک پارامتر دومی از نوع ItemCheckEventArgs نیز وجود دارد که خواص CurrentValue (مقدار فعلی)، NewValue (مقدار جدید) و Index (اندیس) را در بر می‌گیرد.

- خاصیت CurrentValue مقداری از خاصیت CheckState آن آیتم است که، رویداد ItemCheck را قبل از تغییر CheckState، راه اندازی می‌کند.
- خاصیت NewValue مقداری جدیدی است که، خاصیت CheckState آن آیتم به خود می‌گیرد.
- خاصیت Index، اندیس آیتم بر مبنای صفر است (یعنی اندیس‌ها از صفر شروع می‌شوند).

ما امتحان می‌کنیم اگر مقدار NewValue ی آیتم برابر با Checked بود، آن آیتم به Shopping Cart اضافه شود و اگر مقدار NewValue برابر با Unchecked بود، آیتم مورد نظر از Shopping Cart حذف شود. در تصویر زیر مشاهده می‌کنید که آیتم‌هایی که تیک زده شده‌اند، به Shopping Cart اضافه شده‌اند.



کنترل NumericUpDown

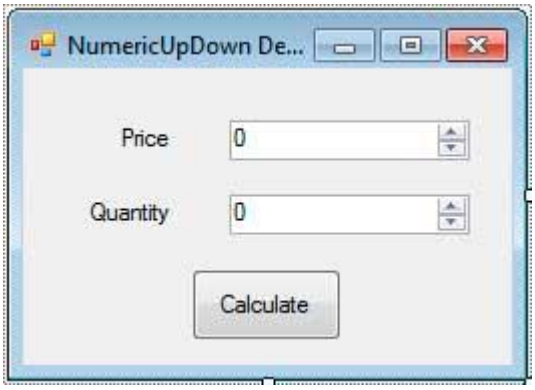
کنترل NumericUpDown عموماً برای دریافت اعداد از ورودی و محدود کردن کاربران برای وارد کردن مقادیر غیرعددی بکار می‌رود. کنترل NumericUpDown از لحاظ شکل ظاهری شبیه به کنترل TextBox است با این تفاوت که دکمه‌هایی به شکل پیکان در سمت چپ یا راست آن برای افزایش و یا کاهش مقدار کنترل وجود دارند.

مقدار عددی کنترل NumericUpDown می‌تواند توسط خاصیت Value مورد دستیابی و یا بازیابی قرار گیرد. نوع داده‌ی این خاصیت از نوع Decimal است. در جدول زیر برخی از خواص کنترل NumericUpDown را مشاهده می‌کنید.

خاصیت	توضیح
DecimalPlaces	تعداد ارقام اعشار را مشخص می‌کند.
Hexadecimal	مشخص می‌کند که کنترل NumericUpDown مقدارش را به وسیله‌ی کدهای HexaDecimal نمایش دهد یا خیر.
Increment	گام افزایش یا کاهش مقدار کنترل را مشخص می‌کند.
InterceptArrowKeys	اگر مقدار آن برابر با True باشد، شما می‌توانید با استفاده از کلیدهای مکان نمای کیبورد مقدار را افزایش و یا کاهش دهید.
Maximum	بالاترین مقداری را که این کنترل می‌تواند در خود جای دهد را مشخص می‌کند.
Minimum	کمترین مقداری را که این کنترل می‌تواند در خود جای دهد را مشخص می‌کند.
ThousandsSeperator	برای جدا کردن ارقام هر رده (یکان, دهگان, صدگان, ...) بکار می‌رود. (برای مثال 1,000)
UpDownAlign	مشخص می‌کند که دکمه‌های پیکانی در چه مکانی قرار بگیرند. اگر مقدار آن Right باشد، آنها در سمت راست و اگر مقدار آن Left باشد، دکمه‌های پیکانی در سمت چپ کنترل NumericUpDown قرار می‌گیرند.
Value	مقدار کنترل NumericUpDown control را مشخص می‌کند.

رویداد پیش فرض کنترل NumericUpDown رویداد ValueChanged است، این رویداد زمانی اتفاق می‌افتد که مقدار خاصیت Value این کنترل تغییر کند. حال می‌خواهیم یک برنامه بسازیم که کنترل NumericUpDown در آن بکار رفته باشد. یک فرم را ایجاد کرده و نام آنرا به

NumericUpDown تغییر دهید. دو برچسب (lable) و دو کنترل NumericUpDown بر روی آن قرار دهید. مقدار خاصیت Text بر چسب‌ها را به "Price" و "Quantity" تغییر دهید. یک دکمه (Button) را به فرم اضافه کرده و عنوان (Caption) آنرا به "Calculate" تغییر دهید. کنترل‌ها را مانند شکل ۲ بر روی فرم جایگذاری کنید.



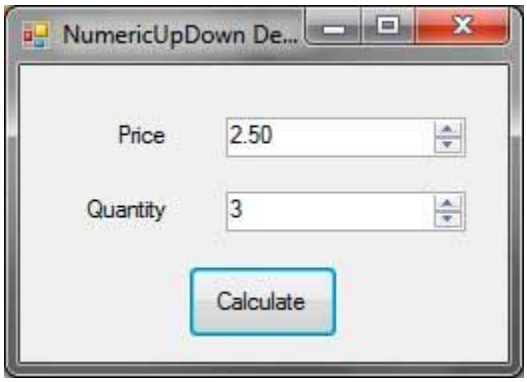
بر اساس جدول زیر خواص کنترل‌های خود را تغییر دهید.

کنترل	خاصیت	مقدار
button1	Name	buttonCalculate
numericUpDown1	Name	numericUpDownPrice
	Decimal	2
	Increment	0.50
	Maximum	10000
numericUpDown2	Name	numericUpDownQuantity
	Maximum	100

مقدار خاصیت Decimal کنترل numericUpDownPrice برابر با ۲ قرار داده شده، بنابراین این کنترل می‌تواند ۲ رقم اعشار دقت داشته باشد. مقدار خاصیت Increment نیز برابر با 0.50 قرار داده‌ایم، بنابراین افزایش و یا کاهش مقدار به اندازه 0.50 واحد کم یا زیاد می‌شود. مقدار خاصیت Maximum، ۱۰۰۰۰ است، پس قیمت (Price) به آن محدود می‌شود و کاربر نمی‌تواند رقمی بالاتر از آنرا وارد کند. از آنجاییکه نهایت مقدار کنترل numericUpDownQuantity را در جدول بالا ۱۰۰ قرار داده‌ایم بنابراین شما می‌توانید فقط مقادیری نهایتاً تا ۱۰۰ را در آن وارد کنید. بر روی دکمه (Button) دوبار کلیک کنید تا یک کنترل کننده‌ی رویداد Click به آن اضافه شود. کدهای زیر را به آن اضافه کنید.

```
private void buttonCalculate_Click(object sender, EventArgs e)
{
    decimal price = numericUpDownPrice.Value;
    int quantity = (int)numericUpDownQuantity.Value;
    decimal total;

    total = price * quantity;

    MessageBox.Show(String.Format("The total price is {0:C}", total));
}
```

برنامه را اجرا کنید. مقدار کنترل numericUpDownPrice را به وسیله‌ی دکمه‌های پیکانی کاهش یا افزایش دهید. مشاهده می‌کنید مقدار آن به اندازه 0.50 واحد افزایش و یا کاهش پیدا می‌کند.



بر روی دکمه‌ی Calculate کلی کنید، این دکمه با ضرب کردن مقدار Quantity در Price، قیمت کل را بدست می‌آورد.

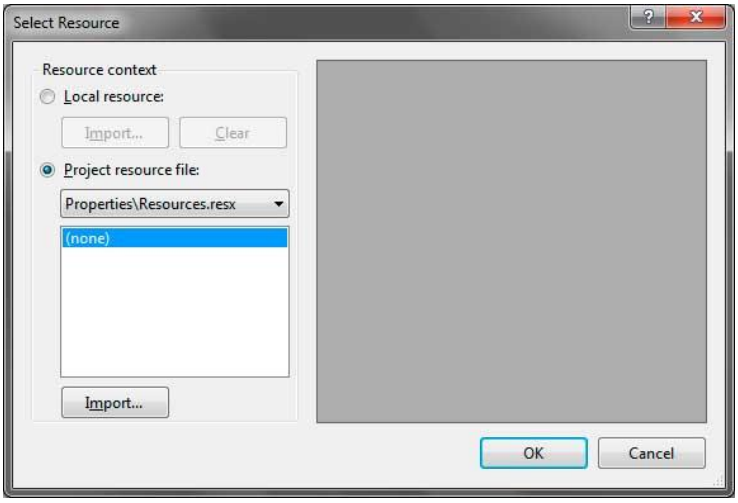
کنترل PictureBox

به وسیله‌ی کنترل PictureBox می‌توانید یک تصویر بر روی فرم قرار دهید. کار اصلی این کنترل نمایش دادن یک تصویر است. تمامی کاری که شما باید انجام دهید این است که، عکس مورد نظر خود را انتخاب کنید تا Visual Studio/VCE آنرا به پروژه‌ی شما وارد کند. شما می‌توانید از فرمت‌های گوناگونی مثل JPEG، PNG، BMP، GIF استفاده کنید.

در جدول زیر برخی از خواص مفید این کنترل را مشاهده می‌کنید.

خواص	توضیحات
ErrorImage	تصویری که در موقع بالا نیامدن عکس مورد نظر به کاربر نمایش داده می‌شود.
Image	عکسی که توسط این کنترل نمایش داده می‌شود.
ImageLocation	مسیر تصویری که توسط PictureBox نمایش داده می‌شود.
InitialImage	تصویری که در هنگام بارگذاری (Load) تصویر اصلی به کاربر نمایش داده می‌شود.
SizeMode	به ما می‌گوید که تصویر چگونه نمایش داده خواهد شد. این خاصیت مقدار خود را از System.Windows.Forms.PictureBoxSizeMode می‌گیرد.
WaitOnLoad	وقتی مقدار آن برابر با True باشد، همه‌ی کارهای (تراکنش‌ها) فرم را تا زمانی که تصویر به صورت کامل بارگذاری شود مسدود می‌کند.

برای نمایش یک تصویر در کنترل PictureBox، چند روش وجود دارد. شما می‌توانید به پنجره‌ی خواص (Properties) رفته و خاصیت Image را پیدا کنید. بر روی دکمه‌ی سمت راست آن کلیک کرده تا پنجره‌ی Select Resource باز شود.



در پنجره‌ی باز شده، شما دو انتخاب دارید. یکی اینکه عکس مورد نظرتان را که در داخل یک درایو (Local resource) موجود است انتخاب کنید و دیگری اینکه عکس را قبلاً در داخل فایل پروژه (project's resouce file) قرار داده‌اید، وارد برنامه کنید. هر دوی آنها یک دکمه‌ی Import دارند که به شما اجازه می‌دهند که تصویر مورد نظر خود را به فرم اضافه کنید. وقتی که شما تصویر مورد نظر خود را انتخاب کردید، VS/VCE تصویر را به کنترل و یا منابع پروژه (Project Resource File) اضافه می‌کند.

حال تصویر شما در داخل PictureBox نمایش داده می‌شود. به یاد داشته باشید که، شما می‌توانید از خاصیت ImageLocation استفاده کنید که از شما مسیر تصویر مورد نظر در داخل هارد را درخواست می‌کند. همچنین شما می‌توانید از مسیر یک تصویر که بر روی صفحات وب قرار دارد استفاده کنید. ممکن است نمایش عکس با آن چیزی که شما انتظارش را دارید متفاوت باشد. اگر تصویر بزرگ‌تر از سایز کنترل PictureBox

باشد، فقط قسمتی از آن نمایش داده می‌شود. شما می‌توانید از خاصیت SizeMode برای تغییر سایز یا مکان تصویر در داخل کنترل استفاده کنید.

این خاصیت از مقادیر شمارشی System.Windows.Forms.PictureBoxSizeMode که در جدول زیر مشاهده می‌کنید استفاده می‌کند.

PictureBoxSizeMode	توضیحات
Normal	تصویر در قسمت بالا چپ جای می‌گیرد، و اگر تصویر از اندازه‌ی PictureBox بیشتر باشد، تصویر برش داده می‌شود (فقط قسمتی از آن نمایش داده می‌شود).
StretchImage	سایز تصویر را متناسب با اندازه PictureBox تغییر می‌دهد و تمام عکس را نمایش می‌دهد.
AutoSize	سایز PictureBox را متناسب با اندازه‌ی تصویر تغییر می‌دهد و قسمتی از عکس را نمایش می‌دهد.
CenterImage	تصویر در وسط PictureBox قرار می‌گیرد. اگر سایز تصویر از سایز PictureBox بیشتر باشد، تصویر برش داده می‌شود (فقط قسمتی از وسط کس در آن نمایش داده می‌شود).
Zoom	همه عکس را طوری نمایش می‌دهد که شکل آن در کنترل تغییر نکند. مثلاً کشیده نشود.

رویداد پیش‌فرض این کنترل، رویداد Click است، و زمانی رخ می‌دهد که بر روی عکس یا کنترل کلیک شود.

کنترل LinkLable

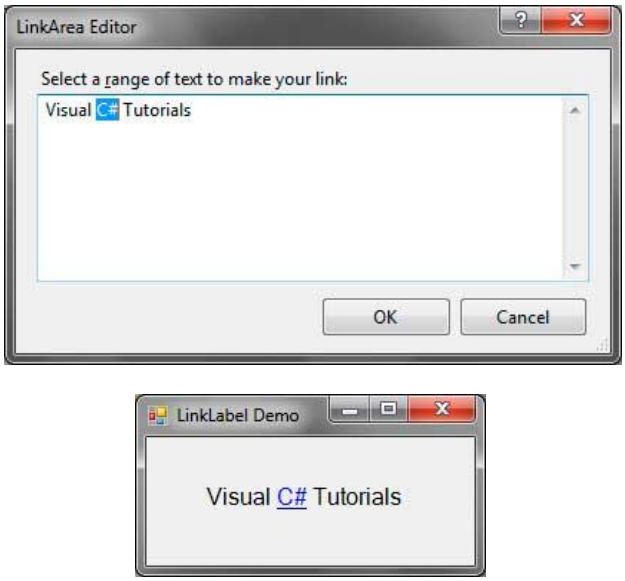
کنترل LinkLable شبیه کنترل Lable (برچسب) است، با این تفاوت که این کنترل یک زیر خط دارد که شبیه به یک لینک در صفحات وب است. کنترل LinkLable می‌تواند برای لینک کردن به فایل‌ها، دایرکتوری‌ها، و حتی صفحات وب استفاده شود. وقتی نشانگر ماوس را بر روی LinkLable ببرید، آیکن نشانگر ماوس به شکل یک دست تغییر می‌کند.

خواص این کنترل را در جدول زیر مشاهده می‌کنید.

خاصیت	توضیحات
BorderStyle	حالت حاشیه‌ی دور برچسب را مشخص می‌کند.
FlatStyle	ظاهر این کنترل را تعیین می‌کند. وقتی مقدار آن برابر با Popup باشد، اگر بر روی آن کمی منتظر بمانید دکمه‌ی آن برجسته می‌شود.
Links	به شما اجازه می‌دهد که در داخل یک متن چند لینک داشته باشید.

LinkArea	متن لینک مورد نظر را مشخص می‌کند.
LinkColor	رنگ لینک مشاهده نشده را مشخص می‌کند.
LinkVisited	وقتی مقدار آن برابر با True باشد، رنگ لینک با رنگ خاصیت VisitedLinkColor عوض می‌شود.
TextAlign	مکان متن داخل کنترل را مشخص می‌کند.
VisitedLinkColor	رنگ لینک مشاهده شده را مشخص می‌کند.

خاصیت LinkArea به شما اجازه می‌دهد که فقط قسمتی از متن را به عنوان لینک استفاده کنید. خاصیت LinkLabel به دو مقدار نیاز دارد:۱- اندیس شروع ۲- طول. برای مثال شما در شکل بالا می‌خواهید فقط قسمت "C#" به عنوان لینک نمایش داده شود، برای اینکار باید مقدار اندیس را برابر با ۷ و طول آنرا برابر با ۲ قرار دهید. همچنین شما می‌توانید بر روی دکمه‌ی سمت راست خاصیت LinkArea در پنجره‌ی خواص (Properties) کلیک کنید تا پنجره‌ی LinkArea Editor نمایش داده شود. در پنجره‌ی باز شده به راحتی قسمتی را که می‌خواهید به عنوان لینک نمایش داده شود را انتخاب کرده و دکمه‌ی OK را بزنید.



خاصیت Links به شما اجازه می‌دهد که در داخل یک متن چند لینک داشته باشید. متاسفانه، شما از طریق پنجره‌ی خواص (Properties) نمی‌توانید به این خاصیت دسترسی داشته باشید، برای دسترسی به این قابلیت، باید کدها و مقادیر آنها را به صورت دستی وارد کنید. شما می‌توانید کدهای زیر را در رویداد Load فرم قرار دهید.

```
private void LinkLabelForm_Load(object sender, EventArgs e)
{
    linkLabel1.Links.Add(new LinkLabel.Link(0, 6));
    linkLabel1.Links.Add(new LinkLabel.Link(10, 9));
}
```

خاصیت Links مجموعه‌ای از اشیاء LinkLabel.Link می‌باشد، در نتیجه از متد ()Add و ایجاد یک نمونه از کلاس LinkLabel.Link استفاده کرده‌ایم (به این نکته توجه کنید که Link کلاسی است که در داخل کلاس LinkLable قرار دارد). کنترل LinkLable باید دارای یک کنترل کننده‌ی رویداد برای رویداد LinkClicked باشد. مکانی که قرار است با کلیک بر روی این کنترل به آنجا منتقل شوید، در داخل کنترل کننده‌ی رویداد مشخص می‌شود.

```
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    System.Diagnostics.Process.Start("http://w3-farsi.com");
    linkLabel1.LinkVisited = true;
}
```

شما می‌توانید از متد ()System.Diagnostics.Process.Start برای باز کردن مرورگر و مرور صفحات وب استفاده کنید. سپس مقدار خاصیت LinkVisited را به True تغییر دادیم تا زمانی که یک لینک مشاهده شد، رنگ آن تغییر کند. به یاد داشته باشید که شما همچنین می‌توانید یک مسیر را بر روی هارد انتخاب کنید. اگر شما در داخل کنترل LinkLabel بیش از یک لینک دارید، باید از خاصیت Links از LinkLabelLinkClickedEventArgs استفاده کنید، تا این رویداد بداند که دقیقاً بر روی کدام لینک کلیک شده است.


```
private void LinkLabelForm_Load(object sender, EventArgs e)
{
    LinkLabel.Link googleLink = new LinkLabel.Link(0, 6);
    LinkLabel.Link bingLink = new LinkLabel.Link(11, 4);

    googleLink.Name = "Google";
    bingLink.Name = "Bing";

    linkLabel1.Links.Add(googleLink);
    linkLabel1.Links.Add(bingLink);
}
```

ما یک نمونه‌های دیگر از LinkLabel با مکانهای مربوط به خودشان ایجاد کردیم. حال هر نام را، به لینک مربوط به آن از طریق خاصیت Name وصل می‌کنیم.

```
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    switch (e.Link.Name)
    {
        case "Google":
            System.Diagnostics.Process.Start("http://google.com");
            break;
        case "Bing":
            System.Diagnostics.Process.Start("http://bing.com");
            break;
    }
}
```

در داخل کنترل کننده‌ی رویداد LinkClicked کنترل LinkLabel، مکان لینکی که فشرده شده است را با استفاده از خاصیت Link دریافت می‌کنیم. و در آخر تست می‌کنیم که بر روی نام کدام لینک کلیک شده است تا مرورگر سایت مربوط به آن را باز کند.

کل کد:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication13
{
    public partial class LinkLabelForm : Form
    {
        public LinkLabelForm()
        {
            InitializeComponent();
        }

        private void LinkLabelForm_Load(object sender, EventArgs e)
        {
            LinkLabel.Link googleLink = new LinkLabel.Link(0, 6);
            LinkLabel.Link bingLink = new LinkLabel.Link(11, 4);

            googleLink.Name = "Google";
            bingLink.Name = "Bing";

            linkLabel1.Links.Add(googleLink);
            linkLabel1.Links.Add(bingLink);
        }

        private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
        {
            System.Diagnostics.Process.Start("http://visualcsharptutorials.com");
            linkLabel1.LinkVisited = true;

            switch (e.Link.Name)
            {
```

```
                case "Google":
                    System.Diagnostics.Process.Start("http://google.com");
                    break;
                case "Bing":
                    System.Diagnostics.Process.Start("http://bing.com");
                    break;
            }
        }
    }
}
```

کنترل MonthCalendar

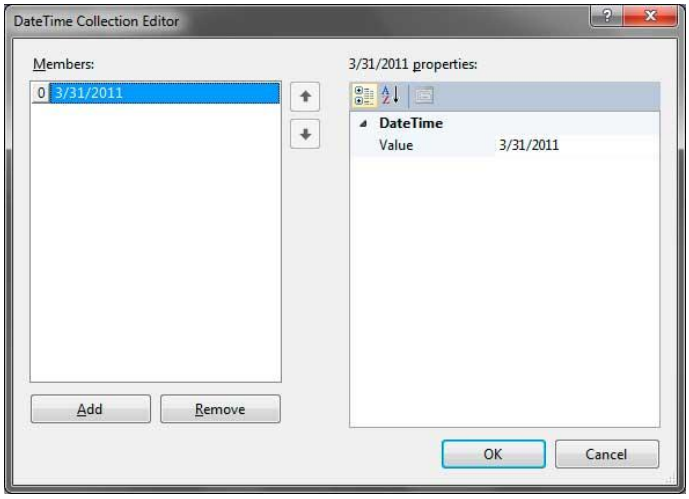
کنترل MonthCalendar (System.Windows.Forms.MonthCalendar) شبیه به یک تقویم است، و یک ماه را به همراه روزهای آن نشان می‌دهد. این کنترل به شما اجازه می‌دهد که یک ماه و یک تاریخ را انتخاب کنید، برای انتخاب یک ماه بر روی پیکآنهای سمت راست و چپ این کنترل کلیک کنید تا به ماههای بعدی و قبلی ماه جاری دسترسی داشته باشید.

با کلیک بر روی نام ماه، تمامی ماه‌های سال جاری نمایش داده می‌شوند. اگر مجدداً بر روی آن کلیک کنید، تمامی سال‌های دهه‌ی اخیر و اگر برای بار سوم بر روی آن کلیک کنید تمامی سال‌های قرن حاضر را به شما نمایش می‌دهد. شما می‌توانید با استفاده از خواص این کنترل آنرا سفارشی کنید.

خاصیت	توضیح
AnnualyBoldedDates	مجموعه‌ای از تاریخ‌های مشخص یک سال، در کنترل MonthCalendar به صورت ضخیم (Bold) نمایش داده می‌شوند.
BoldedDates	مجموعه‌ای از تاریخ‌های غیر تکراری را به صورت ضخیم نمایش می‌دهد.

CalendarDimensions	تعداد سطرها و ستون‌های ماه‌هایی که نمایش داده می‌شوند را مشخص می‌کند.
FirstDayOfWeek	مشخص می‌کند که کدام روز اولین روز هفته باشد.
MaxDate	حداکثر تاریخ قابل قبول برای این کنترل را مشخص می‌کند.
MaxSelectionCount	حداکثر روزهایی را که کاربر می‌تواند به طور همزمان انتخاب کند را مشخص می‌کند.
MinDate	حداقل تاریخ قابل قبول برای این کنترل را مشخص می‌کند.
MonthlyBoldedDates	مجموعه‌ای از تاریخ‌های مشخص در یک ماه، که به طور ماهانه به صورت ضخیم نمایش داده می‌شوند.
ScrollChange	تعداد ماه‌هایی را که با هر بار کلیک بر روی فلش‌ها حرکت می‌کنند.
SelectionEnd	وقتی کاربر یک بازه از تاریخ را انتخاب کند (مثلاً چند روز در یک ماه)، این خاصیت آخرین تاریخ در این بازه را مشخص می‌کند.
SelectionRange	اگر کاربر یک بازه از تاریخ را انتخاب کند، این خاصیت تمامی تاریخ‌های درون این بازه را در خود جای می‌دهد.
SelectionStart	وقتی کاربر یک بازه از تاریخ را انتخاب کند، این خاصیت اولین تاریخ در این بازه را مشخص می‌کند.
ShowToday	مشخص می‌کند که تاریخ امروز در قسمت پایین این کنترل نمایش داده شود یا خیر.
ShowTodayCircle	اگر مقدار آن برابر با True قرار گیرد، تاریخ امروز در داخل یک کادر مربعی و یا یک کادر دایره‌ای قرار می‌گیرد.
ShowWeekNumbers	مشخص می‌کند که شماره‌ی هفته در سمت چپ هر سطر کنترل نمایش داده شود یا خیر.
TodayDate	تاریخی که توسط کنترل MonthCalendar به عنوان تاریخ امروز استفاده می‌شود.

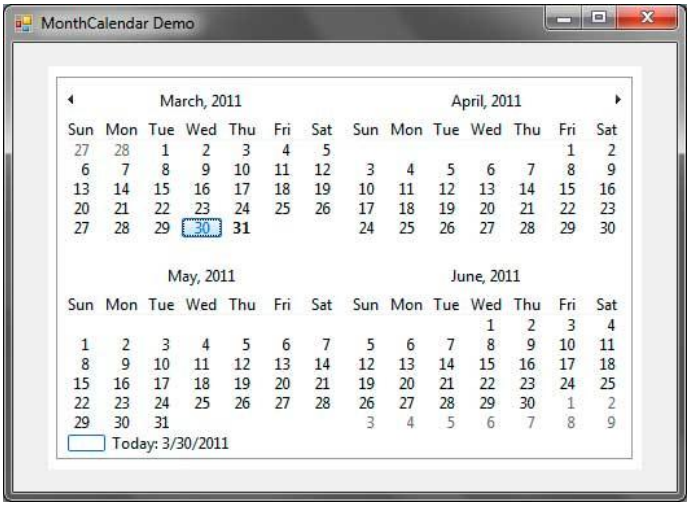
می‌توانید تاریخ‌های مشخصی را به صورت سالانه ضخیم کنید. برای مثال، مشخص کنید که بیست و پنجمین روز دسامبر برای نشان دادن روز کریسمس ضخیم شود. برای اینکار از خاصیت AnnuallyBoldedDates که مجموعه‌ای از تاریخ‌ها را قبول می‌کند استفاده می‌کنیم. به پنجره‌ی خواص (Properties) رفته و این خاصیت را پیدا کرده و بر روی فلش نوار کرکره‌ای آن کلیک کنید. پنجره‌ی DateTime Collection به شما نشان داده خواهد شد.



به سادگی با کلیک کردن بر روی دکمه‌های Add و Remove تاریخ‌های مورد نظر خود را اضافه و یا حذف کنید. در این پنجره بر روی یک آیتم کلیک کرده و تاریخ آنرا به وسیله‌ی خاصیت Value در قسمت Properties در سمت راست آن مشخص کنید. با کلیک بر روی نوار کرکره‌ای یک انتخاب کننده‌ی تاریخ به شما نمایش داده می‌شود که شما به وسیله‌ی آن می‌توانید، تاریخ مورد نظر خود را انتخاب کنید، همچنین به جای اینکار شما به راحتی می‌توانید یک تاریخ را به دلخواه در داخل آن تایپ کنید. خاصیت MonthlyBoldedDates، تاریخ‌هایی را به صورت ماهانه مشخص شده‌اند را به صورت ماهانه ضخیم می‌کند، در حالیکه خاصیت BoldedDates یک تاریخ را مشخص می‌کند که فقط یکبار ضخیم می‌شود و به صورت ماهانه یا سالانه تکرار نمی‌شوند. به طور پیش فرض می‌توانید ۷ تاریخ پشت سر هم را انتخاب کنید. اینکار به وسیله‌ی دکمه‌ی Shift و انتخاب کردن اولین و آخرین تاریخ انجام می‌شود.

شما می‌توانید مقدار خاصیت MaxSelectionCount را تغییر دهید. همه‌ی تاریخ‌هایی که بین این دو تاریخ انتخاب شده وجود دارند، انتخاب خواهند شد. اگر تعداد تاریخ‌های انتخاب شده از حد مقدار خاصیت MaxSelectionCount تجاوز کند، به طور اتوماتیک تعداد تاریخ‌های انتخاب شده برابر با مقدار این خاصیت قرار می‌گیرد (برای مثال اگر شما ۸ تاریخ را انتخاب کنید، و مقدار خاصیت MaxSelectionCount برابر با ۷ باشد، به طور اتوماتیک ۷ تاریخ انتخاب خواهند شد).

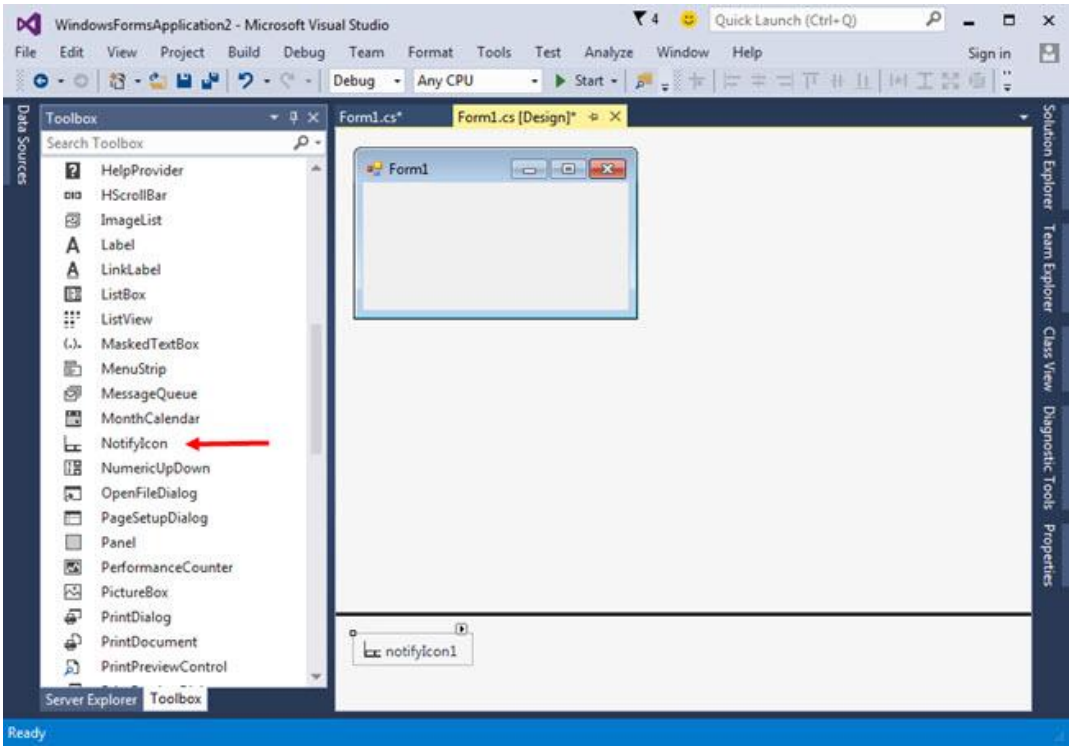
شما می‌توانید از خواص SelectionStart، SelectionEnd و SelectionRange برای برگرداندن تاریخ‌های انتخاب شده استفاده کنید. اگر فقط یک تاریخ انتخاب شده بود، به سادگی می‌توانید از SelectionStart برای برگرداندن آن استفاده کنید. به طور پیش‌فرض فقط یک ماه نمایش داده می‌شود، که فقط یک سطر و یک ستون دارد. شما می‌توانید این حالت را به وسیله‌ی خاصیت CalendarDemensions تغییر دهید. برای مثال، در تصویر زیر یک کنترل MonthCalendar، ۲ در ۲ را مشاهده می‌کنید که چهار ماه را به طور همزمان نمایش می‌دهد.



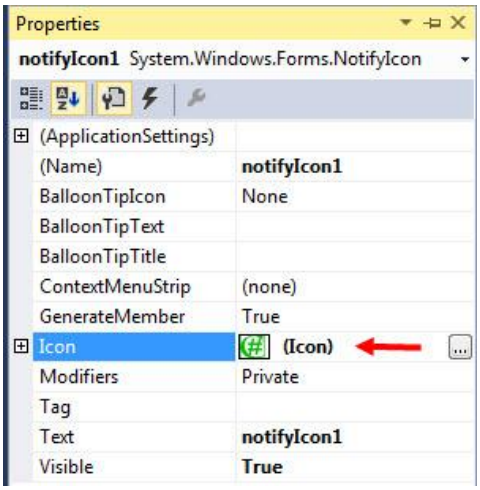
شما می‌توانید رویدادهای DateSelected و DateChanged را کنترل کنید. رویداد DateSelected زمانی که یک تاریخ یا بازه‌ای از تاریخ‌ها انتخاب می‌شود، اتفاق می‌افتد. رویداد DateChanged زمانی که کاربر، تاریخ و یا بازه‌ای از تاریخ‌های انتخاب شده را تغییر می‌دهد، اتفاق می‌افتد.

کنترل NotifyIcon

از کنترل NotifyIcon برای نمایش آیکون برنامه در قسمت System tray استفاده می‌شود. یک برنامه ویندوزی ایجاد کرده و کنترل Notify Icon را به آن اضافه کنید.



برای نمایش آیکون این کنترل در System tray می‌توان به صورت زیر یک آیکون به آن اختصاص داد:



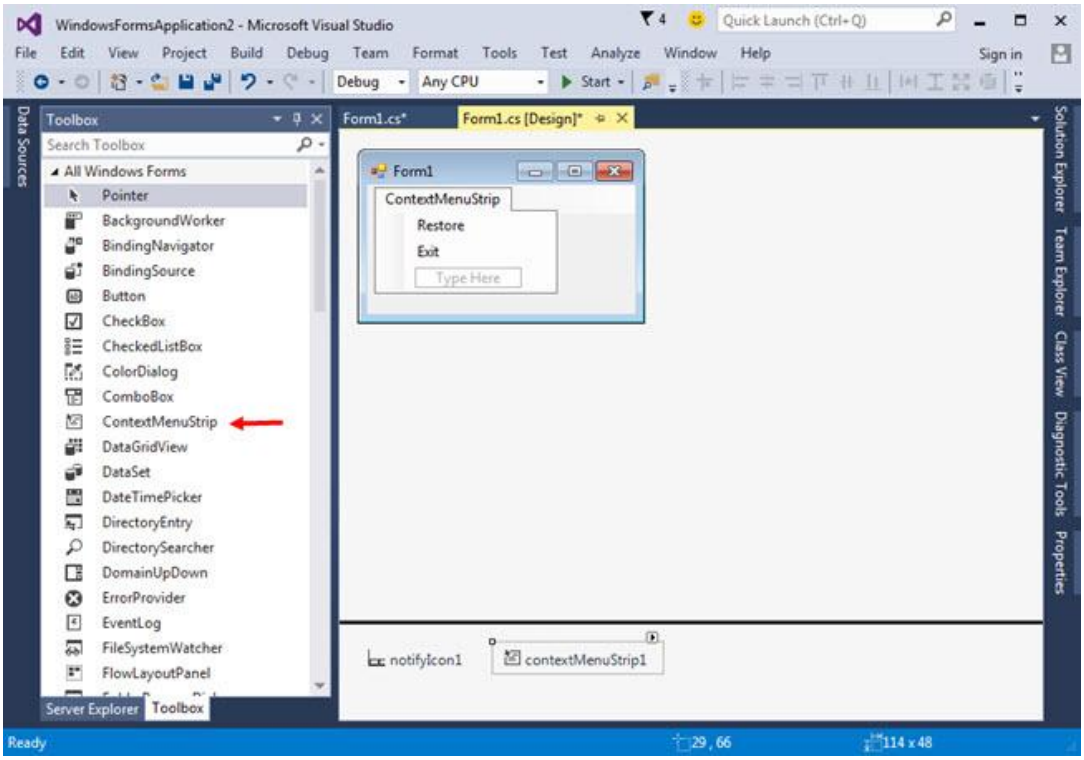
بعد از اضافه کردن کنترل، برای کنترل کننده رویداد Resize فرم، کد زیر را بنویسید:

```
private void Form1_Resize(object sender, EventArgs e)
{
    if (FormWindowState.Minimized == this.WindowState)
        Hide();
}
```

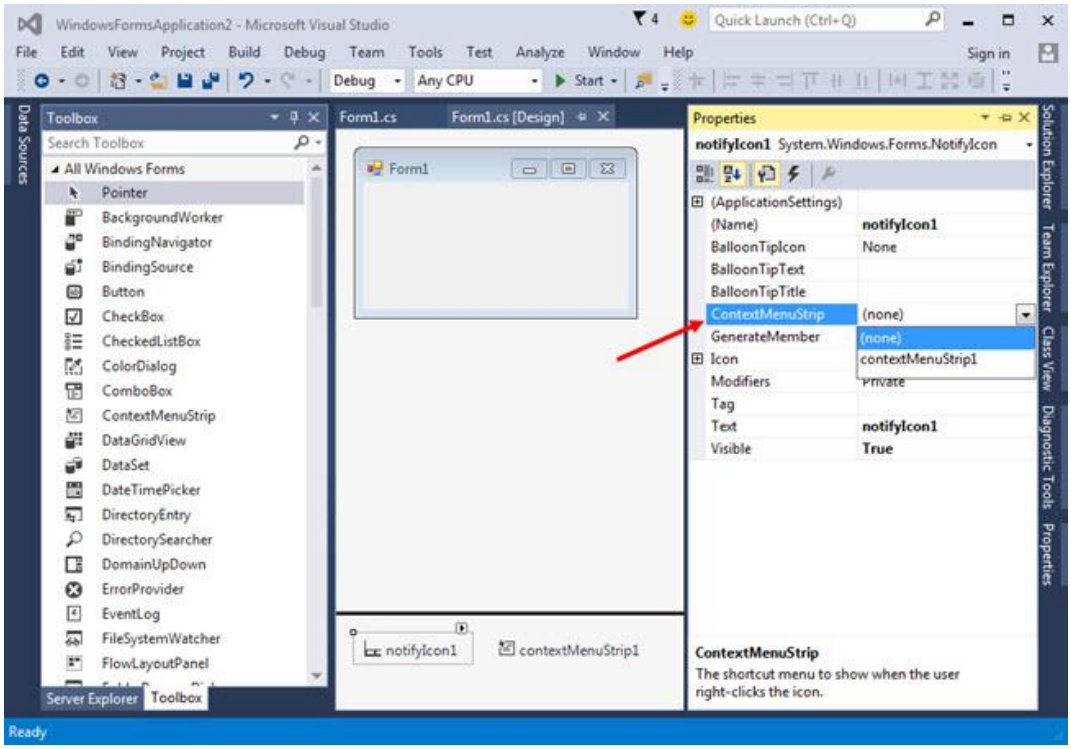
کد فوق باعث می‌شود که فرم هنگام Minimize شدن در taskbar نمایش داده نشود. بر روی رویداد DoubleClick کنترل NotifyIcon نیز دو بار کلیک کرده و کد زیر را بنویسید:

```
private void notifyIcon1_DoubleClick(object sender, EventArgs e)
{
    Show();
    WindowState = FormWindowState.Normal;
}
```

این کد نیز باعث maximize شدن فرم یا برگشت به حالت قبل آن می‌شود. می‌توان یک منو هم به این کنترل اضافه کرد. برای این کار یک کنترل ContextMenuStrip به فرم اضافه کنید و گزینه‌هایی مانند شکل زیر به آن اختصاص دهید:



این منو را از طریق خاصیت `contextMenuStrip` کنترل `NotifyIcon` به آن اضافه می‌کنیم:



کدهای زیر را هم به گزینه‌های منو با دوبار کلیک بر روی هر کدام از آنها اضافه می‌کنیم:

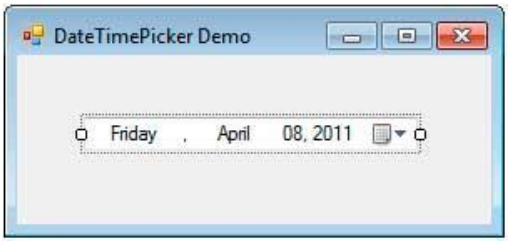
```
private void restoreToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    Show();
    WindowState = FormWindowState.Normal;
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

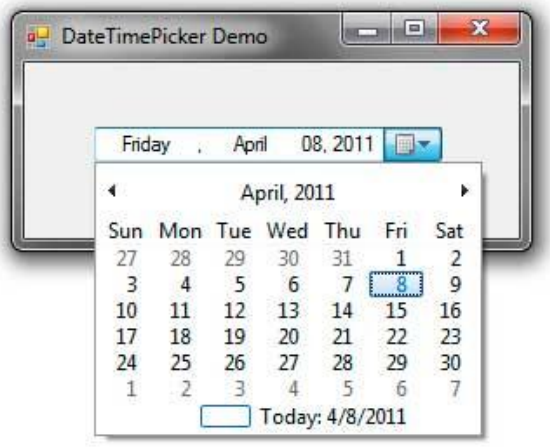
برنامه را اجرا و نتیجه را مشاهده کنید:

کنترل DateTimePicker

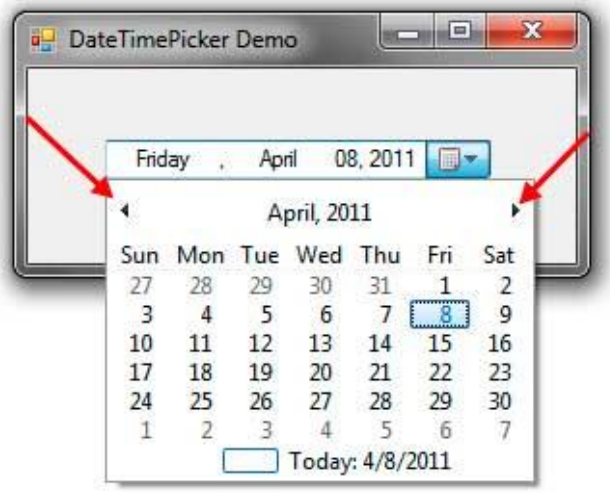
کنترل DateTimePicker(System.Windows.Forms.DateTimePicker) برای انتخاب یک تاریخ بکار می‌رود. این کنترل به طور پیش‌فرض به شکل یک ComboBox که یک آیکن تاریخ در سمت راست آن قرار دارد، است. شما می‌توانید هر کدام از اجزای تاریخ مانند ماه را انتخاب کرده و به وسیله‌ی کلیدهای مکان نما آنها را تنظیم کنید.



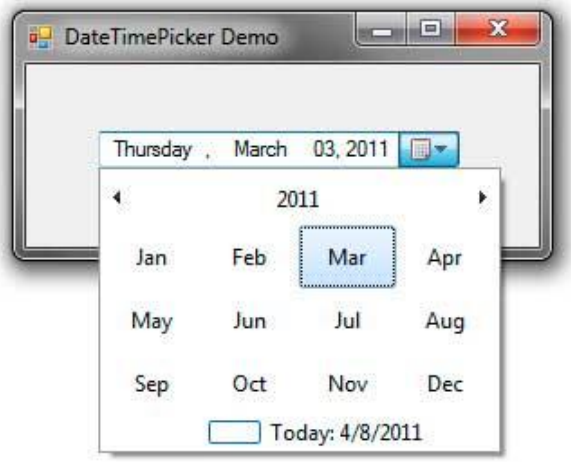
به طور پیش‌فرض، این کنترل تاریخ جاری یا تاریخ انتخاب شده را نمایش می‌دهد. در زمان اجرای برنامه با کلیک بر روی آیکن تاریخ، یک تقویم به شما نمایش داده می‌شود، که شما از طریق آن می‌توانید یک تاریخ را انتخاب کنید.



شما از طریق این تقویم می‌توانید یک تاریخ مشخص را انتخاب کنید. همچنین به وسیله پیکآنهای پیمایش در سمت چپ و راست آن می‌توانید به ماه بعد و قبل بروید.



نمای ابتدایی تقویم، تمامی روزهای ماه جاری را به همراه بعضی از روزهای ماه‌های قبل و بعد را نمایش می‌دهد. با کلیک بر روی عنوانی که ماه و سال را نمایش می‌دهد، می‌توانید نما را تغییر داده و همه‌ی ماههای سال جاری را نمایش دهید.



با یکبار کلیک کردن روی سال، تمامی سال‌های دهه‌ی جاری و با دوباره کلیک کردن روی آن به حالت اولیه باز می‌گردد. در جدول زیر برخی از خواصی را که به شما اجازه می‌دهد که رفتار و ظاهر این کنترل را تغییر دهید را مشاهده می‌کنید.

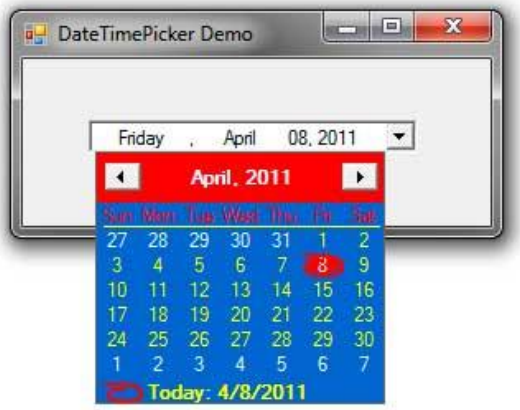
خاصیت	توضیحات
`CalendarFont`	نوع قلم تقویم را مشخص می‌کند.
`CalendarForeColor`	رنگ پیش زمینه‌ی تقویم را مشخص می‌کند.
`CalendarMonthBackground`	پشت زمینه‌ی ماه تقویم را مشخص می‌کند.
`CalendarTitleBackColor`	رنگ پشت زمینه‌ی عنوان تقویم را مشخص می‌کند.

CalendarTitleForeColor	رنگ پیش زمینه‌ی عنوان تقویم را مشخص می‌کند.
Checked	برای کار با این خاصیت باید مقدار خاصیت ShowCheckBox، False باشد. وقتی مقدار این خاصیت برابر با True باشد، تاریخ انتخاب شده می‌تواند تغییر و یا بهنگام سازی شود، و اگر مقدار آن برابر با False باشد، تاریخ انتخاب شده نمی‌تواند تغییر کند.
CustomFormat	یک قالب سفارشی رشته‌ای که برای نمایش قالب تاریخ استفاده می‌شود را قبول می‌کند.
Format	نوع فرمت (قالب) تاریخی که توسط کنترل نمایش داده می‌شود را مشخص می‌کند.
MaxDate	حداکثر تاریخ و ساعتی که در کنترل می‌تواند انتخاب شود را مشخص می‌کند.
MinDate	پایین‌ترین تاریخ و ساعتی که در کنترل می‌تواند انتخاب شود را مشخص می‌کند.
ShowCheckBox	اگر مقدار آن برابر با True باشد، یک تکست باکس در قسمت سمت چپ کنترل نمایش داده می‌شود. وقتی مقدار آن Checked باشد، تاریخ انتخاب شده می‌تواند تغییر کند. و وقتی مقدار آن Uncheck باشد تاریخ انتخاب شده نمی‌تواند تغییر کند.
ShowUpDown	وقتی مقدار آن True باشد، یک دکمه‌ی بالا-پایین به جای دکمه‌ی نوار کرکره‌ای قرار می‌گیرد. در این حالت شما نمی‌توانید به تقویم دسترسی داشته باشید. به جای آن، باید اجزای تاریخ را انتخاب کرده و با استفاده از این دکمه‌ی بالا-پایین آنها را تنظیم کنید.
Value	تاریخ انتخاب شده‌ی جاری را نمایش می‌دهد.

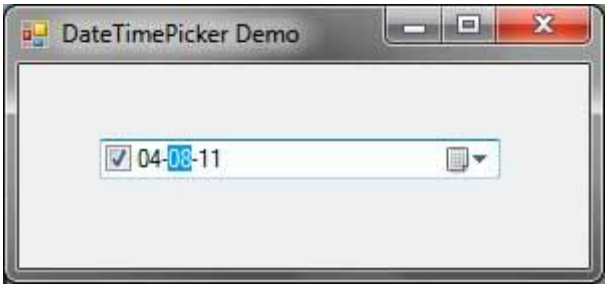
اگر از ویندوز Vista یا ۷ و نیز از تم‌هایی (Theme) مانند Aero استفاده کنید، تأثیر خواصی که مربوط به دستکاری رنگ تقویم هستند از بین می‌رود. برای مشاهده‌ی نتیجه‌ی تغییر رنگ تقویم، باید جلوه‌های ویژه (Visual Styles) را غیر فعال کنید. برای درک این مطلب، به پنجره‌ی Solution Explorer رفته و Program.cs را پیدا کرده و بر روی آن دوبار کلیک کنید و خط زیر را پاک کنید.

```
Application.EnableVisualStyles();
```

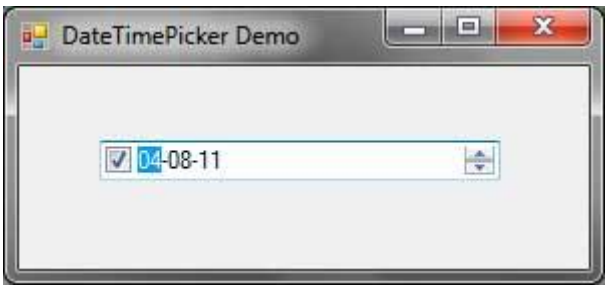
بیاد داشته باشید اگر این خط را پاک کنید، کنترل‌ها حالت Classic (قدیمی) به خود می‌گیرند. حال می‌توانید با استفاده خواص مختلفی که رنگ تقویم را تغییر می‌دهند، رنگ آنرا تغییر دهید.



شما می‌توانید قالب نمایش تاریخ را با استفاده از خاصیت Format تغییر دهید. شما می‌توانید از قالب‌های Long، Short، Time، Custom استفاده کنید. زمانی که Custom را انتخاب کنید، می‌توانید یک قالب رشته‌ای برای آن در خاصیت CustomFormat تعیین کنید. برای مثال، قرار دادن فرمت به MM-dd-yy تاریخی را نمایش می‌دهد که در آن شماره‌ی ماه، روز و دو شماره‌ی آخر سال که به وسیله‌ی یک خط تیره از هم جدا شده‌اند را به کاربر نمایش می‌دهد. برای مثال خروجی 04-08-11 برای ما تولید خواهد شد. خاصیت ShowCheckBox مشخص می‌کند که یک CheckBox در سمت چپ این کنترل قرار گیرد یا خیر.



اگر CheckBox تیک خورده باشد، می‌توانید یک تاریخ یا زمانی را انتخاب کرده و با استفاده از کلیدهای مکان نما آن را تغییر دهید، در غیر اینصورت مجاز به انجام این کار نیستید. با کلیک بر روی نوار کرکره‌ای، checkbox کنار تقویم تیک می‌خورد. حالت CheckBox با استفاده از خاصیت Checked مورد دستیابی قرار می‌گیرد. خاصیت ShowUpDown دکمه‌ی نوار کرکره‌ای را به یک دکمه‌ی بالا-پایین تغییر داده و آیکن تقویم را مخفی می‌کند. در شکل زیر عملکرد آنرا مشاهده می‌کنید.



این خاصیت شما از دیدن تقویم منع کرده و شما را به استفاده از دکمه‌ی بالا- پایین محدود می‌کند. یکی از اجزای تاریخ و زمان را انتخاب کرده و با استفاده از دکمه‌ی بالا – پایین مقدار آن را تغییر دهید. از خاصیت Value برای مقدار دادن به تاریخ انتخاب شده نیز می‌توانید استفاده کنید.

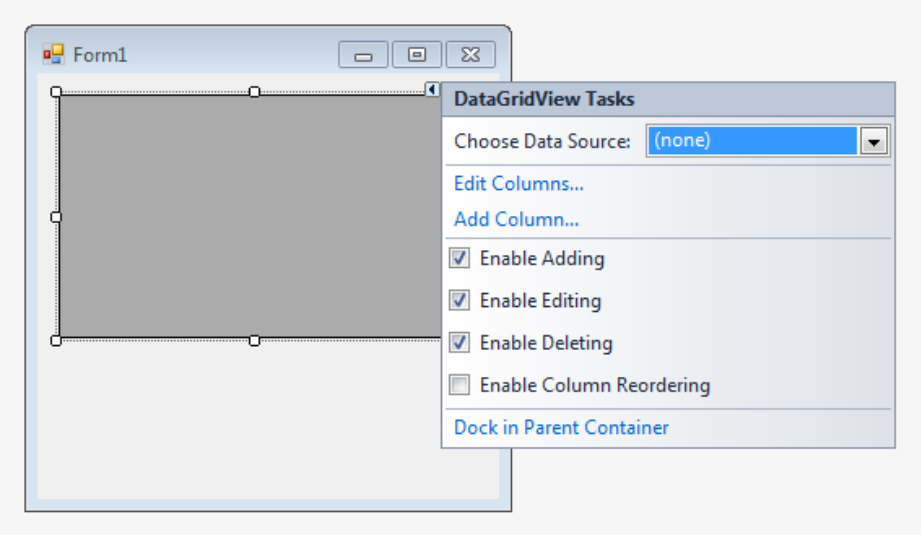
با استفاده از رویداد ValueChanged این کنترل می‌توانید به تغییرات تاریخ واکنش نشان دهید (یعنی براساس تغییراتی که کاربر در آن ایجاد می‌کند، یک پیغام را نشان داده و یا یک کد را اجرا کنید).

کنترل DataGridView

کنترل DataGridView یک کنترل بسیار قدرتمند و منعطف برای نمایش داده‌ها به صورت جدولی می‌باشد. به طور کلی می‌توان گفت که از این کنترل برای نمایش و ویرایش داده‌های موجود در یک یا چند جدول از دیتابیس استفاده می‌شود. خاصیت‌ها، و نکات این کنترل آنقدر زیاد هستن که برای شرح همه آنها باید یک کتاب جداگانه نوشته شود. با این وجود در این درس ما درباره مهم‌ترین نکاتی که در مورد این کنترل وجود دارد، توضیح می‌دهیم.

کار با DataGridView بدون کدنویسی

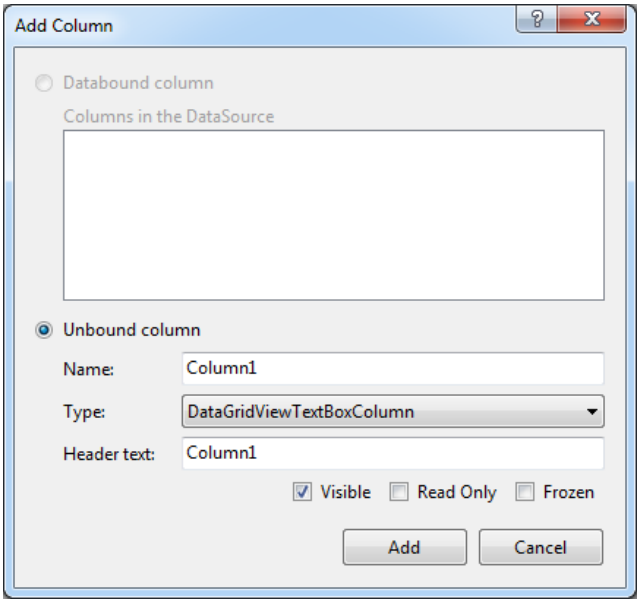
وقتی برای اولین بار یک کنترل DataGridView را بر روی فرم قرار می‌دهید، این کنترل به صورت زیر نمایش داده می‌شود:



گزینه	کاربرد
Choose Data Source	برای انتخاب یک منبع داده و اتصال آن به DataGridView جهت نمایش داده‌ها به کار می‌رود.
Edit Columns	برای ویرایش ستون‌های ایجاد شده به کار می‌رود.
Add Columns	برای اضافه کردن ستون به کار می‌رود.
Enable Adding	در صورت تیک خوردن، کاربر می‌تواند سطر جدید به DataGridView اضافه کند.
Enable Editing	در صورت تیک خوردن، کاربر می‌تواند اطلاعات سطرها را ویرایش کند.

Enable Deleting	در صورت تیک خوردن، کاربر می‌تواند سطرهایی را که می‌خواهد از DataGridView حذف کند.
Enable Column Reordering	در صورت تیک خوردن، کاربر می‌تواند ستون‌ها را مرتب کند.
Dock in Parent Container	در صورت تیک خوردن، کنترل DataGridView تمام فضای فرم را در بر می‌گیرد.

حال بهتر است که با گزینه‌های موجود در جدول بالا در عمل آشنا شوید. از آنجاییکه در این درس نمی‌خواهیم در مورد کار با دیتابیس توضیح دهیم توضیح گزینه Choose Data Source را به آینده موکول می‌کنیم. ابتدا می‌خواهیم به کنترل مان چند ستون اضافه کنیم. بر روی گزینه Add Column کلیک کرده تا صفحه ای به صورت زیر نمایش داده شود:

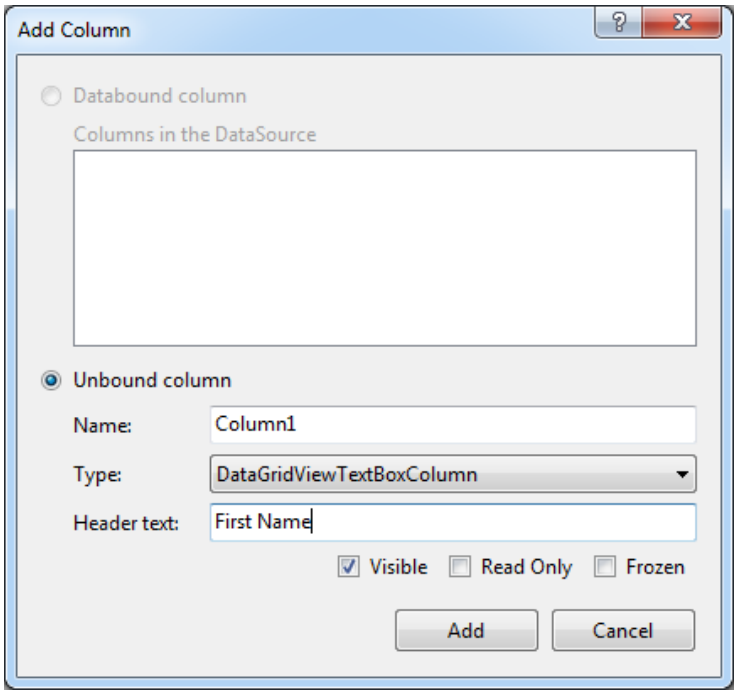


همانطور که در شکل بالا مشاهده می‌کنید این پنجره دارای دو قسمت DataBound column و Unbound column می‌باشد. DataBound column برای نمایش ستون‌های جدول یک منبع داده به کار می‌رود و چون کنترل ما به هیچ منبع داده ای وصل نیست، این قسمت غیر فعال است. چون که در این درس سر و کار ما با بخش UnBound column است در باره گزینه‌های موجود در این قسمت توضیح می‌دهیم:

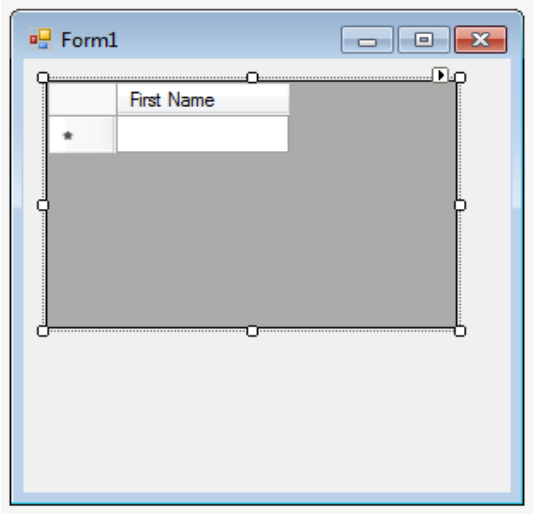
گزینه	کاربرد
Name	با استفاده از آن می‌توان یک نام برای ستون انتخاب کرد. این نام همان نامی است که در موقع کدنویسی با آن سر و کار داریم.
Type	با استفاده از آن می‌توان نوع ستون را مشخص کرد. نوع ستون می‌تواند یکی از انواع زیر باشد: DataGridViewTextBoxColumn DataGridViewCheckBoxColumn DataGridViewImageColumn DataGridViewButtonColumn DataGridViewComboBoxColumn

	DataGridViewLinkColumn
Header Text	برای ایجاد یک عنوان برای ستون به کار می‌رود.
Visible	با تیک زدن این گزینه ستون مخفی می‌شود.
Readonly	با تیک زدن این گزینه ستون فقط خواندنی و غیر قابل ویرایش می‌شود.
Frozen	با تیک زدن این گزینه ستون مربوطه ثابت می‌ماند و با Scroll افقی DataGridView سایر ستون‌ها به زیر آن کشیده می‌شوند (در ادامه یک مثال زده شده است).

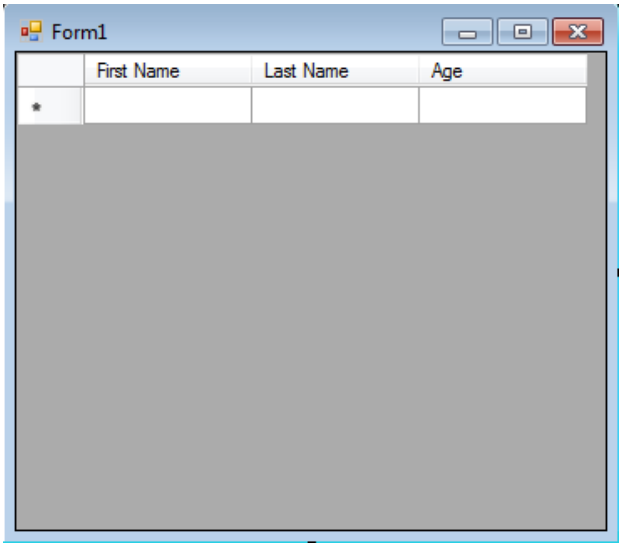
فرض کنید که می‌خواهیم با توجه به گزینه‌های موجود در جدول بالا یک ستون با نام Column1 و عنوان First Name ایجاد کنیم. برای این کار به صورت زیر عمل می‌کنیم:



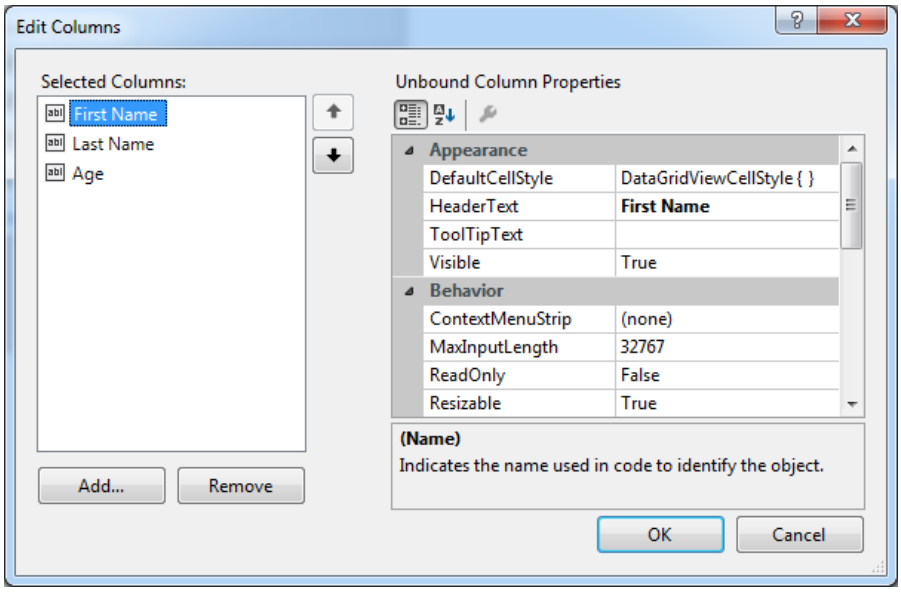
با زدن دکمه Add در شکل بالا یک ستون به صورت زیر به کنترل مان اضافه می‌شود:



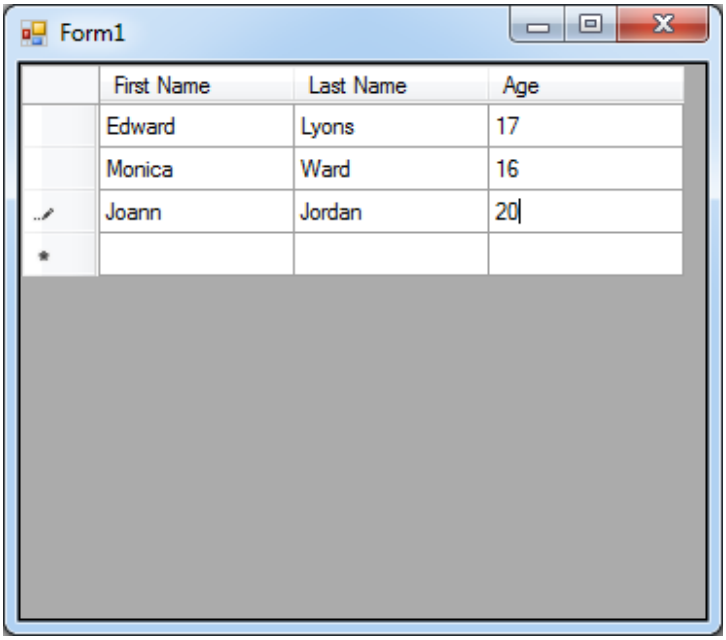
به همین روش دو ستون دیگر با عناوین Last Name و Age به کنترل اضافه می‌کنیم و با کمی تغییر اندازه فرم و با تیک زدن گزینه Dock in Parent Container شکل نهایی برنامه را به صورت زیر در می‌آوریم:



همانطور که مشاهده می‌کنید، بعد از ایجاد ستون‌ها یک سطر هم به صورت پیشفرض به کنترل اضافه می‌شود. گزینه Edit Columns هم چیز خاصی ندارد. با کلیک بر روی این گزینه پنجره ای به صورت زیر نمایش داده می‌شود:



با استفاده از آن می‌توان ستون‌ها را حذف و اضافه کرد، عنوان و چیدمان آنها را تغییر داد و در کل تنظیمات مختلفی بر روی آنها اعمال کرد. حال که ستون‌ها را اضافه کردیم، اجازه دهید که مقادیری را هم به این ستون‌ها اضافه کنیم. با دوبار کلیک بر روی هر مستطیل در این کنترل که به آنها اصطلاحاً سلول (Cell) می گویند می‌توانید مقادیری را به ستون‌ها اضافه کنید. اضافه کردن این مقادیر مترادف اضافه شدن سطر به این کنترل است. بعد از اضافه کردن مقادیر مورد نظر شکل نهایی، چیزی شبیه به شکل زیر می‌شود:



کار با DataGridView به روش کدنویسی

همانطور که در ابتدای درس مشاهده کردید، تقریباً تمام کارها، از جمله حذف و اضافه کردن ستون‌ها، سطرها، اضافه کردن مقادیر و بسیاری از اعمال دیگر را می‌توان با استفاده از پنجره‌ها و گزینه‌های مختلف کنترل DataGridView انجام داد. حال می‌خواهیم نحوه کار با سطرها و

ستون‌ها را با استفاده از کدنویسی به شما آموزش دهیم. ابتدا نحوه اضافه کردن ستون‌ها را به شما آموزش می‌دهیم. یک برنامه جدید ویندوزی ایجاد کرده و یک کنترل DataGridView بر روی فرم قرار دهید. حال بر روی فرم دو بار کلیک کرده تا کنترل کننده رویداد Loadایجاد شود. در داخل این کنترل کننده رویداد کد زیر را بنویسید:

```
private void Form1_Load(object sender, EventArgs e)
{
    dataGridView1.ColumnCount = 3;

    dataGridView1.Columns[0].Name = "Column1";
    dataGridView1.Columns[1].Name = "Column2";
    dataGridView1.Columns[2].Name = "Column3";

    dataGridView1.Columns[0].HeaderText = "First Name";
    dataGridView1.Columns[1].HeaderText = "Last Name";
    dataGridView1.Columns[2].HeaderText = "Age";
}
```

در کد بالا ابتدا با اسفاده از خاصیت ColumnCount مشخص کرده‌ایم که قرار است سه ستون به DataGridView اضافه شود. سپس با استفاده از خاصیت‌های Name و HeaderText به هر ستون یک نام و یک عنوان اختصاص داده‌ایم. همانطور که مشاهده می‌کنید اندیس ستون‌های DataGridView از ۰ شروع می‌شود. کد بالا را به صورت خلاصه تر زیر هم می‌توان نوشت:

```
private void Form1_Load(object sender, EventArgs e)
{
    dataGridView1.Columns.Add("Column1", "First Name");
    dataGridView1.Columns.Add("Column2", "Last Name");
    dataGridView1.Columns.Add("Column3", "Age");
}
```

در این کد ما با استفاده از متد Add() خاصیت Columns سه ستون ایجاد کرده‌ایم. این متد دو آرگومان قبول می‌کند که اولی نام ستون و دیگری عنوان آن است. برنامه را اجرا و نتیجه را مشاهده کنید. برای اضافه کردن سطر هم از خاصیت Rows استفاده می‌شود. این خاصیت دارای یک متد به نام Add() است که دارای سربارگذاری های مختلفی می‌باشد. در ادامه کدهای قبلی کد زیر را بنویسید:

```
dataGridView1.Rows.Add(3);

dataGridView1.Rows[0].Cells[0].Value = "Edward";
dataGridView1.Rows[0].Cells[1].Value = "Lyons";
dataGridView1.Rows[0].Cells[2].Value = 17;

dataGridView1.Rows[1].Cells[0].Value = "Monica";
dataGridView1.Rows[1].Cells[1].Value = "Ward";
dataGridView1.Rows[1].Cells[2].Value = 16;

dataGridView1.Rows[2].Cells[0].Value = "Joann";
dataGridView1.Rows[2].Cells[1].Value = "Jordan";
dataGridView1.Rows[2].Cells[2].Value = 20;
```

در کد با ارسال عدد ۳ به این متد Add()، سه سطر به کنترل اضافه می‌شود. برای اضافه کردن مقادیر به این سطرها هم همانطور که در خطوط بعد مشاهده می‌کنید خاصیت Rows دارای خاصیتی به نام Cells و این خاصیت به نوبه خود دارای خاصیتی به نام Value می‌باشد. از این سه خاصیت برای مقدار دهی به یک سلول خاص از یک سطر خاص استفاده می‌شود. مثلاً برای مقداردهی سلول دوم از سطر سوم به صورت زیر عمل می‌شود:

```
dataGridView1.Rows[2].Cells[1].Value = "Jordan";
```

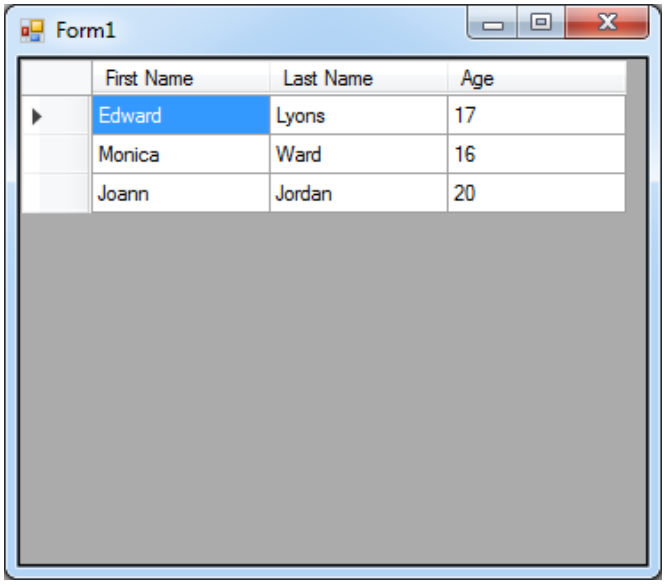
اندیس سطرها و سلول‌ها هم از صفر شروع می‌شود، پس سطر سوم به صورت Rows[2] و سلول دوم به صورت Cells[1] اندیس گذاری می‌شود. خاصیت Value هم یک مقدار از نوع Object قبول می‌کند. در نتیجه بعد از علامت مساوی می‌توان هر نوع داده ای را به این خاصیت اختصاص داد. برای اضافه کردن سطرها به صورت ساده تر می‌توان از دیگر سربارگذای متد Add() به صورت زیر استفاده کرد:

```
dataGridView1.Rows.Add("Edward", "Lyons", 17);
dataGridView1.Rows.Add("Monica", "Ward", 16);
dataGridView1.Rows.Add("Joann", "Jordan", 20);
```

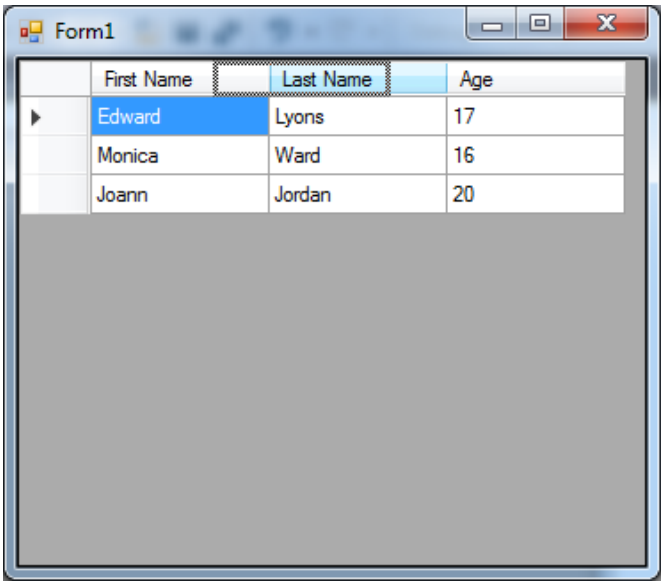
در کل می‌توان تمام کدهایی را که تا کنون برای ایجاد سطر و ستون در رویداد Load فرم نوشتیم به صورت زیر خلاصه کنیم:

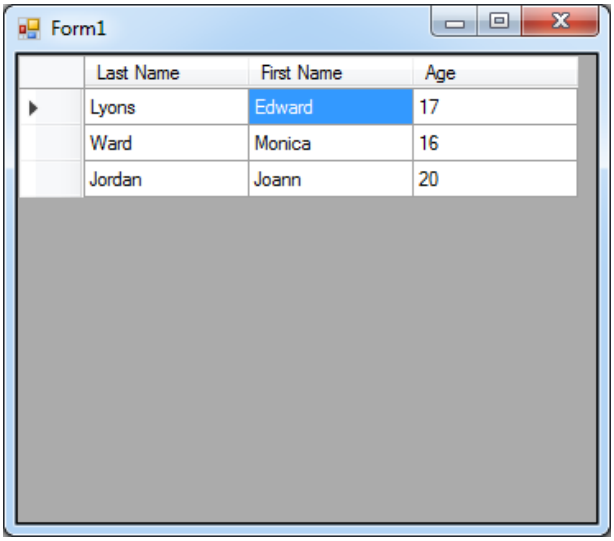
```
private void Form1_Load(object sender, EventArgs e)
{
    dataGridView1.Columns.Add("Column1", "First Name");
    dataGridView1.Columns.Add("Column2", "Last Name");
    dataGridView1.Columns.Add("Column3", "Age");

    dataGridView1.Rows.Add("Edward", "Lyons", 17);
    dataGridView1.Rows.Add("Monica", "Ward", 16);
    dataGridView1.Rows.Add("Joann", "Jordan", 20);
}
```

اکنون وقت آن رسیده که در مورد گزینه‌های ابتدای درس توضیح دهیم. تیگ گزینه Enable Adding را برداشته و برنامه را اجرا کنید. مشاهده می‌کنید که سطر آخر کنترل حذف شده و به شما اجازه اضافه کردن مقدار به کنترل در محیط ویژوال داده نمی‌شود:



با برداشتن تیک گزینه Enable Editing شما نمی‌توانید مقادیر موجود در سلول‌ها را با دوبار کلیک بر روی آنها به حالت ویرایش درآورید. این کار را انجام داده و با اجرای برنامه سعی کنید که مقدار یک سلول را تغییر دهید. مشاهده می‌کنید که این کار امکان پذیر نیست. گزینه Enable Column Reordering را تیک بزنید. با تیک این گزینه شما می‌توانید جای ستون‌ها را با هم عوض کنید. این کار با گرفتن یک ستون با ماوس و کشیدن آن بر روی ستون دیگر انجام دهید:

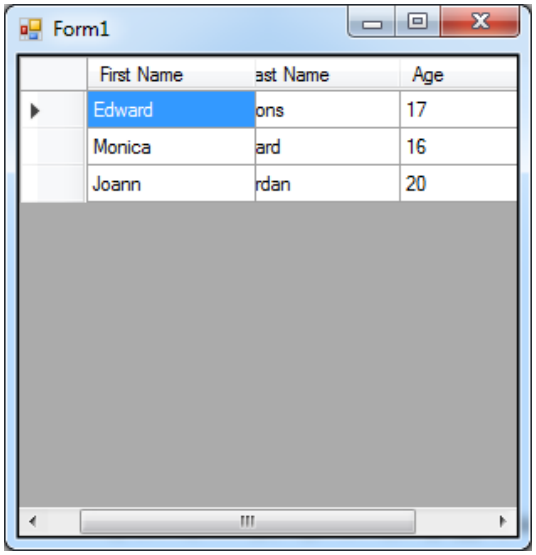




گزینه `Enable Deleting` هم به در صورت تیک خوردن به کاربر اجازه می‌دهد که یک یا چند سطر انتخاب شده را با استفاده از دکمه `Delete` کیبورد حذف کند و اگر تیک برداشته شود این کار امکان پذیر نیست. هر ستون دارای خاصیتی به نام `Frozen` است که دو مقدار `true` یا `false`می‌گیرد. اگر مقدار این خاصیت را در یک ستون `true` کنیم، ستون‌های دیگر در هنگام `Scroll` کنترل `DataGridView` به زیر ستون مذکور کشیده می‌شوند. برای مشاهده کاربرد این خاصیت مقدار آن را درستون اول `true` می‌کنیم:

```
dataGridView1.Columns["Column1"].Frozen = true;
```

حال اگر کمی `DataGridView` را کوچک‌تر کنیم تا `Scroll` آن نمایان شود، مشاهده خواهید کرد که با جابه جایی اسکرول ستون‌های دیگر به زیر این ستون کشیده می‌شوند:



کار با بقیه گزینه‌ها را در ادامه توضیح می‌دهیم. دیتاگریدویو دارای خاصیت و رویدادهای زیادی است که در جداول زیر به آنها اشاره شده است :

خاصیت	توضیح
AllowUserToAddRows	مشخص می‌کند که آیا کاربر می‌تواند سطر اضافه کند یا نه؟
AllowUserToDeleteRows	مشخص می‌کند که آیا کاربر می‌تواند یک سطر را حذف کند یا نه؟
AllowUserToOrderColumns	مشخص می‌کند که آیا کاربر می‌تواند با کشیدن ستون به سمت راست یا چپ ترتیب آنها را تغییر دهد یا نه؟
AllowUserToResizeColumns	مشخص می‌کند که آیا کاربر می‌تواند ستون را تغییر اندازه دهد یا نه؟
AllowUserToResizeRows	مشخص می‌کند که آیا کاربر می‌تواند سطر را تغییر اندازه دهد یا نه؟
AlternatingRowsDefaultCellStyle	قالب اعمال شده به سطرهای فرد را تعیین می‌کند.
AutoGenerateColumns	مشخص می‌کند که آیا ستون‌ها می‌توانند به صورت خودکار تولید شوند یا نه؟
AutoSizeColumnsMode	مشخص می‌کند که عرض ستون‌ها چگونه تغییر کند؟ (بر اساس متن محتوا، یا عنوان و یا...)
AutoSizeRowsMode	چگونگی تغییر ارتفاع سطرها را مشخص می‌کند.
CellBorderStyle	قالب خطوط سلول‌های دیتاگرید را مشخص می‌کند.
ClipboardCopyMode	مشخص می‌کند که آیا می‌توان مقادیر سلول‌ها را در ClipBoard کپی کرد یا خیر؟
ColumnCount	تعداد ستون‌های نمایش داده شده در دیتاگرید را بر می‌گرداند.

ColumnHeadersHeight	ارتفاع ستون هدر را مشخص می‌کند.
ColumnHeadersHeightSizeMode	مشخص می‌کند که آیا ارتفاع ستون هدر قابل تغییر است یا کاربر باید آن را تغییر دهد یا به صورت خودکار با محتوا تغییر کند.
ColumnHeadersVisible	مشخص می‌کند که آیا ستون هدر نمایش داده شود یا خیر؟
Columns	مجموعه ای از همه ستون‌های کنترل را بر می‌گرداند.
CurrentCell	سلول جاری را مقداردهی کرده یا برمی گرداند.
CurrentCellAddress	اندیس سطر و ستون سلول فعال را بر می‌گرداند.
CurrentRow	سطری که حاوی سلول جاری است را بر می‌گرداند.
DataMember	نام لیست یا جدولی از منبع داده که قرار است داده‌های آن در دیتاگرید نمایش داده شوند را مشخص می‌کند.
DataSource	منبع داده ای که دیتاگرید ویو داده‌های آن را نمایش می‌دهد را مشخص می‌کند.
DefaultCellStyle	اگر هیچ قالبی برای سلول تنظیم نشده باشد، قالب پیشفرض را به سلول اعمال می‌کند.
GridColor	رنگ خطوط جدا کننده سلول‌ها را مشخص می‌کند.
HorizontalScrollBar	مشخص می‌کند که کنترل اسکرول بار افقی داشته باشد یا نه؟
HorizontalScrollingOffset	تعداد پیکسل‌هایی را که قرار است به صورت افقی پیمایش شوند را مشخص می‌کند.
IsCurrentCellDirty	مشخص می‌کند که آیا سلول تغییر پذیر است یا نه؟
MultiSelect	مشخص می‌کند که آیا کاربر می‌تواند چندین سلول، سطر یا ستون را انتخاب کند یا نه؟
RowCount	تعداد سطرهای دیتاگرید را مشخص می‌کند.
RowHeadersBorderStyle	قالب خطوط بین سلول‌های هدر را مشخص می‌کند.
RowHeadersDefaultCellStyle	قالب پیشفرض را به سلول‌های هدر اعمال می‌کند.
RowHeadersVisible	مشخص می‌کند که آیا هدر نمایش داده شود یا نه؟
RowHeadersWidth	عرض ستون‌هایی که در داخل سطر هدر هستند را مشخص می‌کند.

RowHeadersWidthSizeMode	مشخص می‌کند که آیا عرض سطر هدر غیر قابل تغیر باشد یا توسط کاربر تغییر کند یا به طور خودکار با محتوای هدر هم اندازه شود.
Rows	مجموعه ای که شامل همه سطرهای دیتاگرید است را بر می‌گرداند.
ScrollBars	نوع نوار پیمایشی که قرار است برای دیتاگرید نمایش داده شود را مشخص می‌کند.
SelectedCells	مجموعه ای از سلول‌هایی را که توسط کاربر انتخاب شده‌اند را بر می‌گرداند.
SelectedColumns	مجموعه ای از ستون‌هایی را که توسط کاربر انتخاب شده‌اند را بر می‌گرداند.
SelectedRows	مجموعه ای از سطرهایی را که توسط کاربر انتخاب شده‌اند را بر می‌گرداند.
SelectionMode	چگونگی انتخاب سلول‌های دیتاگرید را مشخص می‌کند.
ShowCellErrors	مشخص می‌کند که آیا خطاهای سلول نمایش داده شوند یا نه؟
ShowCellToolTips	مشخص می‌کند که آیا توضیحاتی درباره سلولی که ماوس بر روی آن قرار دارد نمایش داده شود یا نه؟
SortOrder	مشخص می‌کند که آیا آیتم‌های کنترل دیتاگرید مرتب شده‌اند یا نه؟

رویداد	توضیحات
AllowUserToAddRowsChanged	وقتی روی می‌دهد که مقدار خاصیت AllowUserToAddRows تغییر کند.
AllowUserToDeleteRowsChanged	وقتی روی می‌دهد که مقدار خاصیتAllowUserToDeleteRowsChanged تغییر کند.
AllowUserToOrderColumnsChanged	وقتی روی می‌دهد که مقدار خاصیتAllowUserToOrderColumns تغییر کند.
AllowUserToResizeColumnsChanged	وقتی روی می‌دهد که مقدار خاصیت AllowUserToResizeColumns تغییر کند.
AllowUserToResizeRowsChanged	وقتی روی می‌دهد که مقدار خاصیتAllowUserToResizeRows تغییر کند.

AlternatingRowsDefaultCellStyleChanged	وقتی روی می‌دهد که مقدار خاصیت AlternatingRowsDefaultCellStyle تغییر کند.
AutoGenerateColumnsChanged	وقتی روی می‌دهد که مقدار خاصیتAutoGenerateColumnsChanged تغییر کند.
AutoSizeColumnModeChanged	وقتی روی می‌دهد که مقدار خاصیت AutoSizeMode تغییر کند.
AutoSizeColumnsModeChanged	وقتی روی می‌دهد که مقدار خاصیت AutoSizeColumnsMode تغییر کند.
AutoSizeRowsModeChanged	وقتی روی می‌دهد که مقدار خاصیتDataGridViewAutoSizeRowsMode تغییر کند.
BackgroundColorChanged	وقتی روی می‌دهد که مقدار خاصیت BackgroundColor تغییر کند.
BindingContextChanged	وقتی روی می‌دهد که مقدار خاصیت BindingContext تغییر کند.
BorderStyleChanged	وقتی روی می‌دهد که مقدار خاصیت BorderStyleتغییر کند.
CausesValidationChanged	وقتی روی می‌دهد که مقدار خاصیت CausesValidation تغییر کند.
CellClick	وقتی روی می‌دهد که بر روی قسمتی از سلول کلیک شود.
CellContentClick	وقتی روی می‌دهد که بر روی محتوای داخل یک سلول کلیک شود.
CellContentDoubleClick	وقتی روی می‌دهد که بر روی محتوای سلول‌ها دو بار کلیک شود.
CellContextMenuStripChanged	وقتی روی می‌دهد که خاصیت ContextMenuStrip تغییر کند.
CellDoubleClick	وقتی روی می‌دهد که بر روی قسمتی از سلول دوبار کلیک شود.
CellEnter	وقتی روی می‌دهد که سلول جاری تغییر کرده و یا فوکوس بگیرد.
CellErrorTextChanged	وقتی روی می‌دهد که خاصیت ErrorText یک سلول تغییر کند.
ColumnDividerDoubleClick	وقتی روی می‌دهد که کاربر بر روی خط جدا کننده دو ستون دوبار کلیک کند.
ColumnDividerWidthChanged	وقتی روی می‌دهد که خاصیت DividerWidth تغییر کند.
ColumnHeaderCellChanged	وقتی روی می‌دهد که محتوای یک سلول هدر تغییر کند.
ColumnHeaderMouseClick	وقتی روی می‌دهد که کاربر بر روی یک ستون هدر کلیک کند.

ColumnHeaderMouseDoubleClick	وقتی روی می‌دهد که بر روی یک ستون هدر دو بار کلیک شود.

همانطور که اشاره شد، این کنترل یک کنترل پیچیده با نکات بسیار زیاد است. در ادامه می‌خواهیم مطالبی را به شما آموزش دهیم که در هنگام کار با DataGridView اکثر کاربران با آنها مواجه می‌شوند. مثلاً اینکه چطور مقدار یک سلول که بر روی آن کلیک شده است را به دست آورند و یا چطور مقادیر یک سطر را در داخل چند TextBox نمایش دهند و .... ابتدا بهتر است با چند رویداد پر کاربرد شروع کنیم .

به دست آوردن تعداد سطرها و ستون‌ها

برای به دست آوردن تعداد سطرها و ستون‌های DataGridView می‌توان از خاصیت‌های RowCount و ColumnCount مربوط به آن استفاده کرد. کد زیر را در رویداد Load فرم بنویسید و برنامه را اجرا کنید:

```
MessageBox.Show(
                "Row : "    + dataGridView1.RowCount.ToString() + "    " +
                "Column : " + dataGridView1.ColumnCount.ToString()
               );
```

رویدادهای مهم DataGridView

دو رویداد از رویدادهای مهم DataGridView عبارت‌اند از CellClick و CellContentClick. رویداد CellClick زمانی رخ می‌دهد که بر روی یک سلول کلیک شود. بر روی رویداد CellClick دوبار کلیک کرده و کد زیر را بنویسید:

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    MessageBox.Show("This is CellClick Event");
}
```

با نوشتن کد بالا روی هر سلولی که کلیک کنید پیغام به شما نمایش داده می‌شود. رویداد CellContentClick زمانی رخ می‌دهد که بر مقداری که در داخل سلول است کلیک شود نه فضای سفید سلول. بر روی رویداد CellContentClick دوبار کلیک کرده و کد زیر را بنویسید:

```
private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
    MessageBox.Show("This is CellContentClick Event");
}
```

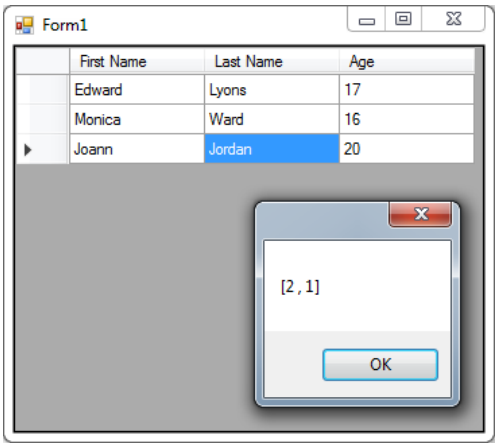
برنامه را اجرا کرده و بر روی یک قسمت خالی از یک سلول کلیک کنید. مشاهده می‌کنید که هیچ پیغامی نمایش داده نمی‌شود. حال بر روی مقدار سلول کلیک کرده و نتیجه را مشاهده نمایید.

به دست آوردن اندیس سلول، سطر و ستون

کنترل کننده رویداد این دو رویداد یک شیء از کلاس DataGridViewCellEventArgs می‌گیرد که اطلاعات مهمی درباره سلول و یا سطری که بر روی آن کلیک شده است در اختیار ما قرار می‌دهد. این کلاس دارای دو خاصیت به نام‌های RowIndex و ColumnIndex می‌باشد که به ترتیب برای به دست آوردن اندیس سطر و ستون سلولی که بر روی آن کلیک شده است به کار می‌روند. برای آشنایی با این خاصیت‌ها بر روی رویدادCellClick دوبار کلیک کرده و کدهای زیر را بنویسید:

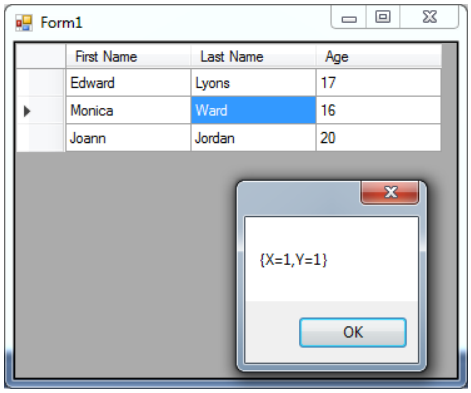
```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    MessageBox.Show("[" + e.RowIndex.ToString() + " , " + e.ColumnIndex.ToString() + "]");
}
```

با اجرای برنامه و با کلیک بر روی هر سلول، اندیس سطر و ستون آن نمایش داده می‌شود:



با توجه به شکل بالا، سلولی با مقدار Jordan در سطر سوم و ستون دوم قرار داد. برای به دست آوردن اندیس سطر و ستون یک سلول از خاصیت CurrentCellAddress هم می‌توان استفاده کرد:

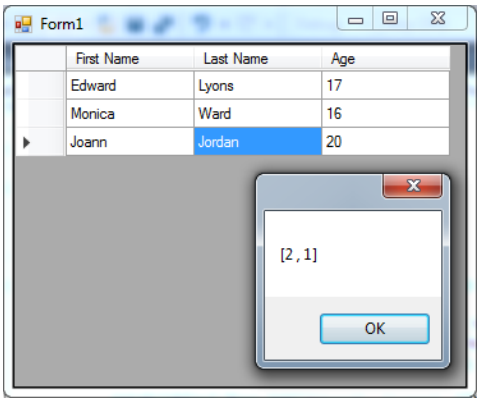
```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    MessageBox.Show(dataGridView1.CurrentCellAddress.ToString());
}
```



یکی دیگر از روش‌های به دست آوردن اندیس سطر و ستون‌های یک سلول استفاده از خاصیت CurrentCell است. این خاصیت دارای دو خاصیت به نام‌های RowIndex و ColumnIndex می‌باشد که به ترتیب اندیس سطر و ستون را بر می‌گردانند:

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    int X = dataGridView1.CurrentCell.RowIndex;
    int Y = dataGridView1.CurrentCell.ColumnIndex;

    MessageBox.Show("[" + X.ToString() + " , " + Y.ToString() + "]");
}
```

با اجرای برنامه و با کلیک بر روی هر سلول، اندیس سطر و ستون آن نمایش داده می‌شود:

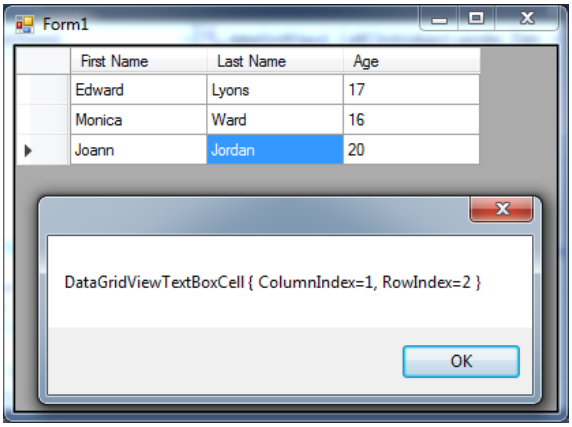


خاصیتی دیگر که با استفاده از آن می‌توان اندیس سطر و ستون‌های یک سلول را به دست آورد خاصیت Rows است:

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    MessageBox.Show(dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].ToString());
}
```

همانطور که می دانید یک سلول از تلاقی یک سطر و ستون به وجود می‌آید. در نتیجه برای به دست آوردن اندیس سطر و ستون سلولی که بر روی آن کلیک شده است به ترتیب از e.RowIndex و e.ColumnIndex استفاده می‌کنیم. قرار دادن e.RowIndex در داخل کروشه خاصیت Rows به معنای سطر فعلی و حال اگر برنامه را اجرا کرده و قرار دادن e.ColumnIndex در داخل خاصیت Cells به معنای ستون جاری است.

بر روی یک سلول کلیک کنید مقدار آن برای شما چاپ می‌شود:



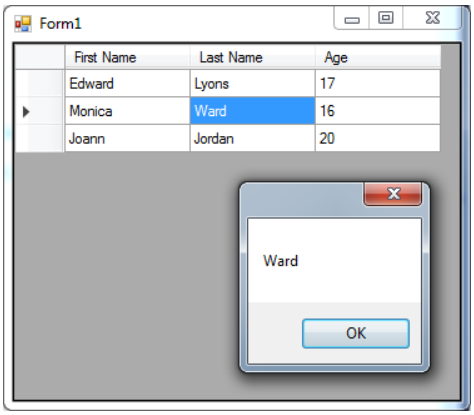
البته به جای [e.RowIndex]Rows در خط اول کد بالا می‌توان از خاصیت CurrentRow به صورت زیر استفاده کرد:

```
string CellValue = dataGridView1.CurrentRow.Cells[e.ColumnIndex].ToString();
```

به دست آوردن مقدار یک سلول

برای به دست آوردن مقدار یک سلول از خاصیت Value استفاده می‌شود. کافیست که بعد از به دست آوردن اندیس سلول مورد نظر این خاصیت را اضافه کنید تا مقدار آن را به شما بدهد. مثلاً نوشتن کدهای زیر در رویدادهای CellClick و CellContentClick باعث می‌شود که با کلیک بر روی سلول، مقدار داخل آن به شما نمایش داده شود:

```
MessageBox.Show(dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString());
MessageBox.Show(dataGridView1.CurrentRow.Cells[e.ColumnIndex].Value.ToString());
```

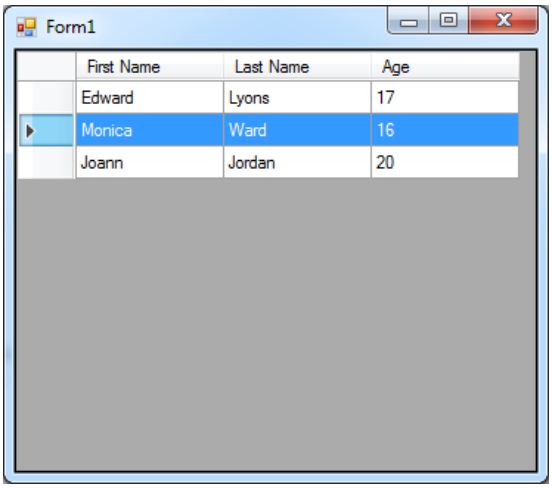


انتخاب یک سطر یا ستون

در حالت پیشفرض با کلیک بر روی DataGridView یک سلول به حالت انتخاب در می‌آید. اگر بخواهید یک سطر را به طور کامل انتخاب کنید کافیست که مثلاً در رویداد Load فرم از کد زیر استفاده کنید:

```
dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
```

خاصیت SelectionMode از دیتاگریدویو یک مقدار از نوع شمارشی DataGridViewSelectionMode می‌گیرد. این نوع شمارشی دارای مقادیر مختلفی است که یکی از آنها FullRowSelect می‌باشد که با اختصاص آن به خاصیت SelectionMode باعث می‌شود که با کلیک بر روی یک سطر، تمام سطر به حالت انتخاب درآید:



برای انتخاب یک ستون از DatagridView هم باید از یک حلقه for استفاده کنیم. کد زیر را مثلاً در رویداد CellClick بنویسید:

```
for (int i = 0; i < dataGridView1.RowCount; i++)
{
    dataGridView1[dataGridView1.CurrentCell.ColumnIndex, i].Selected = true;
}
```

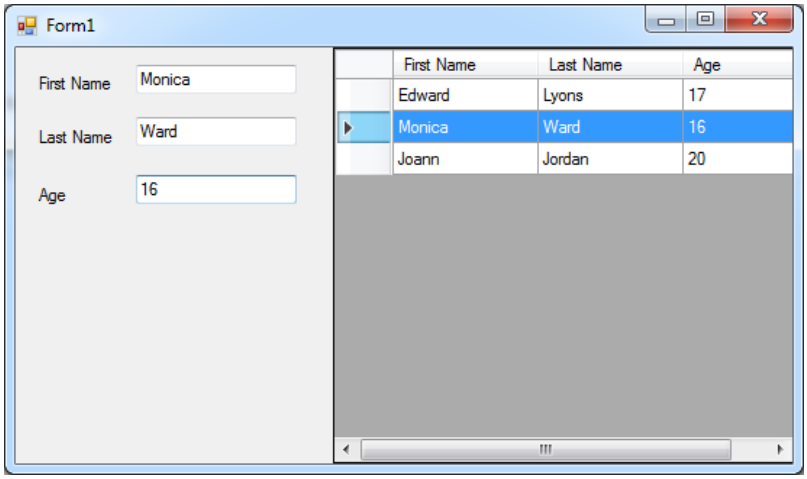
حال اگر برنامه را اجرا و بر روی یک سلول کلیک کنید، کل ستون مربوط به آن به حالت انتخاب در می‌آید.

نمایش مقادیر سطرهای DataGridView در TextBox

انتخاب کامل یک سطر معمولاً زمانی به کار می‌آید که شما بخواهید مقادیر موجود در سلول‌ها را در TextBox های جداگانه نمایش دهید. فرض کنید که می‌خواهید با کلیک بر روی DataGridView مثال مان مقادیر موجود در سه ستون آن در سه TextBox نمایش داده شوند. برای این کار ابتدا سه TextBox بر روی فرم قرار داده و سپس در رویداد CellClick کدهای زیر را بنویسید:

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        textBox1.Text = dataGridView1.CurrentRow.Cells["Column1"].Value.ToString();
        textBox2.Text = dataGridView1.CurrentRow.Cells["Column2"].Value.ToString();
        textBox3.Text = dataGridView1.CurrentRow.Cells["Column3"].Value.ToString();
    }
}
```

در کد بالا و در قسمت شرط چک می‌کنیم که اگر اندیس سطری که بر روی آن کلیک شده است ۰ یا بزرگ‌تر از صفر باشد، مقدار ستون اول از سطر انتخاب شده را در داخل TextBox اول، مقدار ستون دوم را در TextBox دوم و مقدار ستون سوم را در TextBox سوم قرار بده. برنامه را اجرا و بر روی یک سطر کلیک کنید:



این نکته را یادآور می‌شویم که چون اندیس سلول‌های هر سطر از صفر شروع می‌شود، پس به جای مقادیر Column1، Column2 و Column3 می‌توانید از ۰، ۱ و ۲ استفاده نمایید، مثلاً Cells[0] به معنای سلول اول است.

حذف سطر یا سطرهای انتخاب شده

گاه ممکن است بخواهید یک یا چند سطر از DataGridView را حذف کنید. برای این منظور ابتدا یک دکمه بر روی فرم قرار دهید و سپس با دوبار کلیک بر روی آن، کدهای زیر را در رویداد Click بنویسید:

```
private void button1_Click(object sender, EventArgs e)
{
    foreach (DataGridViewRow selectedRow in this.dataGridView1.SelectedRows)
    {
        dataGridView1.Rows.RemoveAt(selectedRow.Index);
    }
}
```

در کد بالا ابتدا یک شیء از کلاس DataGridViewRow ایجاد کرده‌ایم. این کار به معنای این است که ما قرار است بر روی سطرها کاری انجام دهیم. قسمت شرط به این معناست که "به ازای سطرهای انتخاب شده از DataGridView" فلان کار را انجام بده. چه کاری؟ در داخل بدنه حلقه اندیس تک تک سطرهای انتخاب شده را به متد RemoveAt() می‌دهیم تا آنها را حذف کند. برنامه را اجرا کرده و با انتخاب یک یا چند سطر بر روی دکمه کلیک و نتیجه را مشاهده نمایید.

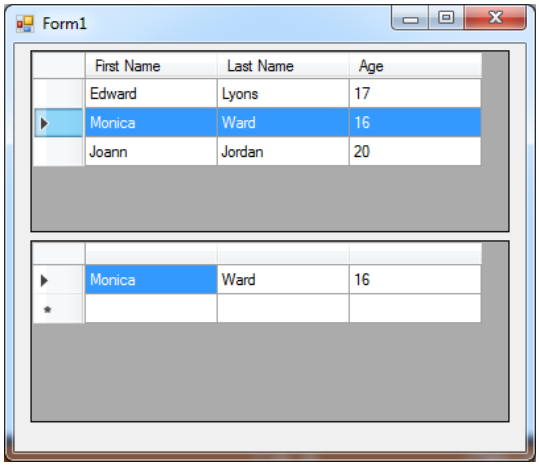
انتقال سطر از یک DataGridView به DataGridView دیگر

فرض کنید، همین DataGridView که در بالا ایجاد کرده‌ایم و دارای سه سطر و سه ستون است را در اختیار داریم و می‌خواهیم با کلیک بر روی هر سطر از آن، سطر انتخاب شده به DataGridView دیگر منتقل شود. ابتدا یک DataGridView دیگر به فرم اضافه کرده و سپس به کد زیر توجه کنید:

```
1  private void Form1_Load(object sender, EventArgs e)
2  {
3      // Add Rows and Columns
4
```

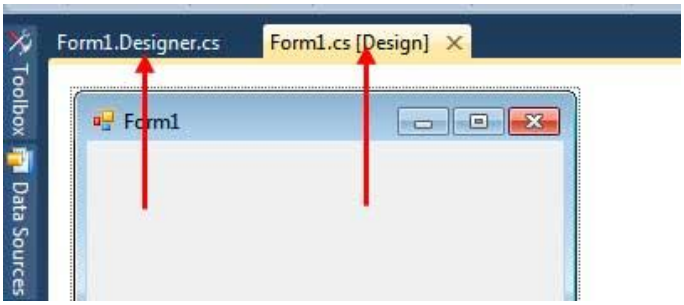
```
5       dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
6
7       dataGridView2.ColumnCount = dataGridView1.ColumnCount;
8   }
9
10  private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
11  {
12      string name   =      dataGridView1.CurrentRow.Cells[0].Value.ToString();
13      string family =      dataGridView1.CurrentRow.Cells[1].Value.ToString();
14      int    age    = (int)dataGridView1.CurrentRow.Cells[2].Value;
15
16      dataGridView2.Rows.Add(new object[] { name, family, age });
17  }
```

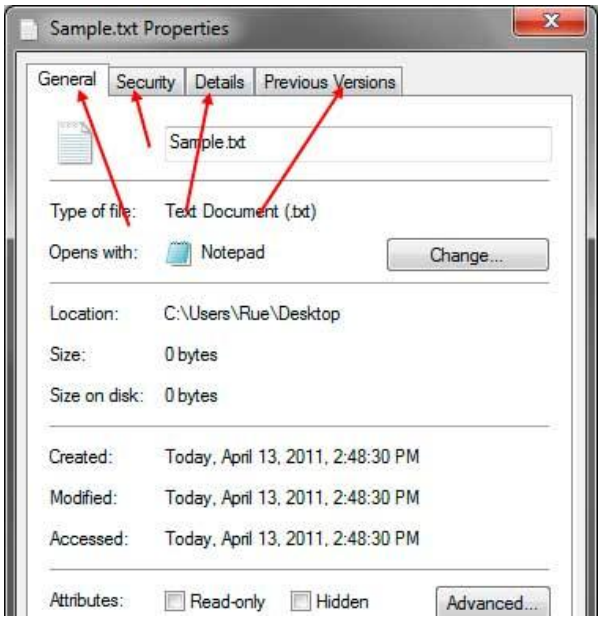
همانطور که در کد بالا مشاهده می‌کنید ما برای جلوگیری از تکرار، کدهای مربوط به اضافه کردن سطرها و ستون‌های مربوط به DataGridView اول را ننوشته‌ایم. برای انتقال یک سطر از یک دیتاگرید ویو به دیگری باید کل سطر انتخاب شود که ما این کار را در خط ۵ انجام داده‌ایم. شرط دوم برای انتقال سطر برابر بودن ستون‌های دو DataGridView است که کد مربوط به آن را در خط ۷ نوشته‌ایم. برای نهایی شدن انتقال هم بر روی CellClick دیتاگریدویو اصلی دوبار کلیک کرده و کدهای مربوط به خطوط ۱۴-۱۲ را در داخل آن نوشته‌ایم. همانطور که در خطوط ۱۶-۱۲ مشاهده می‌کنید، به ترتیب مقدار سلول اول، دوم و سوم از سطر انتخاب شده را در سه متغیر قرار می‌دهیم و سپس در خط ۱۶ آنها را به DataGridViewدوم اضافه می‌کنیم. برنامه را اجرا و با کلیک بر روی یک سطر نتیجه را مشاهده کنید :



کنترل TabControl

کنترل TabContorl، به شما اجازه می‌دهد که برای پنجره‌های خود سربرگ (Tab) بسازید، شما در برنامه‌های زیادی این نوع پنجره‌ها را مشاهده کرده‌اید. برای مثال پنجره‌ی Properties فایل‌ها و سربرگ‌های برنامه‌ی Visual Studio.





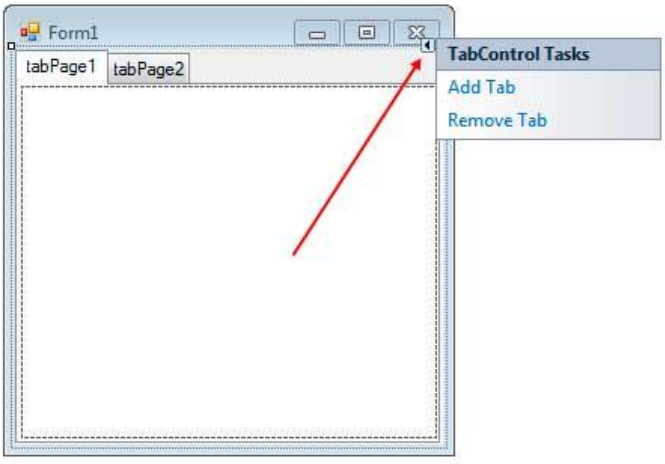
TabControl می‌تواند حاوی کنترل‌های دیگر باشد. با کلیک بر روی هر سربرگ (Tab) می‌توانید محتویات آنرا مشاهده کنید. ظاهر یک سربرگ فعال با دیگر سربرگ‌ها متفاوت است، بنابراین شما می‌توانید متوجه شوید که کدام یک از سربرگ‌ها فعال است. زمانی که شما بر روی یک سربرگ کلیک می‌کنید، کنترل‌هایی که به آن تعلق دارند نمایش داده می‌شوند. TabControl به شما اجازه می‌دهد که یک فرم را در داخل سربرگ‌های مختلف سازماندهی کنید، به طوری که هر سربرگ یک دسته را نمایش دهد.

برای مثال، می‌توانید یک فرم که حاوی اطلاعات شخصی را در سربرگ Personal Info و پیش زمینه‌ی آموزشی را در سربرگ Educational Background قرار دهید. برای اضافه کردن یک کنترل TabContorl، به قسمت Containers در جعبه ابزار (ToolBox) رفته و کنترل TabControl را بر روی فرم قرار دهید. می‌توانید سایز این کنترل را تغییر دهید اما بهترین راه استفاده از خاصیت Dock است، که باعث می‌شود تمام فضای فرم اشغال شود. به پنجره‌ی خواص (Properties) رفته و خاصیت Dock را پیدا کنید. بر روی نوار کرکره‌ای کلیک کرده و سپس دکمه‌ی وسط (Middle) را انتخاب کنید. این گزینه به این معناست که می‌تواند تمام فرم را پر کند. به شکل زیر توجه کنید:

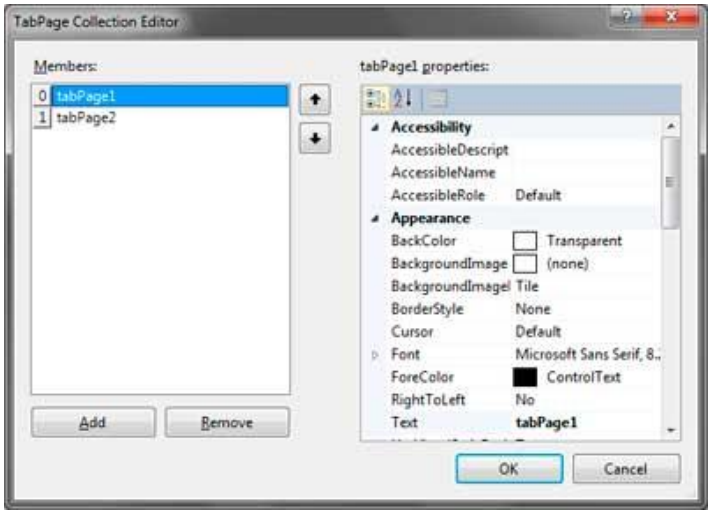
TabControl شامل کنترل‌های TabPage است، که هر کدام از آنها شامل سربرگ و اجزای مخصوص به خودشان هستند. این کنترل‌ها توسط خاصیت TabPages از کنترل TabControl مورد دستیابی قرار می‌گیرند. در جدول زیر لیستی از خاصیت‌های مفید کنترل TabPage را مشاهده می‌کنید.

خاصیت	توضیح
Controls	کنترل‌هایی که در داخل TabPage وجود دارند.
ImageIndex	اندیس تصویری که برای Tabpage انتخاب می‌شود را تعیین می‌کند.
Text	متنی که در دکمه‌ی سربرگ نمایش داده می‌شود.

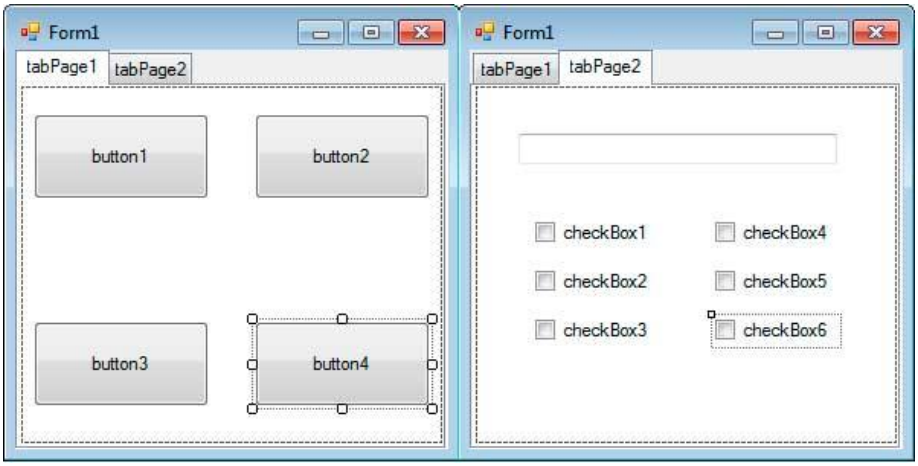
برای اضافه کردن یک TabPage (سربرگ صفحه‌ای)، می‌توانید بر روی فلش کوچک که در قسمت بالا و راست TabControl وجود دارد کلیک کنید. با زدن این دکمه یک منو باز می‌شود که به وسیله‌ی آن شما می‌توانید Tabها اضافه و یا حذف کنید.



روش دیگر، استفاده از پنجره‌ی Properties و خاصیت TabPages می‌باشد. برای باز کردن TabPages Collection Editor ابتدا مطمئن شوید که کنترل TabControl فعال است سپس بر روی دکمه در کنار خاصیت TabPages کلیک کنید.



پنجره TabPage Collection Editor به شما اجازه می‌دهد که TabPageهای مختلفی را ایجاد یا حذف کرده و خواص مربوط به هرکدام از آنها را تغییر دهید. هر TabPage می‌تواند در برگیرنده‌ی کنترل‌های مخصوص به خود باشد. به عنوان مثال چندین کنترل را به سربرگ اول مانند شکل زیر اضافه کنید سپس بر روی سربرگ دیگر کلیک کرده و کنترل‌های دیگری را به آن اضافه نمایید.

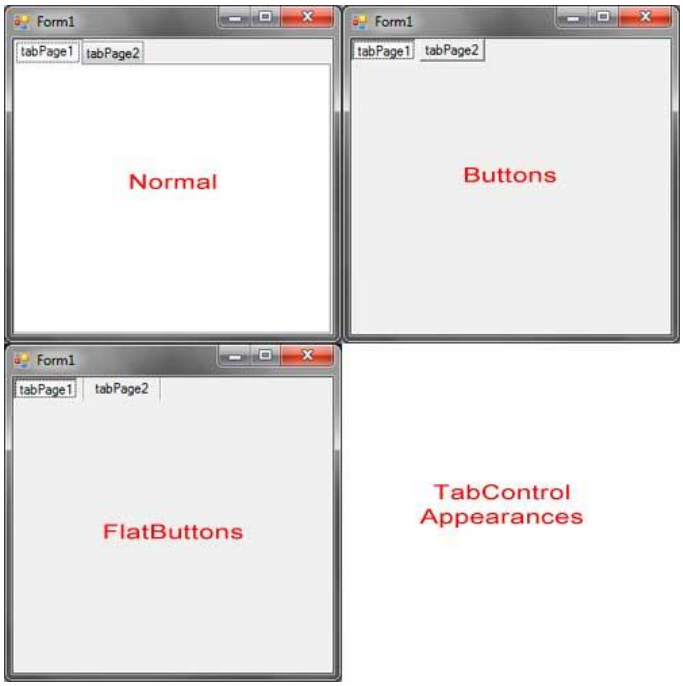


حال اجازه دهید در مورد خود TabControl توضیح دهیم. در زیر برخی از خواص مهم این کنترل ذکر شده‌اند:

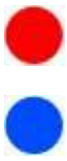
خاصیت	توضیح
Alignment	ناحیه ای که Tabها بر اساس آن تراز بندی می‌شوند. مقدار پیش‌فرض آن top است.
Appearance	ظاهر Tabها را تعیین می‌کند.
ImageList	شامل لیستی از عکس‌هایی است که می‌توانید برای هر سربرگ تعیین کنید.
ItemSize	اندازه tab را تعیین می‌کند.
Multiline	به شما اجازه می‌دهد tabها را در چندین سطر نمایش دهید. هنگامی کاربرد دارد که تعداد tabها زیاد باشد.

SelectedIndex	اندیس tab جاری را تعیین می‌کند.
SelectedTab	Tab جاری را تعیین یا بر می‌گرداند.
TabCount	تعداد tabهای کنترل را بر می‌گرداند.
TabPages	به شما اجازه مدیریت (حذف و اضافه کردن) tabها را می‌دهد.

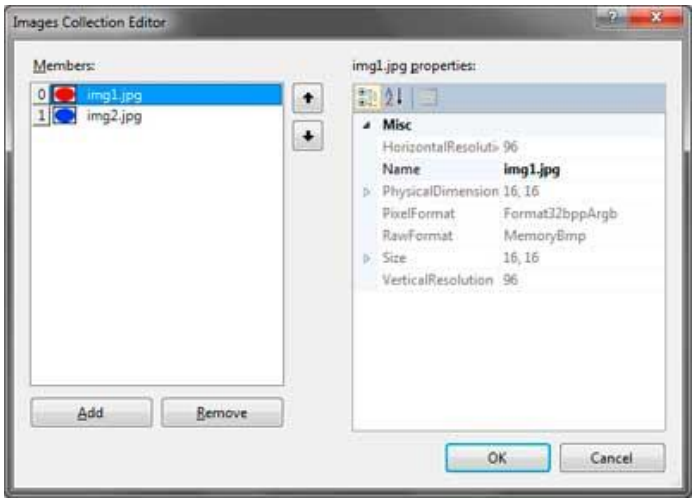
خاصیت Alignment مکان قرار گیری tabها را تعیین می‌کند. در حالت پیش‌فرض مقدار این خاصیت برابر top است که باعث نمایش tabها در بالای tabcontrol می‌شود. خاصیت Appearance شکل ظاهری tabها را تعیین می‌کند. این خاصیت سه مقدار Normal (پیشفرض)، Buttons (نمایش tabها به شکل دکمه)، FlatButtons (نمایش tabها به شکل دکمه‌های مسطح). حالت‌های مختلف این ۳ مقدار را در شکل زیر نمایش داده شده است.



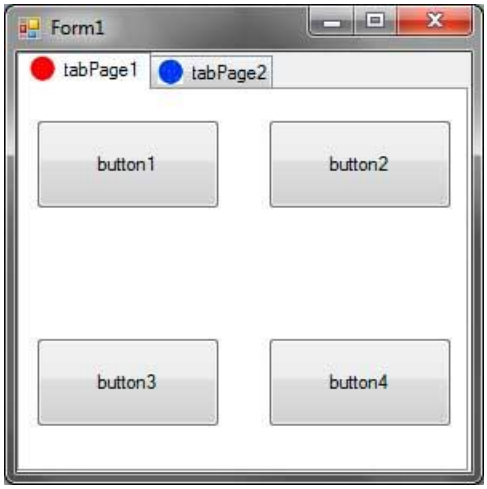
خاصیت ImageList به شما اجازه می‌دهد که برای هر tab، آیکونی را مشخص کنید. این خاصیت به یک کنترل ImageList ارجاع دارد. کنترل ImageList در این خاصیت که به طور مشابه در کنترل‌های دیگری مانند TreeView و منوها وجود دارد، مورد استفاده قرار می‌گیرد. به طور خلاصه نحوه‌ی استفاده از این کنترل برای نمایش عکس در TabControl را توضیح می‌دهیم. یک کنترل ImageList را از جعبه ابزار بر روی فرم قرار دهید. این کنترل شمای ظاهری ندارد و در قسمت کنترل‌های غیر بصری (Componenttray) قرار می‌گیرد. ImageList از دو عکس زیر استفاده می‌کند.



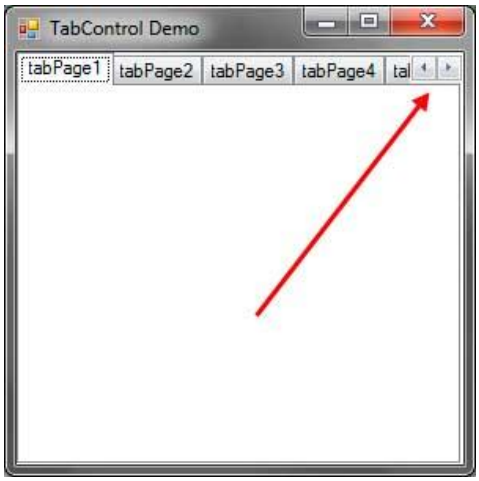
این دو عکس را بر روی قسمتی از هارد خود ذخیره و سپس بر روی کنترل ImageList کلیک کنید. خاصیت Images را از پنجره‌ی Properties پیدا کنید و بر روی دکمه کنار آن کلیک کنید. با کلیک بر روی این دکمه پنجره‌ی Image Collection Editor نمایش داده می‌شود.



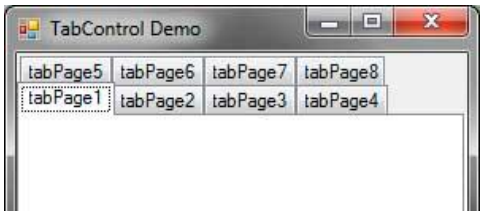
بر روی دکمه‌ی Add برای اضافه کردن یک عکس کلیک کنید. برای هر دو عکسی که قبلاً بر روی هارد ذخیره کرده‌اید، این مرحله را تکرار کنید و در آخر بر روی دکمه‌ی OK کلیک کنید. سپس خاصیت ImageList از کنترل TabControl را پیدا کرده و ارجاع این کنترل را به آن نسبت دهید. برای قرار دادن عکس‌ها در کنار هر tab روی tab مورد نظر کلیک کرده و سپس در پنجره‌ی Properties خاصیت ImageIndex آن را پیدا کنید. حال بر روی دکمه‌ی پایین افتادنی کنار آن کلیک و عکس مورد نظر خود را انتخاب نمایید. برای چند tab مختلف می‌توانید عکس مشابه‌ای را انتخاب کنید.



برای تغییر اندازه‌ی هر تب آن را انتخاب کنید و خاصیت ItemSize آنرا تغییر دهید. این کار زمانی مفید است که اندازه عکس‌های کناری tab بسیار بزرگ باشد. اگر تعداد tabها کنترل بسیار زیاد و از اندازه‌ی عرض فرم بیشتر باشد، دکمه‌های فلش دار در سمت راست و بالای tabcontrol نمایش داده می‌شوند و شما می‌توانید با استفاده از آنها به تب‌های دیگر دسترسی یابید.



برای نمایش tabها در چند سطر می‌توانید از خاصیت MultiLine استفاده کنید.



خاصیت TabPages به شما اجازه حذف یا اضافه کردن tabها را می‌دهد. در زیر با کدنویسی tabها مختلفی را ایجاد کرده و به کنترل اضافه می‌کنیم.

```
TabPage tab1 = new TabPage("Tab 1");
TabPage tab2 = new TabPage("Tab 2");
TabPage tab3 = new TabPage("Tab 3");
tabControl1.TabPages.Add(tab1);
tabControl1.TabPages.Add(tab2);
tabControl1.TabPages.Add(tab3);
```

خصوصیت TabPages دارای نوع TabPageCollection است که مجموعه‌ای از اشیاء TabPage می‌باشد. بنابراین شما می‌توانید از متدهای عمومی مانند Add()، Remove()، AddRange() استفاده کنید. برای تعیین تب جاری می‌توان از خصوصیت SelectedIndex استفاده کرد. این خصوصیت اندیس tab جاری را تعیین می‌کند.

```
MessageBox.Show("The selected tab is " + tabControl1.TabPages[tabControl1.SelectedIndex].Text);
```

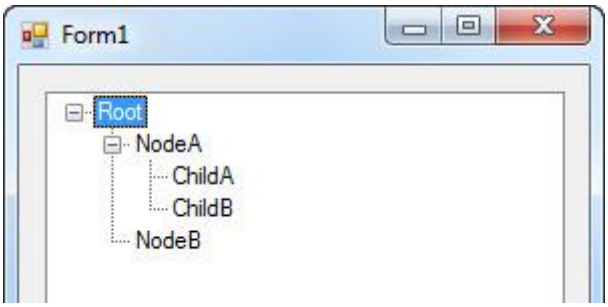
خصوصیت SelectedTab شیء tab جاری را تعیین یا بر می‌گرداند.

```
TabPage selectedTab = tabControl1.SelectedTab;
MessageBox.Show("The selected tab is " + selectedTab.Text);
```

خصوصیت TabCount تعداد تب‌های کنترل را بر می‌گرداند. وقتی که در بین tabهای مختلف حرکت می‌کنید، رویداد SelectedIndexChanged اتفاق می‌افتد. کنترل TabControl زمانی مفید است که فرم از قسمت‌های مختلفی تشکیل شده باشد و فضای کافی در اختیار نداشته باشید. به عنوان یک مثال زمانی که برنامه دارای قسمتی برای تعیین تنظیمات خود باشد می‌توانید از این کنترل استفاده کنید.

کنترل TreeView

کنترل treeView برای نمایش درختی آیتم‌ها به کار می‌رود. برای درک بهتر عملکرد این کنترل می‌توان به فهرست مطالب یک کتاب و یا نمایش فایل‌ها و یا پوشه‌های درون هارد اشاره کرد. هر آیتم در این کنترل شامل یک برچسب و یک عکس اختیاری است و هر آیتم خود می‌تواند شامل زیر آیتم‌های دیگر باشد. با کلیک بر روی علامت + یا – کنار هر آیتم، کاربر می‌تواند زیر آیتم‌های مربوطه را مشاهده کند. در شکل زیر یک کنترل treeview ساده با چند آیتم نمایش داده شده است. همانطور که مشاهده می‌کنید، آیتم Root آیتم یا گره اصلی است که دارای دو زیر آیتم به نام‌های NodeA و NodeB می‌باشد. NodeA نیز به نوبه خود دارای دو زیر آیتم به نام‌های ChildA و ChildB است.



بعد از ایجاد treeView می‌توان هر یک از آیتم‌های آن را دستکاری کرد (حذف، اضافه، ویرایش و مرتب کرد). در زیر به برخی از خاصیت‌های این کنترل اشاره شده است:

خاصیت	توضیح
CheckBox	یک جعبه تیک یا چک باکس در کنار هر گره نمایش می‌دهد.
ItemHeight	ارتفاع گره‌های کنترل را مشخص می‌کند.
LabelEdit	مشخص می‌کند که آیا متن گره‌ها قابل ویرایش باشد یا نه؟
LineColor	رنگ خطی که به گره‌ها متصل است را مشخص می‌کند.
Nodes	مجموعه‌ای از گره‌های کنترل را بر می‌گرداند.
Scrollable	مشخص می‌کند که آیا کنترل دارای اسکرول باشد یا نه؟
SelectedImageIndex	مقدار اندیس عکسی که قرار است هنگام انتخاب گره، در کنار گره نمایش داده شود، را نمایش می‌دهد.
SelectedNode	گره جاری انتخاب شده در کنترل را مشخص می‌کند.
ShowLines	مشخص می‌کند که آیا خط ارتباط دهنده بین گره‌ها، نمایش داده شود یا خیر؟
ShowNodeToolTips	مشخص می‌کند که آیا هنگام مکث ماوس بر روی گره‌ها متنی نمایش داده شود یا نه؟

ShowPlusMinus	مشخص می‌کند که آیا علائم + و منفی در کنار گره‌ها نمایش داده شوند یا نه؟
ShowRootLines	مشخص می‌کند که آیا خط زیر گره اصلی، نمایش داده شود یا خیر؟
Sorted	مشخص می‌کند که آیا گره‌ها در داخل کنترل مرتب نمایش داده شوند یا نه؟
TreeViewNodeSorter	با پیاده سازی اینترفیس IComparer اجازه می‌دهد که گره‌ها را به صورت سفارشی مرتب سازی کنید.

در جدول زیر هم برخی از متدهای این کنترل ذکر شده است:

متد	توضیح
CollapseAll	همه گره‌های باز شده را جمع می‌کند.
ExpandAll	همه گره‌های بسته شده را باز می‌کند.
GetNodeCount	تعداد تمامی گره‌ها و زیر گره‌های کنترل را بر می‌گرداند.

و در زیر هم روی رویدادهای مهم کنترل treeView آمده است:

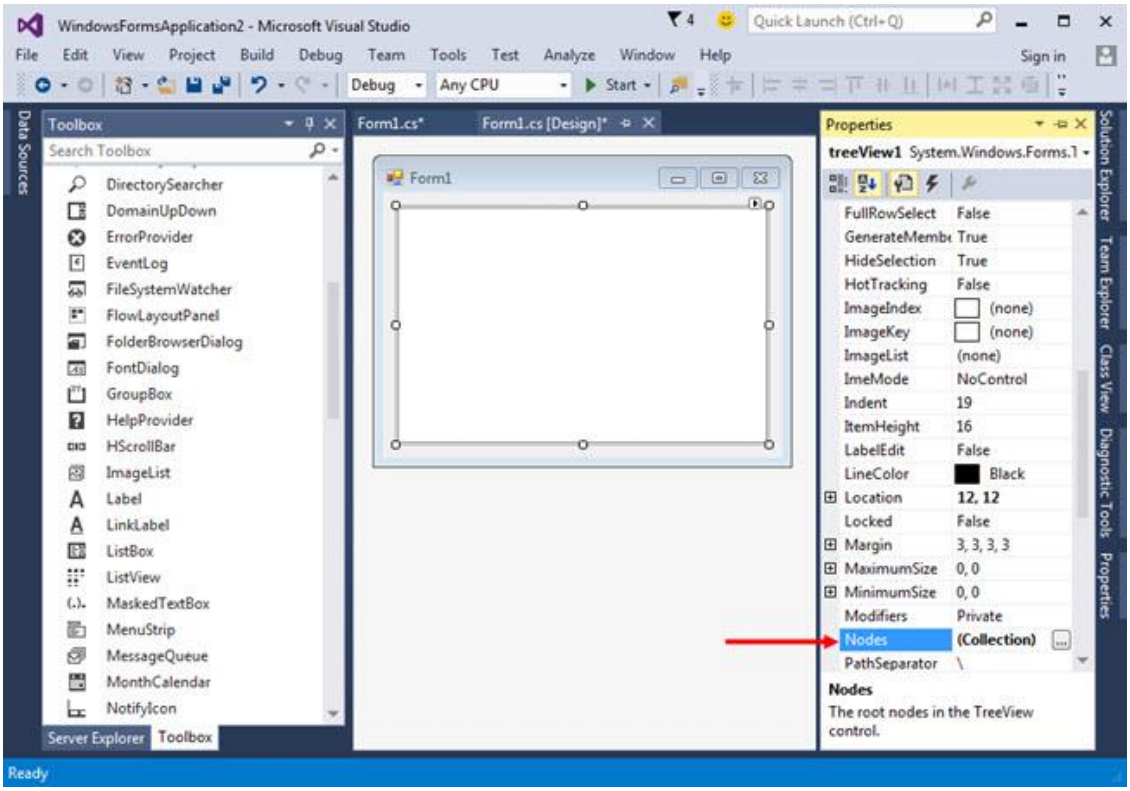
رویداد	توضیح
AfterCheck	بعد از تیک خوردن جعبه کنار گره، روی می‌دهد.
AfterCollapse	بعد از بسته شدن گره‌ها، روی می‌دهد.
AfterExpand	بعد از باز شدن گره‌ها روی می‌دهد.
AfterLabelEdit	بعد از ویرایش متن گره‌ها روی می‌دهد.
AfterSelect	بعد از انتخاب یک گره روی می‌دهد.
BeforeCheck	قبل از تیک حوردن یک گره روی می‌دهد.
BeforeCollapse	قبل از بسته شدن یگ گره روی می‌دهد.
BeforeExpand	قبل از باز شدن یک گره روی می‌دهد.
BeforeLabelEdit	قبل از ویرایش متن یک گره روی می‌دهد.
BeforeSelect	قبل از انتخاب یک گره روی می‌دهد.
NodeMouseClick	وقتی روی می‌دهد که بر روی یک گره کلیک شود.

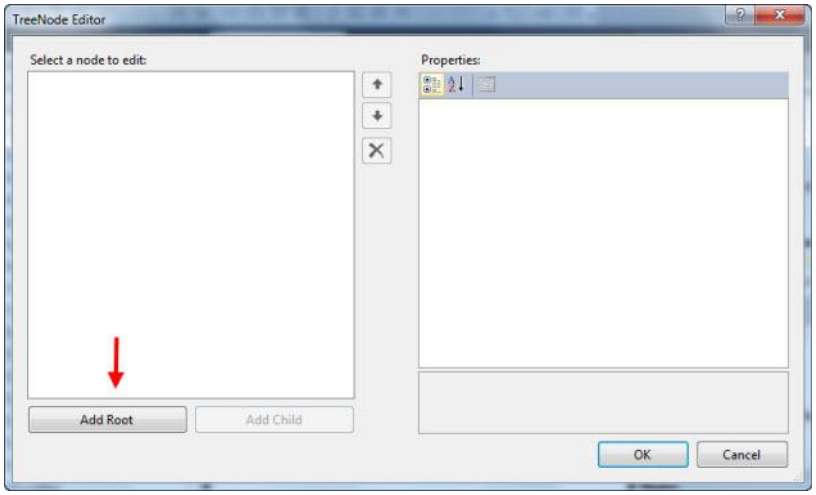
NodeMouseDoubleClick	وقتی روی می‌دهد که بر روی یک گره دو بار کلیک شود.
NodeMouseHover	وقتی روی می‌دهد که با ماوس بر روی یک گره برویم.

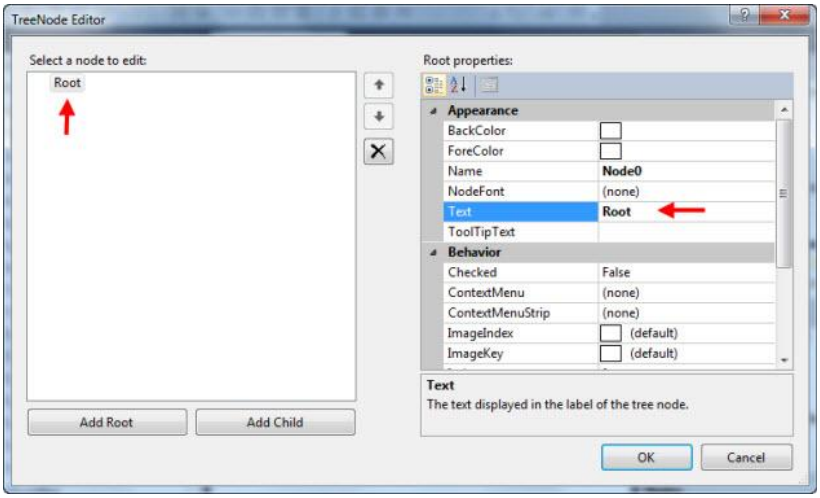
برای اضافه کردن آیتم یا گره به یک کنترل treeView می‌توان به دو روش عمل کرد:

- استفاده از پنجره Properties
- استفاده از کدنویسی

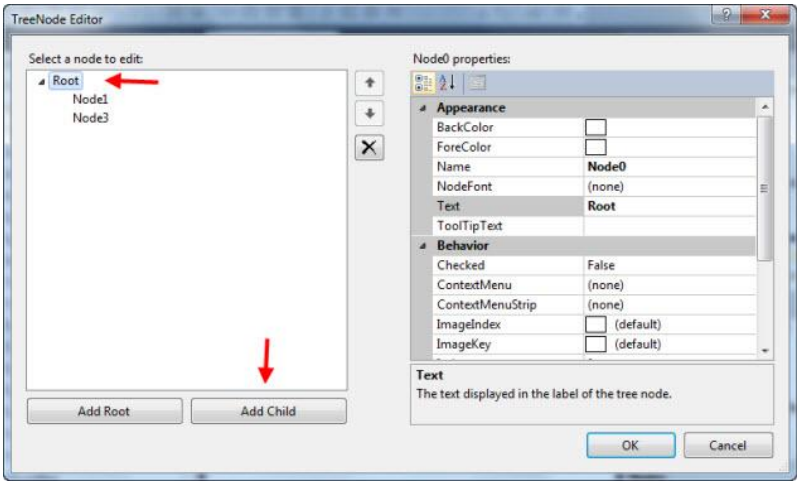
ابتدا روش اول را توضیح می‌دهیم. یک کنترل treeView بر روی فرم قرار داده و آن را در حالت انتخاب قرار دهید. سپس در پنجره Properties بر روی دکمه ای که در خاصیت Nodes قرار دارد کلیک کنید. بعد از کلیک، پنجره‌ای به صورت زیر باز می‌شود که شما می‌توانید با استفاده از آن گره‌های لازم را به کنترل treeView اضافه کنید. فرض کنید می‌خواهیم گره‌های شکل بالا را ایجاد کنیم. ابتدا به صورت زیر گره اصلی (Root) را ایجاد و نام گذاری می‌کنیم:



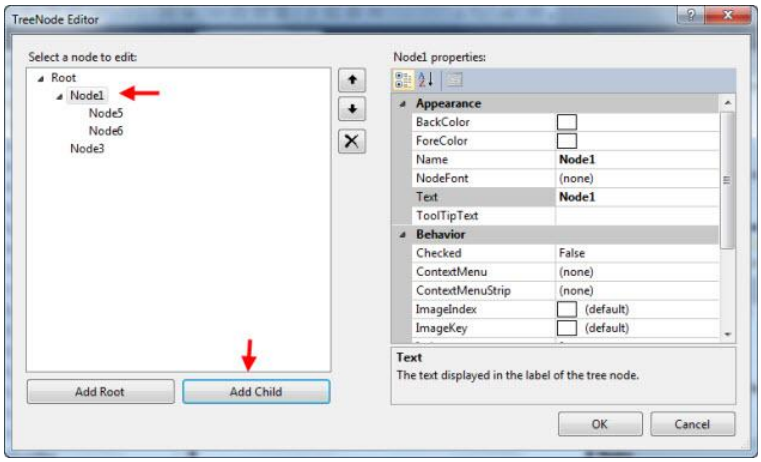




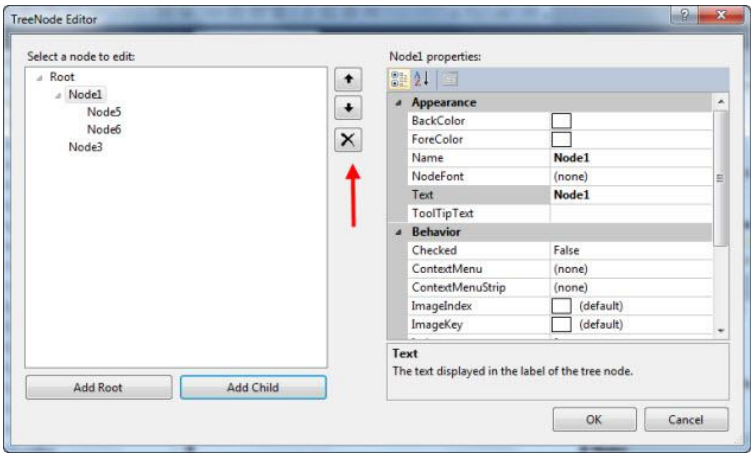
بعد از ایجاد گره اصلی، برای اضافه کردن دو زیر گره ابتدا بر روی گره اصلی کلیک کرده و سپس دو بار کلیک بر روی دکمه `add Child` دو زیر گره به آن اضافه می‌کنیم:



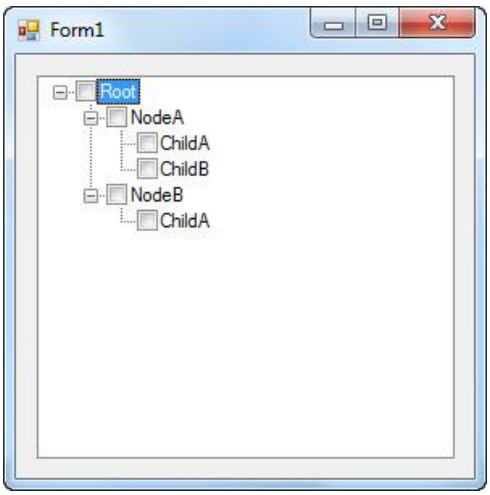
برای اضافه کردن یک زیر گره به گره جدید ایجاد شده هم ابتدا بر روی آن کلیک کرده و سپس دکمه `add Child` را می‌زنیم.



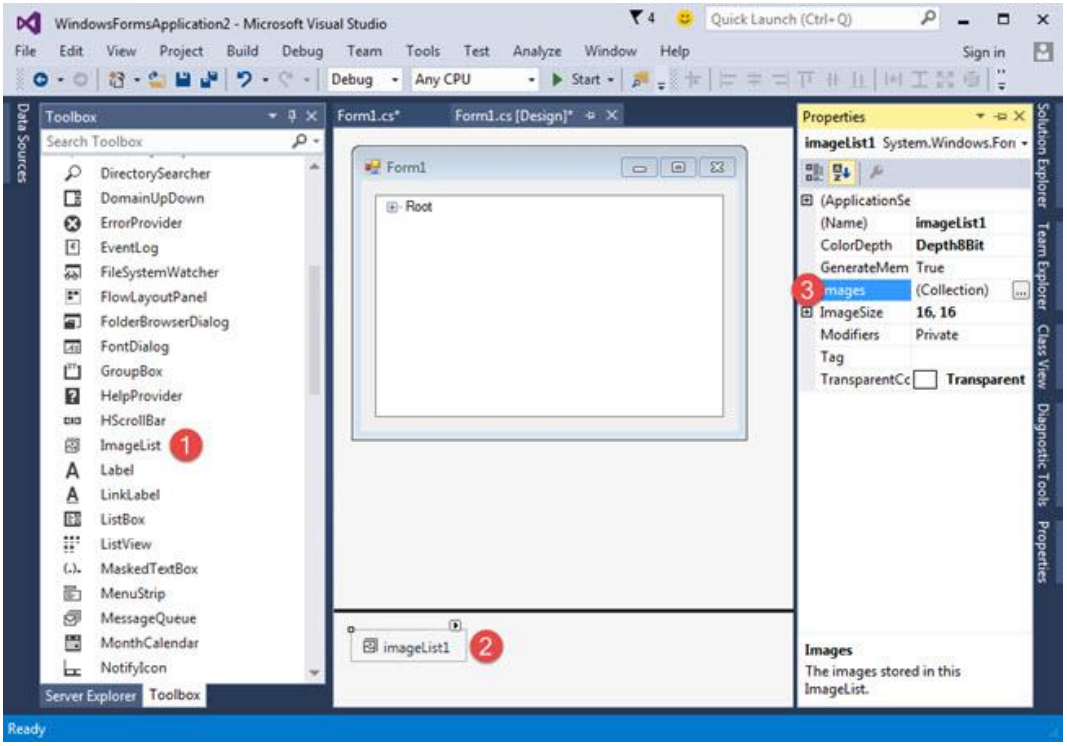
به این نکته توجه کنید که برای اضافه کردن یک زیر گره، حتماً باید گره‌ای که قرار است یک زیر گره به آن اضافه شود در حالت انتخاب باشد. برای حذف گره‌های اضافی و ناخواسته و همچنین جابه جایی گره‌ها از دکمه‌های زیر استفاده می‌شود:

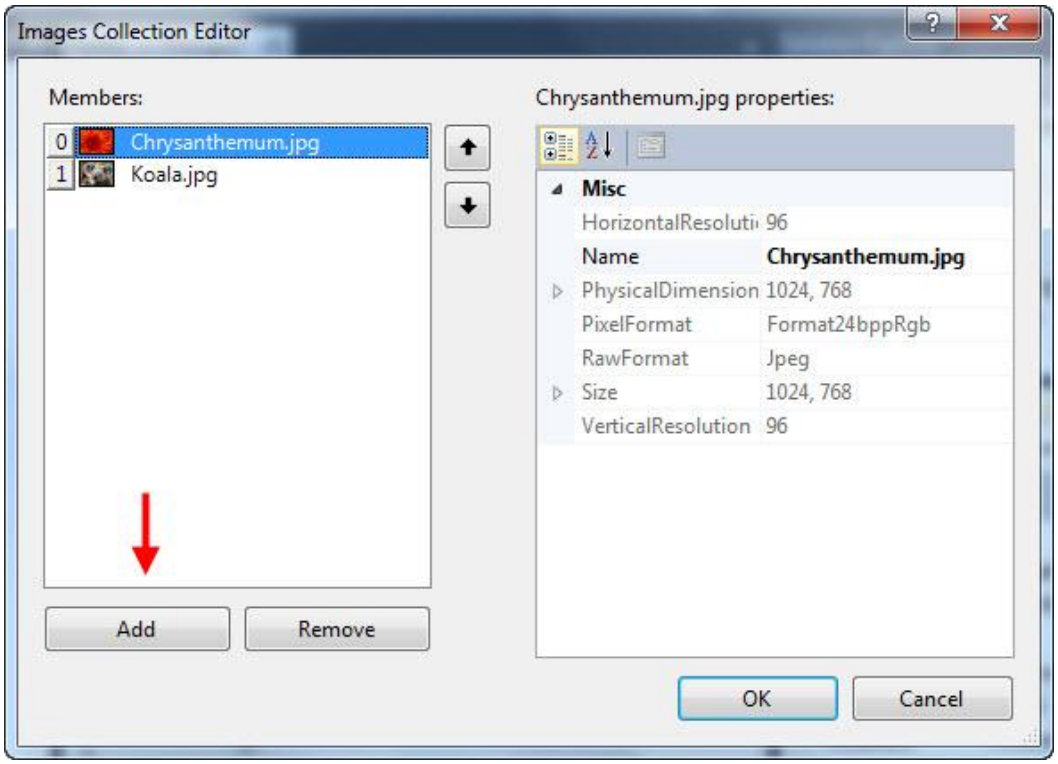


برای اینکه در کنار هر گره یک جعبه متن قرار دهیم، می‌توانیم از خاصیت `CheckBoxes` در پنجره `Properties` استفاده کرده و مقدار آن را برابر `True` قرار دهیم:

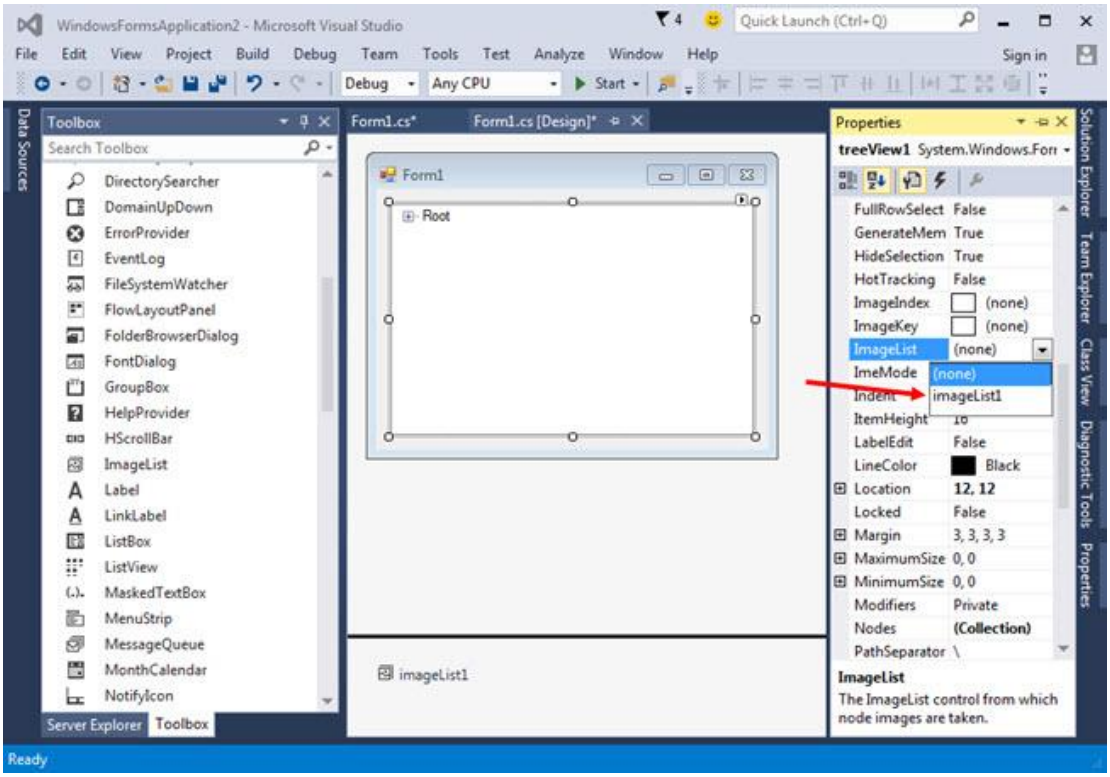


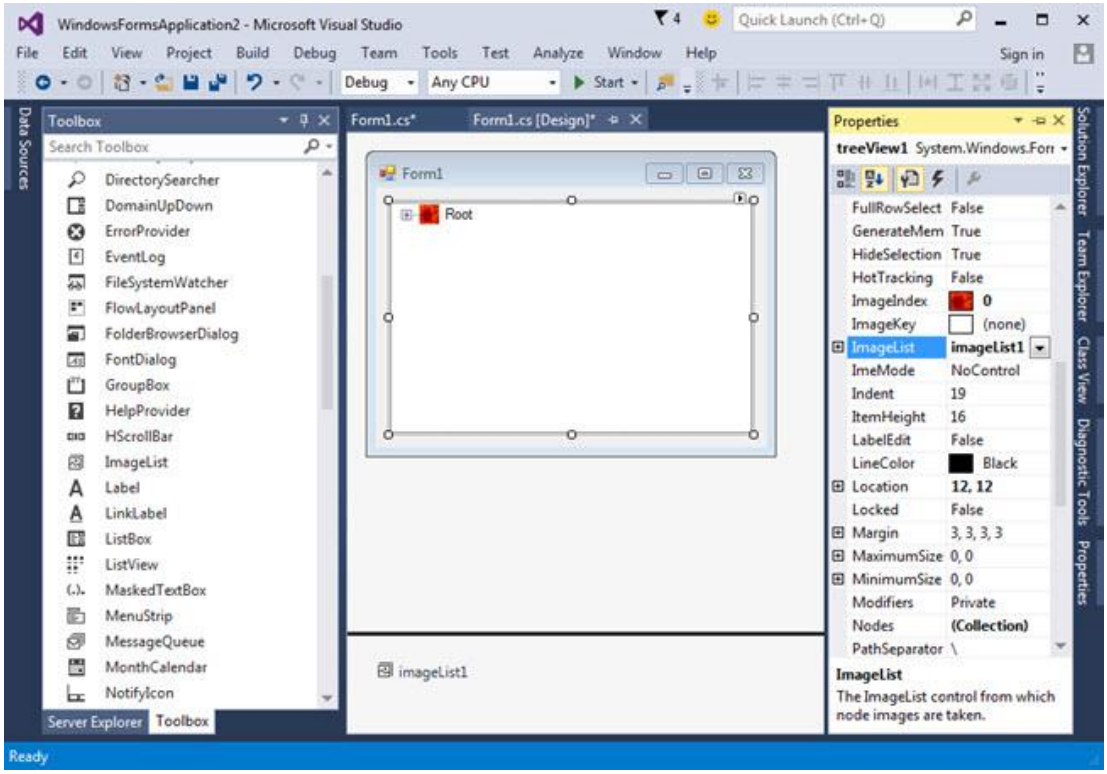
اگر بخواهیم یک عکس به گره‌ها اختصاص دهیم، ابتدا باید یک کنترل `imageList` بر روی فرم بکشیم و سپس با استفاده از خاصیت `Image` این کنترل عکسی را که می‌خواهیم انتخاب کنیم:





در مرحله بعد بر روی کنترل `treeView` کلیک کرده و از خاصیت `ImageList` آن را بر روی نام کنترل `imageList` اضافه شده به فرم قرار می‌دهیم تا عکس به گره‌ها اضافه شود:

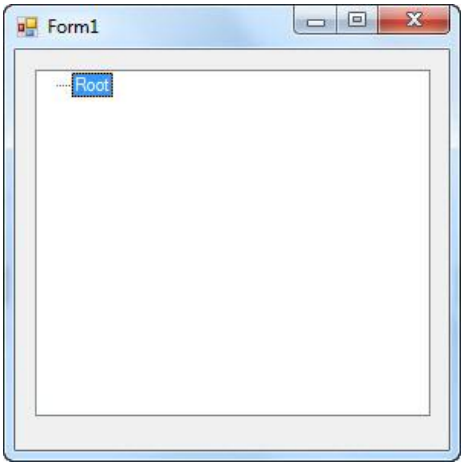




حال همین مراحل بالا را می‌خواهیم با استفاده از روش کدنویسی انجام دهیم. یک برنامه ویندوزی ایجاد کرده و یک کنترلtreeView بر روی فرم قرار دهید. سپس مثلاً بر روی فرم دو بار کلیک کرده و کد زیر را بنویسید:

```
treeView1.Nodes.Add("Root");
```

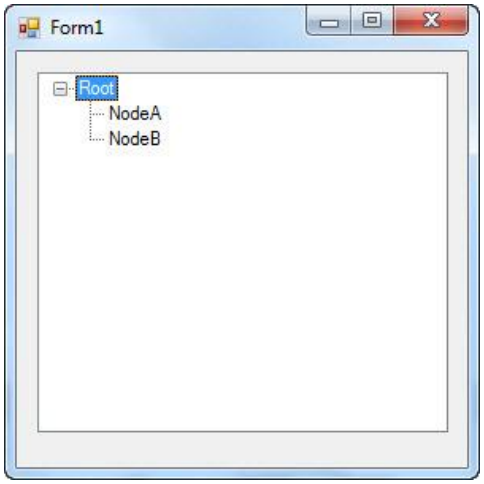
با اجرای برنامه مشاهده می‌کنید که گره اصلی به فرم اضافه شده است:



برای اضافه کرده دو زیر گره به این گره اصلی هم به صورت زیر عمل می‌کنید:

```
treeView1.Nodes[0].Nodes.Add("NodeA");
treeView1.Nodes[0].Nodes.Add("NodeB");
```

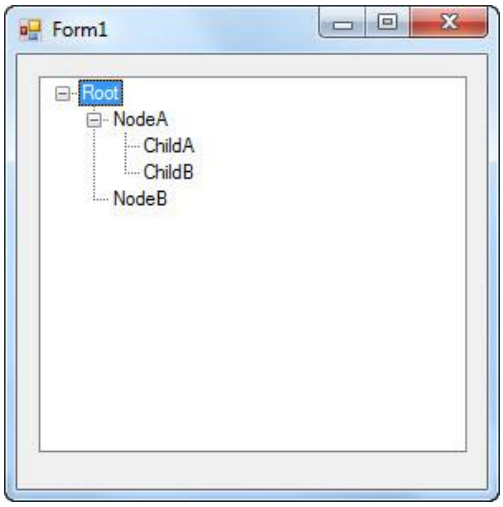
عدد صفر در کد بالا، اندیس گره اصلی (Root) است:



و برای اضافه کردن زیر گره به گره NodeA به صورت زیر عمل می‌کنیم:

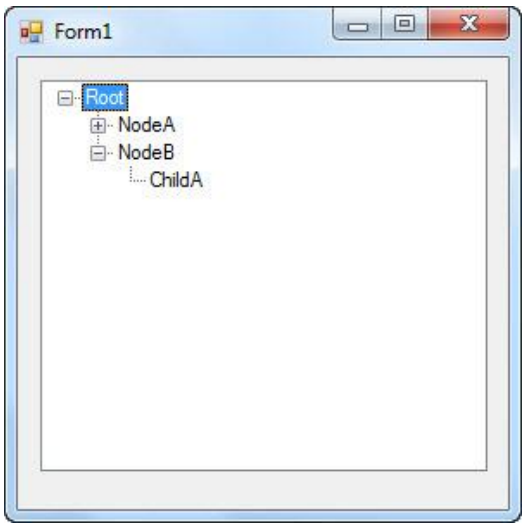
```
treeView1.Nodes[0].Nodes[0].Nodes.Add("ChildA");
treeView1.Nodes[0].Nodes[0].Nodes.Add("ChildB");
```

همانطور که در کد بالا مشاهده می‌کنی اولین صفر نشان دهنده اندیس گره Root و دومین صفر نشان دهنده اولین زیر گره آن یعنی NodeA است:



از کدهای بالا نتیجه می‌گیریم که برای اضافه کردن یک زیر گره به گره NodeB باید به جای عدد صفر، عدد یک قرار دهیم:

```
treeView1.Nodes[0].Nodes[1].Nodes.Add("ChildA");
```



در کل کدهای زیر گره‌های شکل بالا را به وجود می‌آورند:

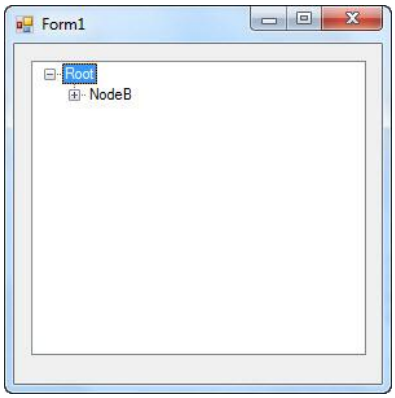
```
using System.Windows.Forms;

namespace TreeView
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            treeView1.Nodes.Add("Root");
```

```
            treeView1.Nodes[0].Nodes.Add("NodeA");
            treeView1.Nodes[0].Nodes[0].Nodes.Add("ChildA");
            treeView1.Nodes[0].Nodes[0].Nodes.Add("ChildB");

            treeView1.Nodes[0].Nodes.Add("NodeB");
            treeView1.Nodes[0].Nodes[1].Nodes.Add("ChildA");
        }
    }
}
```

برای حذف یک گره هم می‌توانیم از متد Remove() استفاده کنیم. مثلاً اگر بخواهیم گره NodeA را حذف کنیم به صورت زیر عمل می‌نماییم:

```
treeView1.Nodes[0].Nodes[0].Remove();
```

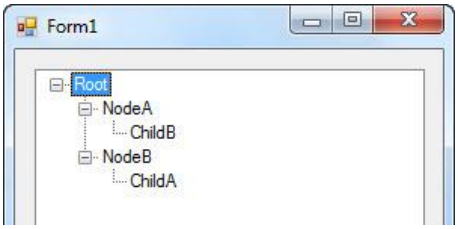
همانطور که مشاهده می‌کنید، اندیس گره را به صورت بالا نوشته و بعد متد Remove() را فراخوانی می‌کنیم. به این نکته توجه کنید که با حذف یک گره مادر تمام زیر گره‌های آن هم حذف می‌شوند:



مطمئناً تا کنون متوجه شده‌اید که برای حذف مثلاً زیر گره ChildA، باید به صورت زیر، سلسه مراتب وراثتی آن را رعایت کنیم:

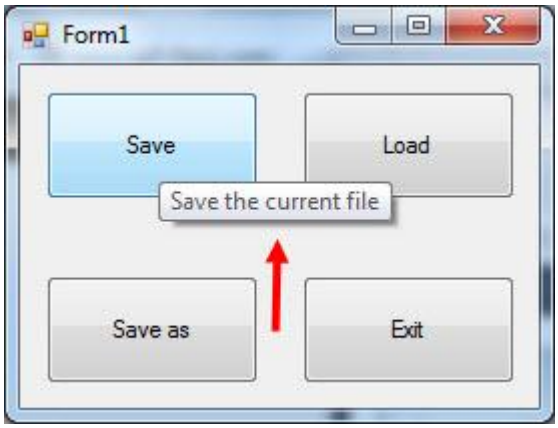
```
treeView1.Nodes[0].Nodes[0].Nodes[0].Remove();
```

در کد بالا به ترتیب از چپ به راست، صفر اول اندیس گره Root، صفر دوم اندیس گره NodeA و صفر آخر هم اندیس ChildA است:

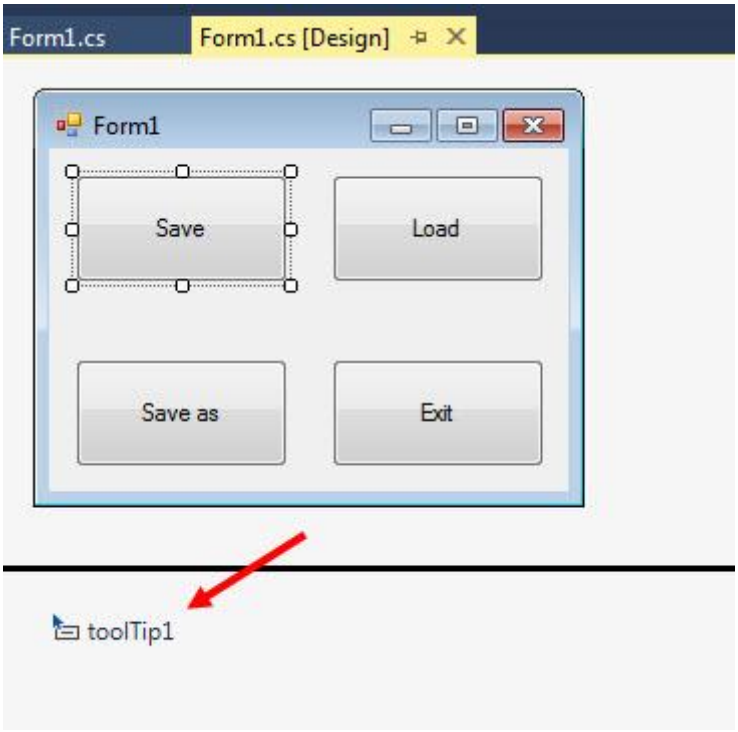


کنترل ToolTip

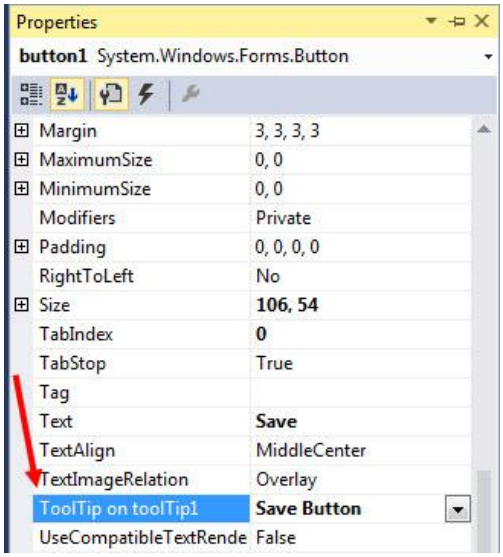
از کنترل ToolTip برای نمایش توضیحاتی در مورد هر یک از کنترل‌های به کار برده شده در GUI استفاده می‌شود. Tooltipها یا همان توضیحات در مورد کنترل‌ها، در بسیاری از برنامه‌های کاربردی دیده می‌شود و موقعی ظاهر می‌شوند که نشانگر ماوس را بر روی کنترل‌ها نگه داریم.



این کنترل در دسته common controls ویژوال استودیو قرار دارد. هنگامی که یک کنترل ToolTip را بر روی فرم می‌کشید، مشاهده می‌کنید که به قسمتی که در شکل زیر نشان داده شده است، منتقل می‌شود.



سپس می‌توان از آن در کنترل‌های مختلف استفاده کرد. هنگامیکه این کنترل به فرم اضافه شد، یک خاصیت جدید به نامToolTip به هر کنترل در فرم اضافه می‌شود. این خاصیت را می‌توانید در پنجره properties مشاهده کنید.

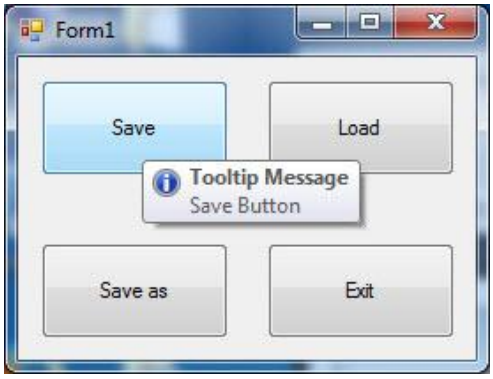


این خاصیت جدید، تعیین کننده همان متنی است که اگر با ماوس بر روی کنترل مکث کنیم، نمایش داده می‌شود. با اضافه کردن هر tooltip به فرم یک خاصیت tooltip به هر کنترل اضافه می‌شود. به عنوان مثال اگر شما دو کنترل ToolTip به فرم اضافه کنید، هر کنترل موجود در فرم دارای دو خاصیت ToolTip می‌شود. برخی از خواص کنترل ToolTip در جدول زیر آمده است:

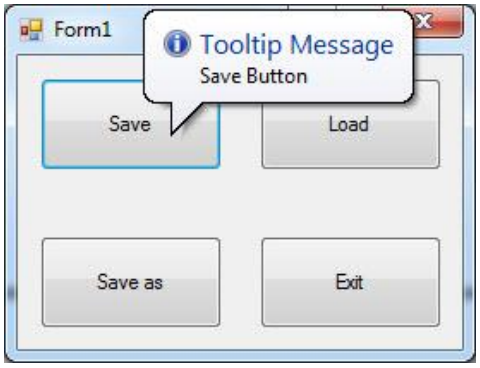
خاصیت	توضیح
Active	از این خاصیت برای فعال یا غیر فعال کردن tooltip استفاده می‌شود .
AutomaticDelay	مقدار این خاصیت بر روی سایر خصوصیات مربوط به تأخیر در نمایش، تأثیر می‌گذارد.
AutoPopDelay	مدت زمان نمایش tooltip را تعیین می‌کند.
InitialDelay	مدت زمانی که نشانگر باید بر روی کنترل باشد تا tooltip نمایش داده شود.
IsBalloon	مشخص می‌کند که آیا tooltip از پنجره‌ی به شکل بالون برای نمایش متن استفاده کند یا نه.
ReshowDelay	مدت زمان نمایش tooltip در هنگام جابه جایی نشانگر بین کنترل‌ها را تعیین می‌کند.
ShowAlways	اگر این گزینه را برابر true قرار دهید، در صورت غیر فعال بودن کنترل در برگیرنده کنترل، باز هم tooltip نمایش داده می‌شود.
ToolTipIcon	برای نمایش icon در tooltip به کار می‌رود.
ToolTipTitle	یک عنوان به متن داخل tooltip اضافه می‌کند.
UseFading	هنگامی که این خاصیت را برابر true قرار دهید، پنجره tooltip با افکت محو (fade effect) نمایش داده می‌شود.

برای اینکه یک Tooltip نمایش داده شود باید مقدار خاصیت Active آن true شود. خاصیت AutoPopDelay مدت زمان نمایش Tooltip وقتی که ماوس بر روی کنترل قرار می‌گیرد را مشخص می‌کند. خاصیت InitialDelay مشخص می‌کند که چه مدت زمان ماوس بر روی کنترل قرار بگیرد و سپس Tooltip نمایش داده شود. خاصیت ReshowDelay مدت زمان دوباره نشان داده شدن Tooltip وقتی که از یک کنترل به کنترل دیگر می‌روید را مشخص می‌کند. خاصیت AutomaticDelay به طور خودکار سه خاصیت قبل را متعادل می‌کند.

خاصیت AutoPopDelay ده برابر مقدار AutomaticDelay است. مقدار خاصیت InitialDelay برابر مقدار AutomaticDelay می‌شود و مقدار خاصیت ReshowDelay نصف مقدار خاصیت AutomaticDelay است. به این نکته توجه کنید که این خاصیت‌ها مقداری از نوع صحیح دریافت می‌کنند و این مقدار بر حسب میلی ثانیه است. همچنین شما با استفاده از خواص ToolTipTitle و ToolTipIcon می‌توانید برای tooltip عنوان و آیکن تعیین کنید.



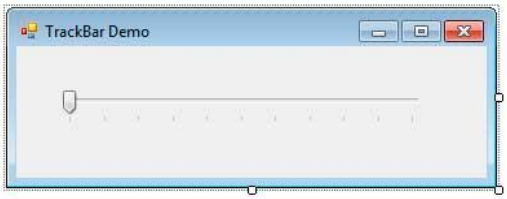
همچنین ظاهر tooltip را با استفاده از خاصیت IsBalloon تغییر داد.



شما می‌توانید چندین کنترل tooltip را به فرم اضافه کرده که هر کنترل دارای tooltip ی با خواص متفاوت باشد. همچنین یک کنترل می‌تواند از چندین tooltip به طور همزمان استفاده کند، البته این روش توصیه نمی‌شود.

کنترل TrackBar

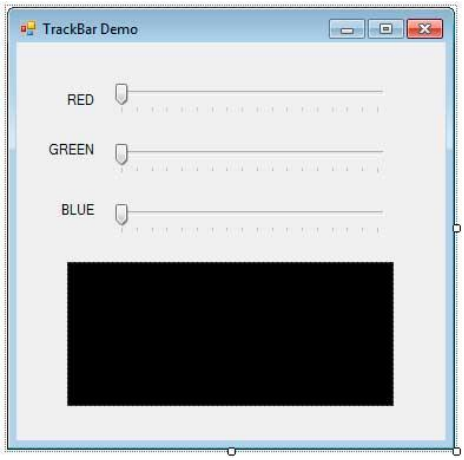
کنترل TrackBar (System.Windows.Forms.TrackBar) شبیه یک نوار لغزنده با یک دستگیره است که با استفاده از این دستگیره می‌توان مقدار آن را تعیین کنید. ناحیه ای که دستگیره به آن اشاره می‌کند، نشان دهنده مقدار جاری کنترل است. شکل این کنترل به صورت زیر است:



کاربر می‌تواند دستگیره به در جهت افقی حرکت دهد و همچنین می‌توان جهت حرکت را به صورت عمودی تغییر داد. مقدار این کنترل در حالت افقی از سمت چپ به راست و در حالت عمودی از پایین به بالا افزایش می‌یابد. در زیر تعدادی از خواص این کنترل را مشاهده می‌کنید.

خاصیت	توضیح
LargeChange	مقدار این خاصیت مشخص می‌کند که مقدار TrackBar با کلیک کردن روی آن یا با استفاده از دکمه‌های PageDown یا PageUp چه مقدار افزایش یابد.
Maximum	بیشترین مقداری که به TrackBar اختصاص داده‌ایم.
Minimum	کمترین مقداری که به TrackBar اختصاص داده‌ایم.
Orientation	نحوه نمایش TrackBar را تعیین می‌کند (افقی یا عمودی).
SmallChange	مقداری که در هنگام فشار دادن کلیدهای جهت نما به TrackBar اضافه یا کم می‌شود را تعیین می‌کند.
TickFrequency	تعداد خطوط عمودی که در زیر TrackBar نمایش داده می‌شوند.
Value	مقدار جاری TrackBar را نشان می‌دهد.

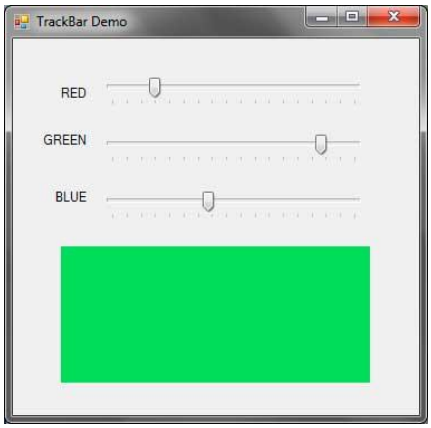
هنگامی که شما دستگیره کنترل را جابه جا می‌کنید رویداد Scroll اتفاق می‌افتد. در مثال زیر، نحوه‌ی استفاده از کنترل TackBar به شما نشان داده می‌شود. یک پروژه Windows application ایجاد کرده و سه کنترل TrackBar و سه Label به آن اضافه کنید. همچنین یک کنترل Panel به فرم اضافه کنید و خاصیت BackColor آن را به Black تغییر دهید.



خاصیت Name آنها را به ترتیب به trackBarRed، trackBarGreen و trackBarBlue و خاصیت Maximum را برابر ۲۵۵ و خاصیت TickFrequency هر یک را به ۱۵ تغییر دهید. به محیط کد نویسی رفته و کنترل کننده رویداد زیر را برای استفاده سه TrackBar بنویسید.

```
private void trackBar_Scroll(object source, EventArgs e)
{
    int red = trackBarRed.Value;
    int green = trackBarGreen.Value;
    int blue = trackBarBlue.Value;

    Color color = Color.FromArgb(red, green, blue);

    panelColor.BackColor = color;
}
```

به محیط طراحی برگردید و سه کنترل TrackBar را انتخاب و در قسمت Events مربوط به پنجره Properties رویداد Scroll را یافته و بر روی آن دو بار کلیک کنید. در داخل کنترل کننده رویداد، مقادیر خاصیت Value از هرکدام از Trackbarها را در سه متغیر ذخیره کرده‌ایم. مقادیر این سه متغیر نشان دهنده رنگ‌های قرمز، سبز، آبی (RGB values) می‌باشد. سپس یک شیء از کلاس System.Drawing.Color را ساخته و از مقادیر TrackBarها به عنوان پارامترهای متد FromArgb() استفاده می‌کنیم. سپس رنگ Panel را به رنگی که بر اساس مقادیر TrackBarها ایجاد شده است، تغییر می‌دهیم. برنامه را اجرا و دستگیره‌های TrackBarها را جابه جا و نتیجه را مشاهده کنید.

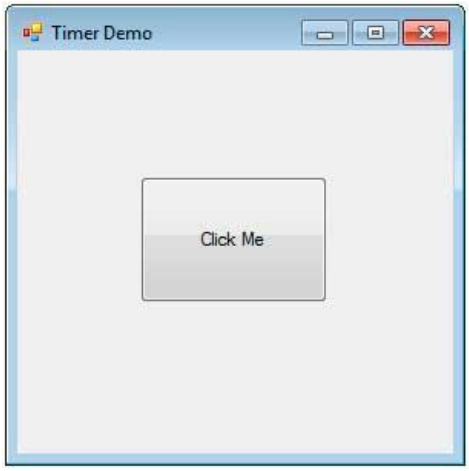


کنترل Timer

کنترل Timer (System.Windows.Forms.Timer) دستور یا دستورات خاصی را در بازه‌های زمانی خاصی که در خصوصیت Interval مشخص می‌شود اجرا می‌کند. به عنوان مثال، آگه بخواهید نامتان را هر ۵ ثانیه یکبار چاپ کنید، می‌توانید از این کنترل استفاده نمایید. کنترل Timer از کنترل‌های غیر بصری است و در قسمت Component Tray قرار می‌گیرد. این کنترل خصوصیات زیادی ندارد. دو خاصیت مهم آن عبارت‌اند از Enabled و Interval.

وقتی که خصوصیت Enabled را برابر true قرار دهید، کنترل شروع به کار و اگر برابر false قرار دهید، متوقف می‌شود. خصوصیت Interval مقدار زمانی که باید بگذرد تا دستور یا دستورات داخل رویداد Tick فراخوانی شود را مشخص می‌کند. رویداد Tick رویداد پیش‌فرض کنترل تایمر است. به این نکته توجه کنید که، مقدار این خصوصیت (interval) بر حسب میلی ثانیه مشخص می‌شود. به عنوان مثال قرار دادن مقدار ۵۰۰۰

در جلوی این خصوصیت به معنای این است که رویداد Tick هر ۵ ثانیه یکبار فراخوانی شود. به یک نمونه برنامه که در آن از کنترل timer استفاده شده است، توجه نمایید. برنامه شامل یک دکمه است که در جهت بالا و پایین فرم حرکت می‌کند. یک پروژه Windows Application با نام TimerDemo ایجاد کنید.. یک دکمه را در وسط فرم مانند شکل زیر قرار دهید.



یک کنترل Timer را به روی فرم بکشید. این کنترل در قسمت Component جعبه ابزار قرار دارد. همانطور که قبلاً گفته شد این کنترل در قسمت Component Tray قرار می‌گیرد.

کنترل timer را انتخاب و خاصیت Interval آن را برابر ۵۰ و خاصیت Enabled آنرا برابر true قرار دهید. این کار باعث می‌شود زمانی که برنامه اجرا شود، کنترل Timer شروع به کار کند. روی کنترل Timer دابل کلیک کنید تا کنترل کننده رویداد پیش‌فرض آن که همان رویداد Tick است ایجاد شود. به یاد داشته باشید دستورات داخل این کنترل کننده رویداد هر ۵۰ میلی ثانیه یکبار فراخوانی می‌شوند. یک نمونه متغیر به شکل زیر در کلاس فرم تعریف کنید. این متغیر برای نگهداری تعداد پیکسل‌هایی است که در هر بار فراخوانی رویداد Tick پیمایش می‌شوند.

```
private int velocity = 5;
```

کدهای زیر را به کنترل کننده رویداد Tick اضافه کنید.

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (button1.Top <= 0 || button1.Bottom > this.ClientSize.Height)
        velocity = -velocity;
    button1.Top += velocity;
}
```

وقتی که رویداد Tick فراخوانی می‌شود دستورات بدنه آن اجرا می‌شوند. در دستور شرط ابتدا بررسی شده است که دکمه به پایین فرم رسیده است یا به بالای فرم. برای اینکه مشخص کنیم که دکمه به بالای فرم چسبیده است از خاصیت Button.Top استفاده کرده‌ایم. این خصوصیت

مقدار مختصات y گوشه‌ی بالا و چپ دکمه را مشخص می‌کند. اگر مقدار این خصوصیت برابر یا کوچکتراز ۰ باشد، به این معنی است که، دکمه به بالای فرم رسیده است.

به این نکته توجه کنید که مبدأ مختصات در فرم گوشه بالای و سمت چپ فرم است. قسمت دوم شرط بررسی می‌کند که آیا دکمه به قسمت پایین فرم رسیده است یا نه؟ برای این کار از خاصیت Button.Bottom که مختصات y دکمه را بر می‌گرداند استفاده کرده‌ایم. سپس این مقدار را با مقدار خصوصیت Height (ارتفاع) فرم مقایسه کرده‌ایم. برای بدست آوردن ارتفاع فرم از خصوصیت Height که خود در خاصیت Form.ClientSize قرار دارد استفاده کرده‌ایم.

اگر هر کدام از شرط‌ها درست باشند به راحتی با قرینه کردن مقدار ضرب ۱ در مقدار فعلی velocity جهت حرکت دکمه را تغییر می‌دهیم. اگر دکمه به سمت بالا فرم حرکت کند (به این خاطر که ۵ پیکسل به سمت بالا حرکت می‌کند) و به بالای فرم برسد جهت حرکت آن تغییر می‌کند. برنامه را اجرا کنید و نتیجه را مشاهده کنید.

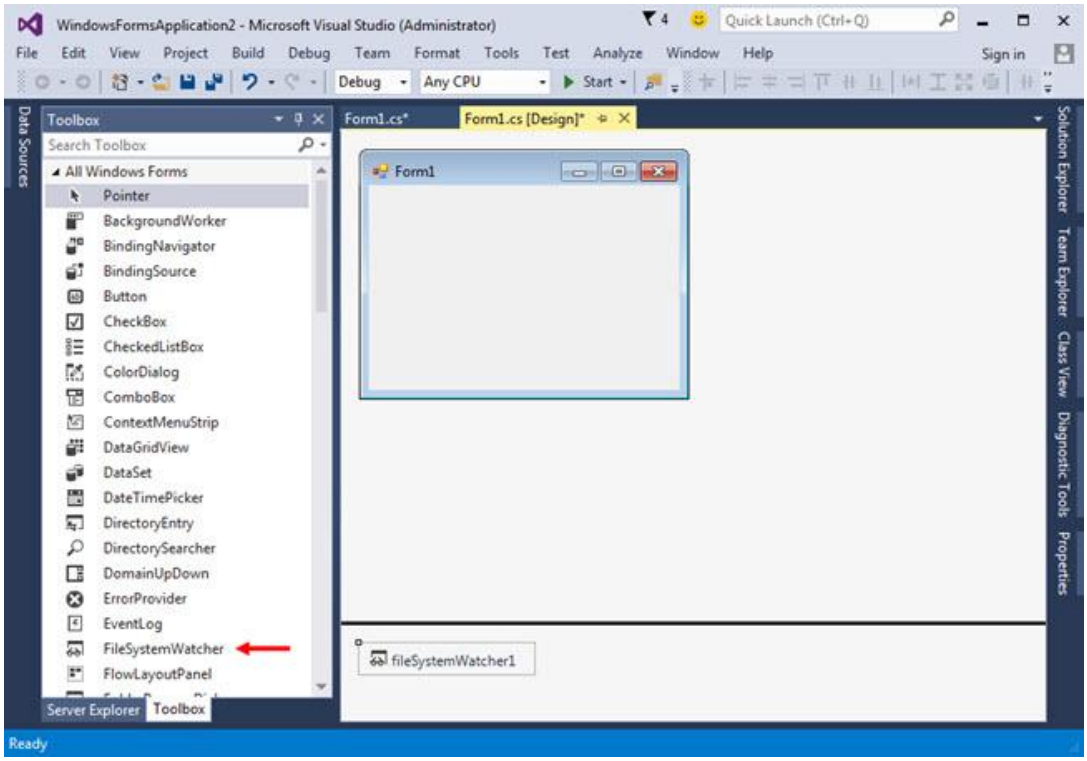
به یک مثال ساده دیگر در مورد عملکرد این کنترل توجه کنید. یک پروژه Windows Application جدید ایجاد کنید و یک کنترل RichTextBox و یک کنترل Timer بر روی فرم قرار دهید. خاصیت Interval کنترل Timer را برابر ۱۰۰۰ و خاصیت Enabled آن را برابر True تنظیم کنید. روی کنترل Timer دابل کلیک کنید و در داخل رویداد Tick آن کدهای زیر را بنویسید:

```
private void timer1_Tick(object sender, EventArgs e)
{
    richTextBox1.Text += "Hello World!\n";
}
```

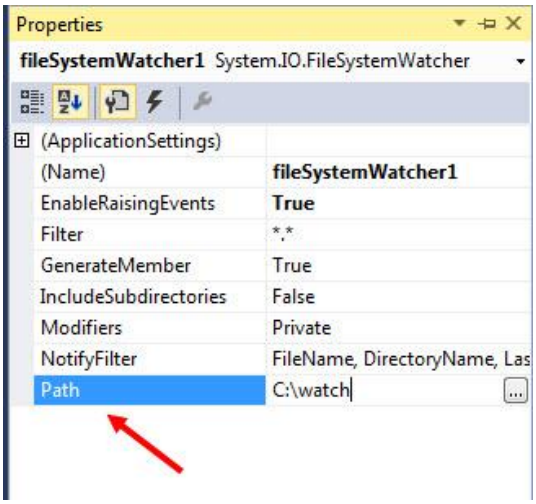
با تنظیم خاصیت Interval بر روی ۱۰۰۰ هر یک ثانیه یک بار متن Hello World! در داخل کنترل RichTextBox چاپ می‌شود. برنامه را اجرا و نتیجه را مشاهده کنید.

کنترل FileSystemWatcher

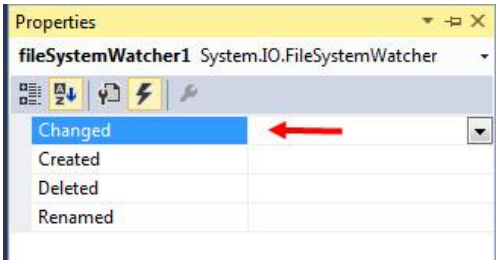
از این کنترل برای مانیتور کردن یک پوشه استفاده می‌شود. بدین معنی که هر گونه تغییر اعم از حذف، تغییر نام، ایجاد یک فایل جدید و .... در پوشه مورد نظر را به ما گزارش می‌دهد. برای روشن شدن مطلب یک برنامه ویندوزی جدید ایجاد کنید و یک کنترل FileSystemWatcher بر روی آن بکشید (کنترل به قسمت component try می‌رود).



با استفاده از پنجره Propertis و خاصیت Path پوشه‌ای را که قرار است تغییرات آن مانیتور شود مشخص می‌کنیم که در این مثال پوشه‌ای به نام watch است که در درایو C قرار دارد:



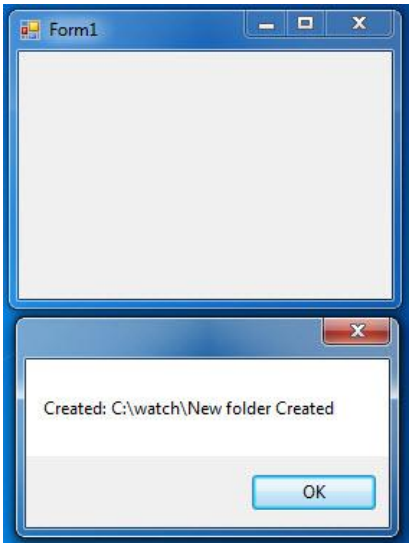
حال نوبت به کدنویسی می‌رسد. این کنترل دارای چهار رویداد می‌باشد که هر کدام مسئول چک کردن تغییرات در پوشه مورد نظر می‌باشند:



به ترتیب کدهای زیر را در رویدادهای مربوطه بنویسید :

```
private void fileSystemWatcher1_Changed(object sender, System.IO.FileSystemEventArgs e)
{
       MessageBox.Show(string.Format("Changed: {0} {1}", e.FullPath, e.ChangeType));
}

private void fileSystemWatcher1_Created(object sender, System.IO.FileSystemEventArgs e)
{
       MessageBox.Show(string.Format("Created: {0} {1}", e.FullPath, e.ChangeType));
}

private void fileSystemWatcher1_Deleted(object sender, System.IO.FileSystemEventArgs e)
{
       MessageBox.Show(string.Format("Deleted: {0} {1}", e.FullPath, e.ChangeType));
}

private void fileSystemWatcher1_Renamed(object sender, System.IO.RenamedEventArgs e)
{
       MessageBox.Show(string.Format("Renamed: {0} {1}", e.FullPath, e.ChangeType));
}
```

`e.FullPath` مسیر فایل و `e.ChangeType` نوع رویدادی که به وقوع می‌پیوندد را، به ما نشان می‌دهد .حال برنامه را اجرا کرده و مثلاً یک پوشه در داخل پوشه مورد نظر ایجاد کنید. به محض ایجاد پوشه، پیغامی مبنی بر ایجاد یک پوشه جدید (یا تغییر در پوشه مورد نظر) به شما نمایش داده می‌شود:



کنترل WebBrowser

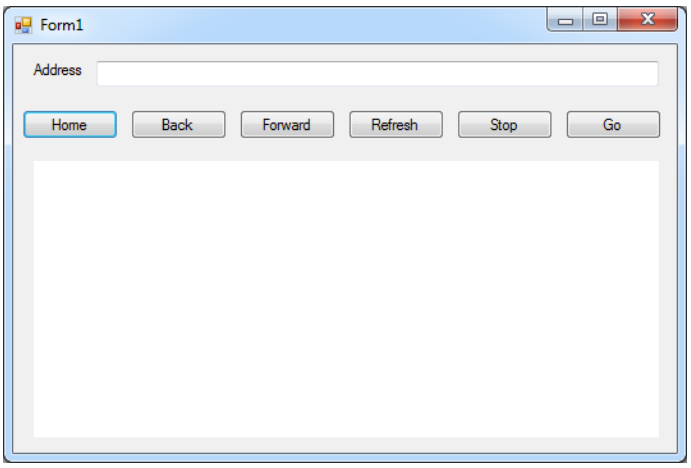
از کنترل WebBrowser برای نمایش محتویات صفحات وب، فایل‌های XML و متنی استفاده می‌شود. این کنترل به طور خودکار، لینک‌هایی را که کاربر بر روی آنها کلیک می‌کند، دنبال کرده و با استفاده از آن می‌توان تمام امکانات مربوط به یک مرورگر وب را پیاده سازی کرد. در اصل این کنترل یک نسخه از مرورگر Internet Explorer ویندوز را در داخل فرم ما نمایش می‌دهد. در نتیجه تمام صفحاتی که ما در برنامه با آنها سر و کار داریم در تاریخچه این مرورگر ذخیره می‌شوند. کنترل WebBrowser دارای خواص و متدهای مختلفی برای پیمایش اسناد مختلف می‌باشد، که در زیر به برخی از آنها اشاره شده است:

خاصیت	توضیح
AllowNavigation	مشخص می‌کند که آیا کنترل WebBrowser می‌تواند وقتی که یک صفحه بارگذاری شد، صفحات دیگر را باز کند یا نه؟
AllowWebBrowserDrop	مشخص می‌کند که آیا کنترل WebBrowser می‌تواند سندی را که به داخل آن Drag شده است را پیمایش کند یا نه؟
CanGoBack	مشخص می‌کند که آیا صفحات دیگر در لیست تاریخچه وجود دارند و کنترل می‌تواند از صفحه فعلی یک صفحه رو به عقب حرکت کند. یعنی صفحه قبل را باز کند.
CanGoForward	مشخص می‌کند که آیا صفحات دیگر در لیست تاریخچه وجود دارند و کنترل می‌تواند از صفحه فعلی یک صفحه رو به جلو حرکت کند. یعنی صفحه بعد را باز کند.
Document	یک HtmlDocument که نماینده صفحه وب جاری است را بر می‌گرداند.
DocumentText	محتویات HTML صفحه نمایش داده شده در کنترل را بر می‌گرداند.
DocumentTitle	عنوان سند جاری که در کنترل در حال نمایش است را بر می‌گرداند.
DocumentType	نوع سند جاری که در کنترل در حال نمایش است را بر می‌گرداند.
IsOffline	با برگرداندن یک مقدار مشخص می‌کند که آیا کنترل WebBrowser در حالت آفلاین است یا خیر؟
ReadyState	با برگرداندن یک مقدار وضعیت جاری کنترل WebBrowser را مشخص می‌کند.
ScrollBarsEnabled	با تعیین یا برگرداندن یک مقدار نشان می‌دهد که آیا اسکرول بار در کنترل نمایش داده شده شود یا نه؟
Url	برای تعیین یا برگرداندن آدرس سند جاری مورد استفاده قرار می‌گیرد.
Version	نسخه Internet Explorer نصب شده را بر می‌گرداند.

متد	توضیح
GoBack()	در تاریخچه صفحات جستجو شده، به شما اجازه می‌دهد که به صفحه قبل بر گردید.
GoForward()	در تاریخچه صفحات جستجو شده، به شما اجازه می‌دهد که به صفحه جلو بروید.
GoHome()	کاربر را به صفحه اصلی می‌برد.
Navigate()	یک سند جدید را بارگذاری می‌کند.
Stop()	تمام مراحل پیمایش و بارگذاری صفحات پویا را لغو می‌کند.

رویداد	توضیح
DocumentCompleted	زمانی رخ می‌دهد که بارگذاری یک سند در WebBrowser به پایان رسیده باشد.
DocumentTitleChanged	زمانی رخ می‌دهد که مقدار خاصیت DocumentTitle تغییر کند.
FileDownload	زمانی رخ می‌دهد که کنترل WebBrowser یک فایل را دانلود کند.
Navigated	زمانی رخ می‌دهد که کنترل WebBrowser یک سند جدید را پیمایش و بارگذاری آن را آغاز کرده باشد.
Navigating	قبل از پیمایش یک سند جدید توسط کنترل WebBrowser رخ می‌دهد.
NewWindow	قبل از باز شدن یک پنجره جدید از مرورگر رخ می‌دهد.

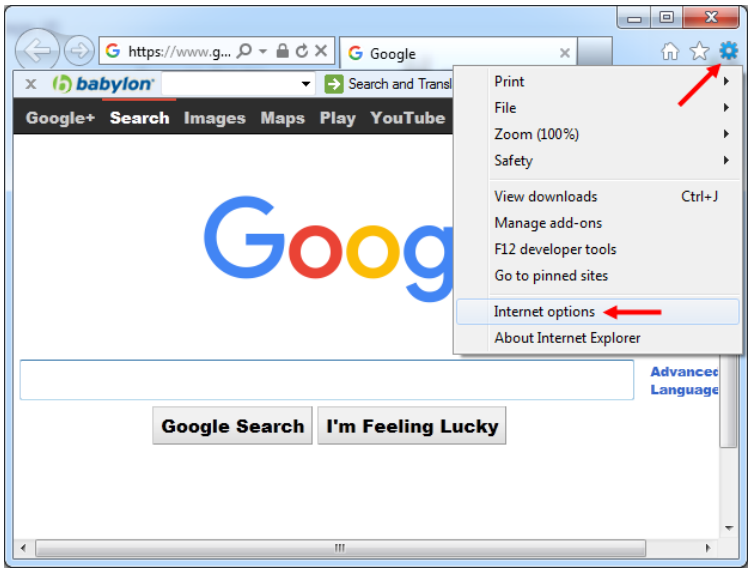
برای آشنایی با کاربرد موارد ذکر شده در جدول بالا یک برنامه ویندوزی به صورت زیر ایجاد کرده و نام آن را WebBrowserControl بگذارید :

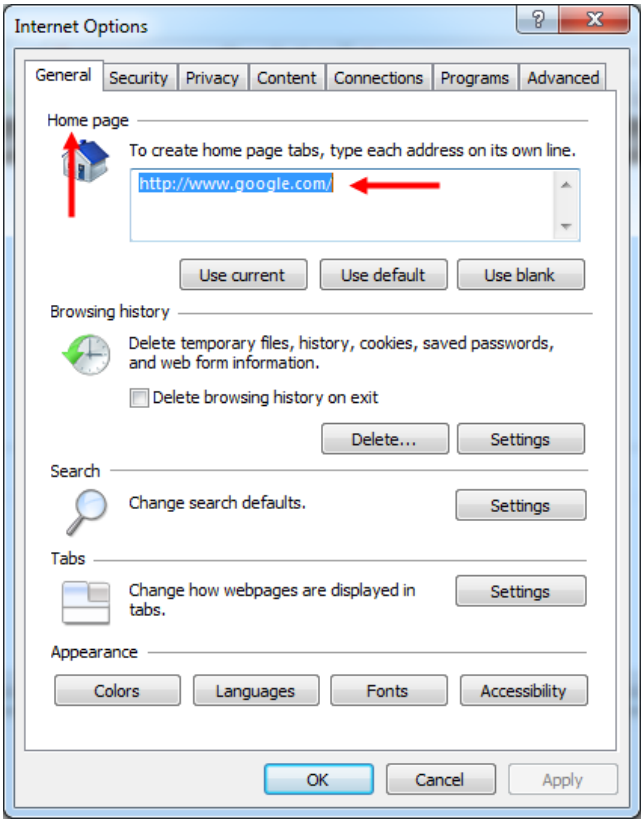


بر روی دکمه Home دو بار کلیک کرده و کد زیر را در داخل آن بنویسید :

```
private void button1_Click(object sender, EventArgs e)
{
    webBrowser1.GoHome();
}
```

متد GoHome() در کد بالا، به صفحه ای مراجعه می‌کند در مرورگر Internet Explorer ما به عنوان صفحه اصلی معرفی شده است:





یعنی هر بار که بر روی دکمه Home کلیک کنید صفحه گوگل باز می‌شود. حال بر روی دکمه Back دوبار کلیک کرده و کدهای زیر را بنویسید :

```
private void button2_Click(object sender, EventArgs e)
{
    if (webBrowser1.CanGoBack)
    {
        webBrowser1.GoBack();
    }
}
```

در کد بالا ابتدا در داخل شرط حلقه و با استفاده از خاصیت CanGoBack چک می‌کنیم که آیا صفحه‌ی قبلی هم وجود دارد. اگر وجود داشت، مرورگر را با استفاده از متد ()GoBack به آن انتقال می‌دهیم. برنامه را اجرا کرده و بر روی دکمه Home کلیک کنید. در داخل صفحه گوگل یک کلمه بنویسید و سپس روی دکمه Enter کلیک کنید. بعد از باز شدن صفحه جدید، اگر بر روی دکمهBack کلیک کنید. مشاهده می‌کنید که مرورگر به صفحه قبل که همان صفحه اصلی گوگل است بر می‌گردد. برای دکمهForWard هم کد زیر را می‌نویسیم:

```
private void button3_Click(object sender, EventArgs e)
{
    if (webBrowser1.CanGoForward)
    {
        webBrowser1.GoForward();
    }
}
```

در کد بالا با استفاده از خاصیت CanGoForward چک می‌کنیم که آیا صفحه‌ی بعدی هم وجود دارد. اگر وجود داشت، مرورگر را با استفاده از متد ()GoForward به آن انتقال می‌دهیم. برای این کار ابتدا باید یک صفه جدید باز کنید و یک بار با دکمه Back رو به عقب بر گردید و سپس از دکمه ForWard استفاده کرده و تا عملکرد آن را مشاهده کنید. برای کدنویسی دکمه Refresh هم به صورت زیر عمل می‌کنید :

```
private void button4_Click(object sender, EventArgs e)
{
    webBrowser1.Refresh();
}
```

همانطور که در کد بالا مشاهده می‌کنید، از متد ()Refresh استفاده کرده‌ایم. این متد زمانی کارایی دارد که یک صفحه ناقص بارگذاری شده است و می‌خواهیم آن را دوباره نمایش دهیم. برای جلوگیری از بارگذاری یک صفحه هم از دکمه Stop استفاده می‌کنیم. برای کدنویی این دکمه دو بار بر روی آن کلیک کرده و متد ()Stop را فراخوانی کنید :

```
private void button5_Click(object sender, EventArgs e)
{
    webBrowser1.Stop();
}
```

و در آخر نوبت به کدنویسی دکمه Go می‌رسد. می‌خواهیم وقتی بر روی این دکمه کلیک شد، کنترل webBrowser سایتی که آدرس آن را در داخل TextBox نوشته‌ایم، را نمایش دهد. برای این منظور بر روی دکه Go دو بار کلیک کرده و کدهای زیر را در داخل آن بنویسید :

```
private void button6_Click(object sender, EventArgs e)
{
    webBrowser1.Navigate(textBox1.Text.Trim());
}
```

در کد بالا، ما متد Navigate() این کنترل را فراخوانی کرده‌ایم. این متد یک رشته دریافت می‌کند که در اصل آدرس همان وب سایتی است که می‌خواهیم توسط webBrowser نمایش داده شود. آدرس یک وب سایت را در داخل TextBox نوشته و با کلیک بر روی دکمه Go نتیجه را مشاهده کنید. بر روی رویداد دوبار کلیک کرده و کدهای زیر را بنویسید:

```
private void webBrowser1_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
{
    this.Text = webBrowser1.DocumentTitle;
}
```

در کد بالا گفته‌ایم که وقتی صفحه به طور کامل بارگذاری شد، عنوان فرم را به عنوان صفحه ای که باز شده است تغییر بده. این کار را با استفاده از خاصیت DocumentTitle انجام داده‌ایم. برنامه را اجرا کرده و با کلیک بر روی دکمه Home نتیجه را مشاهده کنید:

حال به جای DocumentTitle، خاصیت DocumentType را بنویسید. این خاصیت نوع سندی که در داخل کنترل در حال نمایش است را بر می‌گرداند. با ماوس یک عکس به داخل کنترل بکشید و رها کنید:

بر روی فرم دوبار کلیک کرده و کد زیر را در داخل رویداد Load آن بنویسید :

```
private void Form1_Load(object sender, EventArgs e)
{
    webBrowser1.AllowWebBrowserDrop = false;
}
```

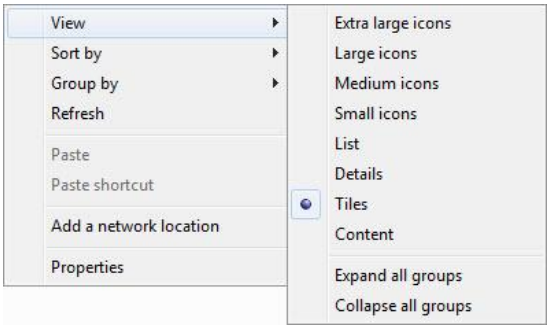
در کد بالا مقدار خاصیت AllowWebBrowserDrop را برابر false قرار داده‌ایم. حال اگر برنامه را اجرا کنید و بخواهید مثلاً یک عکس را از محیط ویندوز با ماوس بگیرید و در داخل کنترل webBrowser رها کنید به شما اجازه این کار داده نمی‌شود. حال خط بالا را پاک کرده و به جای آن کد زیر را در رویداد Load فرم بنویسید :

```
private void Form1_Load(object sender, EventArgs e)
{
    webBrowser1.AllowNavigation = false;
}
```

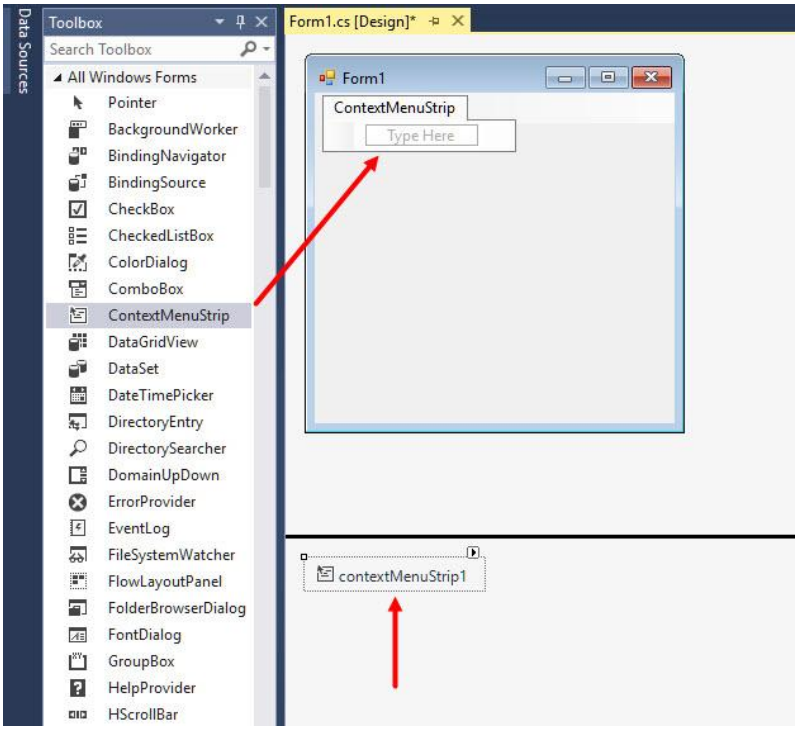
اگر مقدار خاصیت AllowNavigation برابر false باشد دکمه‌های Back و ForWard از کار می افتند. و به نوعی می‌توان گفت که بعد از باز شدن صفحه اول، اجازه باز شدن صفحات دیگر به شما داده نمی‌شود.

کنترل ContextMenuStrip

کنترل ContextMenuStrip یک پنجره Pop up می‌باشد که وقتی ظاهر می‌شود که، کاربر بر روی فرم و یا یک کنترل، راست کلیک کند. نمونه بارز این کنترل را می‌توانید با کلیک راست بر روی دسکتاپ ویندوز و یا قسمت‌های مختلف ویندوز، مشاهده کنید :



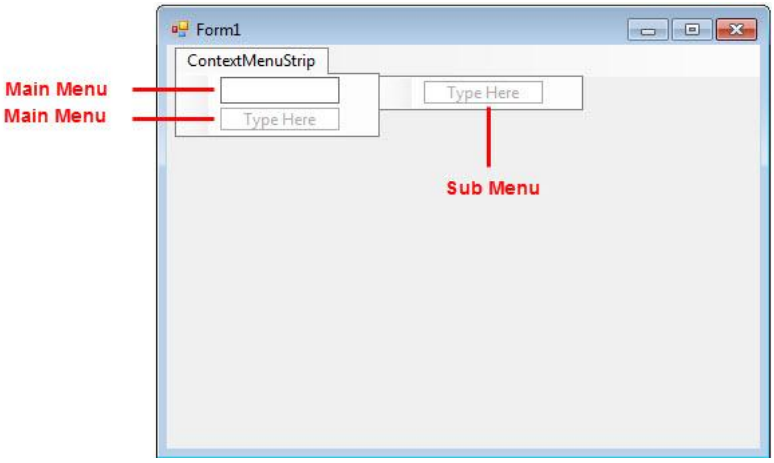
برای استفاده از این کنترل در فرم و سایر کنترل‌ها کافیست که یک نمونه از آن را از جعبه ToolBox کشیده و بر روی فرم قرار دهید :



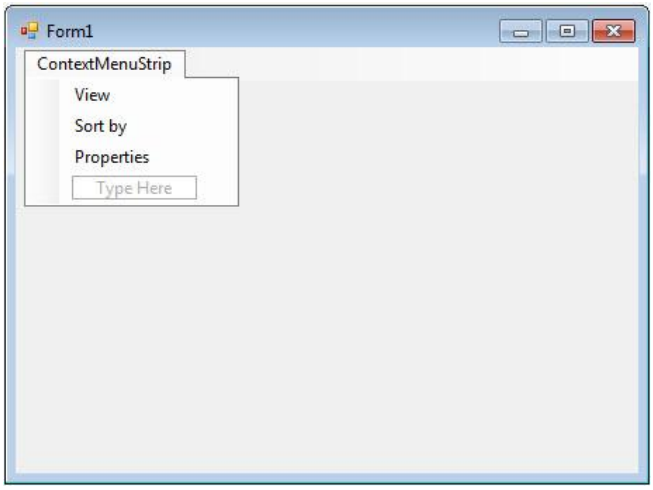
با این کار همانطور که در شکل زیر مشاهده می‌کنید، کنترل به قسمت بالای فرم چسبیده و همچنین می‌توانید نام آن را در قسمت Component Tryمشاهده کنید. اضافه شدن کنترل به قسمت Component Try بدین معناست که کنترل جزء کنترل‌های غیر بصری است. یعنی در حالت عادی و با اجرای برنامه قابل مشاهده نبوده و در شرایط خاصی ظاهر می‌شود، مثلاً کنترل ContextMenuStrip با کلیک راست بر روی فرم و یا کنترل نمایش داده می‌شود.

اضافه کردن آیتم به ContextMenuStrip

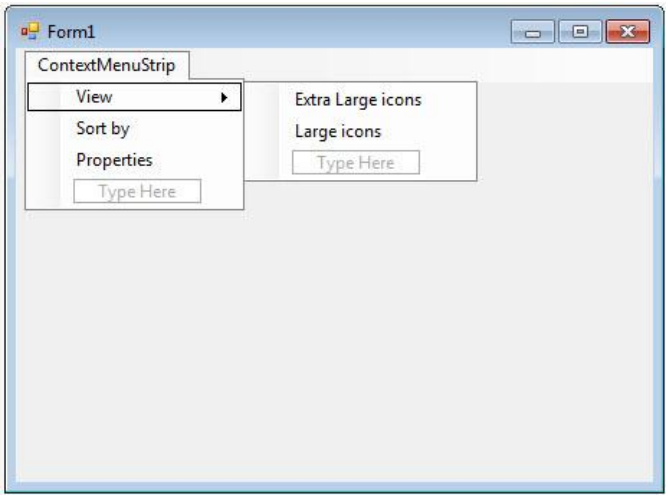
برای اضافه کردن آیتم به ContextMenuStrip بر روی جعبه‌های خالی آن کلیک کرده و آیتم‌های مورد نظر خود را اضافه می‌کنیم. با کلیک بر روی هر جعبه خالی یک جعبه خالی در سمت راست و یک جعبه خالی در پایین آن ظاهر می‌شود. برای اضافه کردن آیتم‌های اصلی، از جعبه زیرین و برای زیر آیتم هم، از جعبه سمت راستی استفاده می‌کنیم:



حال می‌خواهیم منوی ابتدای درس را شبیه سازی کنیم. برای این کار ابتدا منوهای اصلی را ایجاد می‌کنیم:



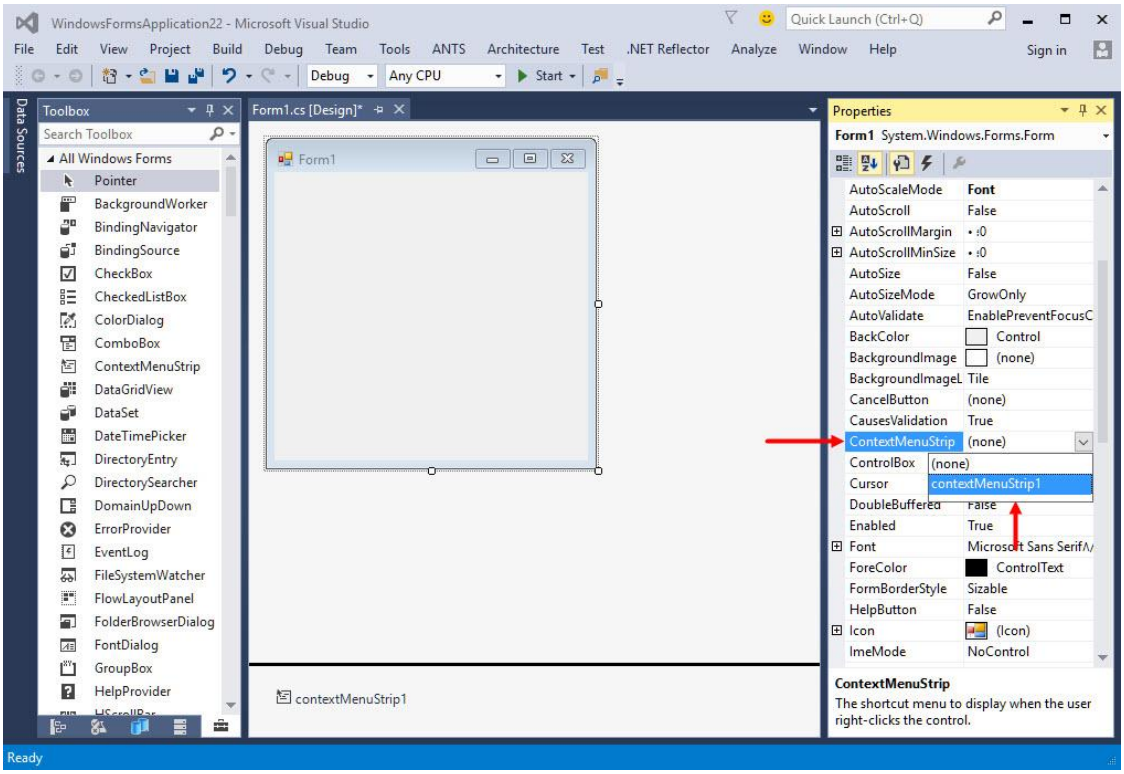
فرض کنید می‌خواهیم زیر منوهایی را هم به منوی `View` اضافه کنیم. بر روی منوی `View` کلیک کرده و زیر منوهای آن را هم اضافه می‌کنیم:



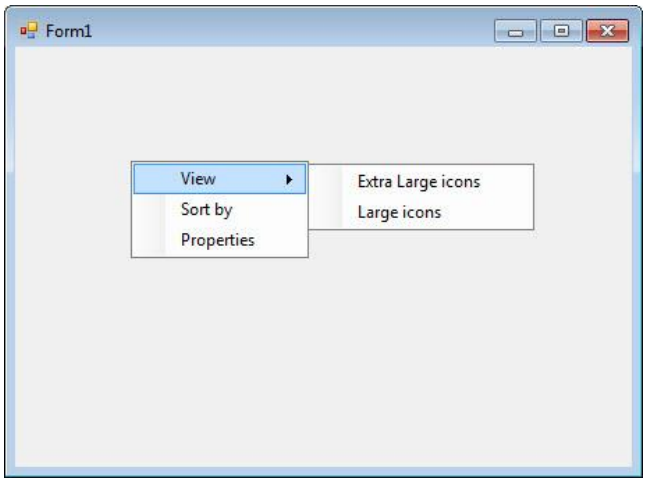
تا اینجا `ContextMenuStrip` تکمیل شده است و می‌خواهیم آن را به کنترل مورد نظرمان اضافه کنیم.

اضافه کردن `ContextMenuStrip` به کنترل‌ها

با اضافه کردن `ContextMenuStrip` به فرم اتفاق خاصی نمی‌افتد و وقتی برنامه اجرا شود و بر روی فرم کلیک راست کنیم این کنترل ظاهر نمی‌شود. حال که کنترل را به فرم اضافه کردیم، باید به برنامه اعلام کنیم که مثلاً فرم قرار است که از فلان `ContextMenuStrip` استفاده کند. برای این کار بر روی فرم کلیک کرده تا به حالت فعال در آید، سپس از پنجره `Properties` گزینه `ContextMenuStrip` را بر روی `ContextMenuStrip` مورد نظر خودمان تنظیم می‌کنیم:



برنامه را اجرا و بر روی فرم راست کلیک کنید :

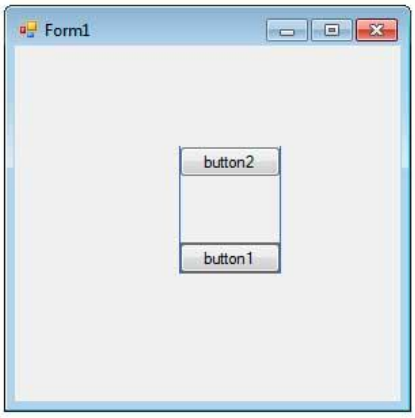


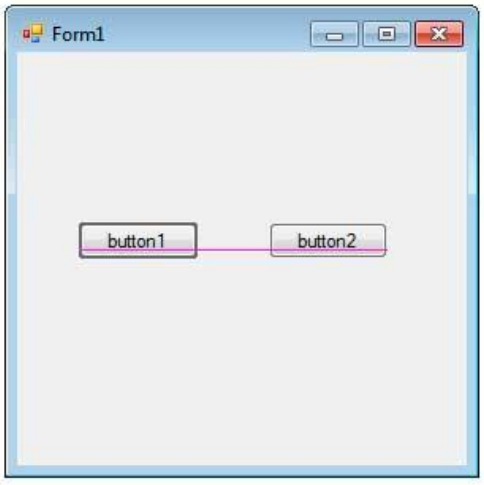
طراحی فرم‌های ویندوزی

هنگام طراحی یک رابط گرافیکی، باید به فکر مکان، اندازه، تراز بندی و رنگ کنترل‌ها یا اجزا گرافیکی باشید. خوشبختانه، ویژوال استودیو دارای ابزارهایی است که به شما در طراحی رابط گرافیکی کمک می‌کنند. برای توضیح این ابزارها یک پنجره ویندوزی ایجاد کنید. هنگامی که یک کنترل

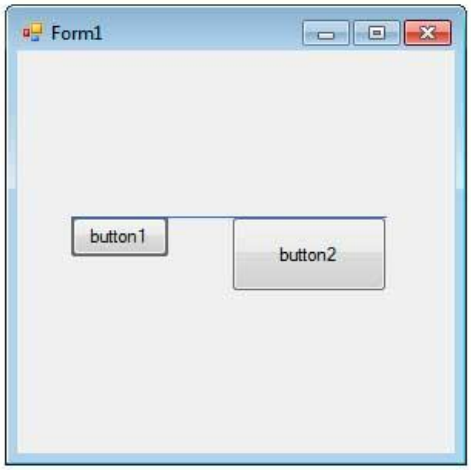
را بر روی فرم می‌کشید یا قرار می‌دهید، متوجه ظاهر شدن خطوطی می‌شوید. این خطوط به شما اجازه می‌دهند که موقعیت کنترل را نسبت به کنترل‌های دیگر یا خود فرم تراز بندی کنید.

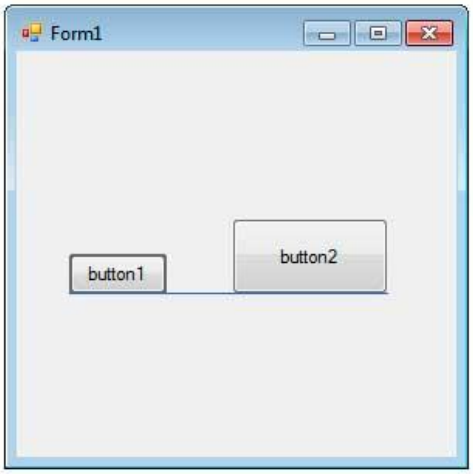
می‌توان یک کنترل را در گوشه‌های مختلف فرم قرار داد. اگر کنترل‌های دیگری نیز در فرم قرار داشته باشند، کنترلی را که شما بر روی فرم می‌کشید می‌تواند به طور خودکار خودش را نسبت به کنترل‌های دیگر تراز کند.

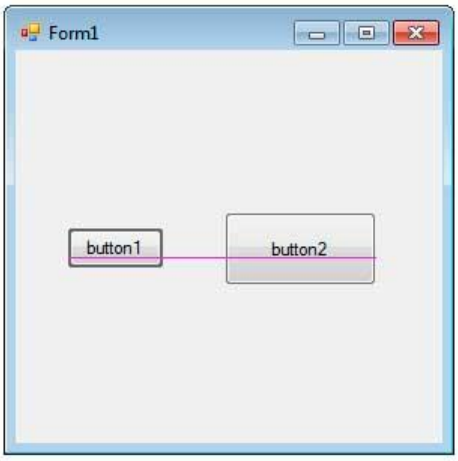


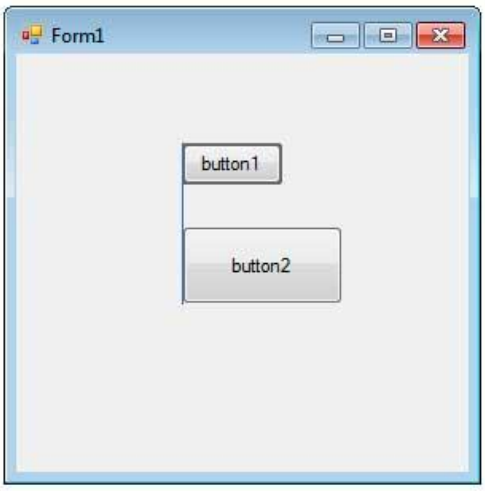


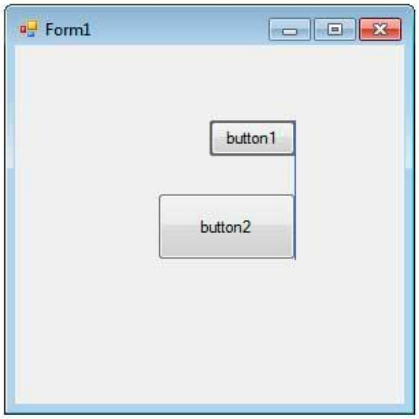
می‌توان یک کنترل را نسبت به قسمت بالا، وسط، پایین، چپ یا راست یک کنترل بزرگ‌تر، تراز کرد.


Form1
button1
button2

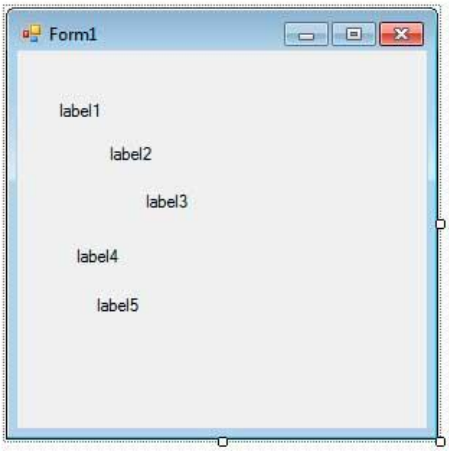

Form1
button1
button2


Form1
button1
button2

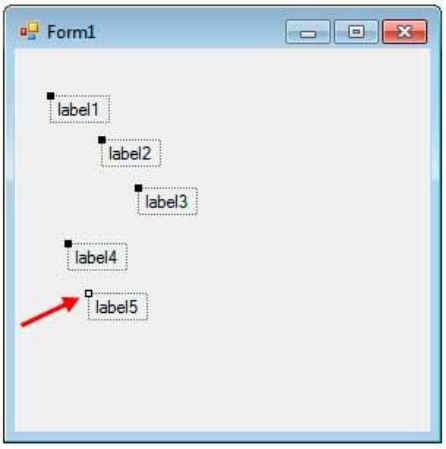




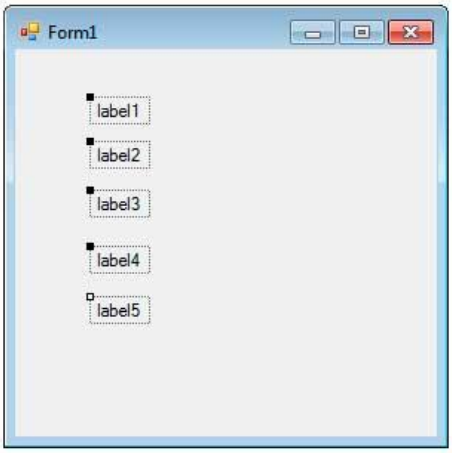
ویژوال استودیو همچنین دارای دستوراتی برای تراز بندی خودکار و یکنواخت کردن فضاها و اندازه کنترل‌ها دارد. به فرم زیر که دارای چندین کنترل است توجه کنید:



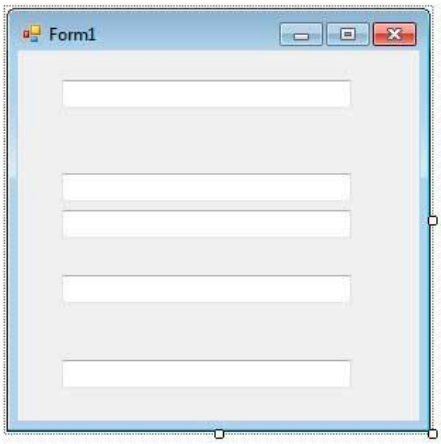
می‌توانید هر کنترلی را که می‌خواهید به وسیله پایین نگه داشتن دکمه Shift و کلیک بر روی کنترل تراز بندی کنید. همچنین می‌توان کنترل‌ها را با کشیدن ماوس به دور آنها انتخاب کرد.



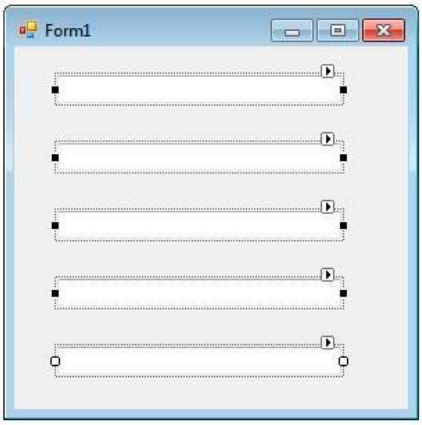
وقتی که کنترل‌ها در حالت انتخاب هستند، کنترلی که در گوشه بالا و سمت چپ خود دارای یک مربع کوچک سفید است، کنترلی است که بقیه کنترل‌ها نسبت به آن تراز بندی می‌شوند. برای تراز بندی کنترل‌ها به مسیر `Format > Align` رفته و سپس گوشه‌ای را که می‌خواهید تراز بندی بر اساس آن انجام شود، انتخاب کنید. برای روشن شدن مطلب، سمت چپ را انتخاب کنید. کنترل‌های دیگر همگی بر اساس گوشه سمت چپ کنترل اصلی تراز می‌شوند.



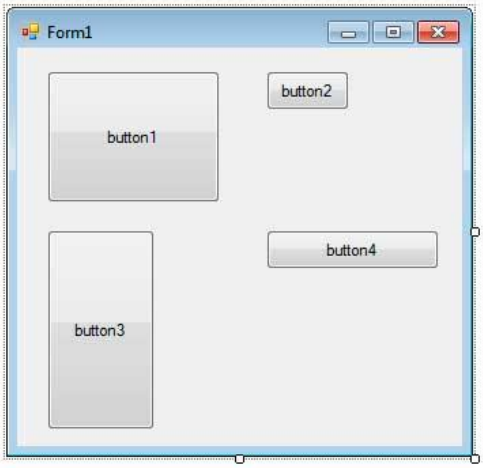
می‌توان یک کنترل را به صورت افقی یا عمودی در وسط یک فرم قرار داد. فقط کافیست به مسیر `Format > Center Form` رفته و جهتی را که می‌خواهید کنترل در وسط آن قرار بگیرد، انتخاب کنید. از دستور دیگری برای مساوی کردن فضاها و فاصله‌های بین کنترل‌ها می‌توان استفاده کرد. به فرم زیر توجه کنید:



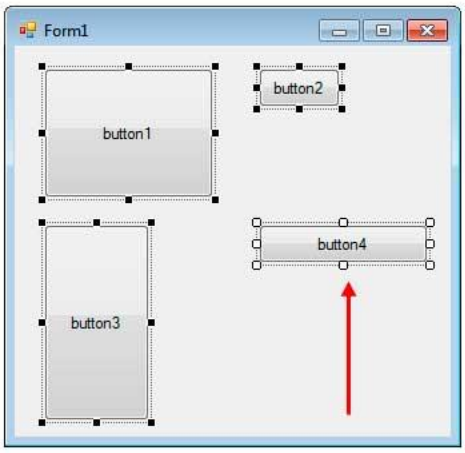
فرم بالا، شامل کنترل‌هایی است که فاصله بین آنها با هم برابر نیست. برای مساوی کردن این فاصله‌ها، همه کنترل‌ها را انتخاب کنید و به منوی Format رفته و گزینه Vertical Spacing را برای تنظیم فاصله‌های پایین و بالای کنترل‌ها و یا Vertical Spacing را برای تنظیم فاصله‌های سمت چپ و راست کنترل‌ها انتخاب کنید. برای فرم مثالا بالا به مسیر Format > VerticalSpacing رفته و بر روی گزینه Make Equal کلیک کنید.



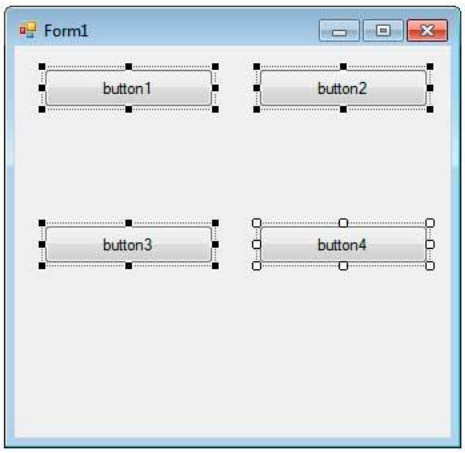
مشاهده می‌کنید که فضای عمودی بین کنترل‌ها تراز می‌شود. می‌توانید چندین کنترل را انتخاب و اندازه آنها را با هم برابر کنید. به فرم زیر توجه کنید:



کنترلی که دارای یک دستگیره سفید است، کنترل مرجع می‌باشد و اندازه دیگر کنترل‌ها بر اساس اندازه آن تغییر می‌کند.



به مسیر `Format > Make Same Size` و سپس گزینه‌های `widths`، `height` یا `both` را انتخاب کرده تا اندازه کنترل‌ها از نظر عرض، ارتفاع و یا هر دو یکسان شود.



خاصیت Anchor

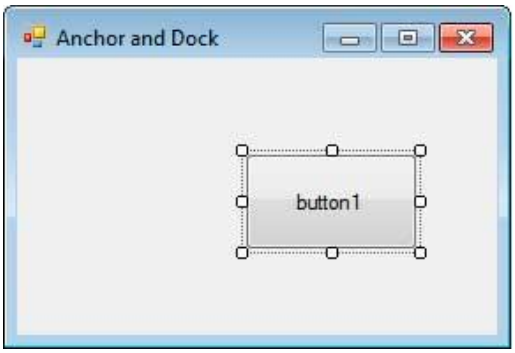
یکی از مشکلات قرار دادن کنترل‌ها بر روی کنترل فرم این است که، مکان آنها هنگام تغییر سایز فرم، تغییر نمی‌کند و در جای قبلی می‌مانند. به مثال زیر توجه کنید. یک فرم ساده همراه با چند کنترل در داخل آن ایجاد می‌کنیم. وقتی برنامه را اجرا می‌کنیم، هیچ چیز اشتباهی به چشم نمی‌خورد.

اما وقتی فرم را به وسیله کشیدن گوشه‌های آن و یا با زدن دکمه بیشینه، بزرگ می‌کنیم، مشکل معلوم می‌شود.

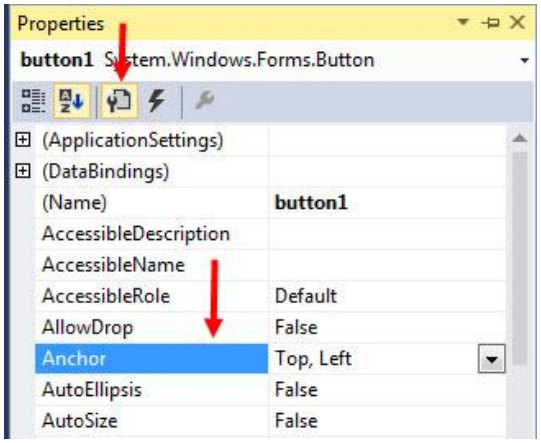
به عکس بالا که اندازه آن به اندازه صفحه دسکتاپ بزرگ کرده‌ایم توجه کنید. همانطور که مشاهده می‌کنید مکان کنترل‌ها با افزایش اندازه فرم تغییر نمی‌کند. برای برطرف کردن این مشکل می‌توان از خاصیت Anchor که در بیشتر کنترل‌ها در دسترس است، استفاده نمود. این خاصیت رفتار کنترل‌ها را در هنگام تغییر سایز فرم تعیین می‌کند. می‌توانیم مشخص کنیم که کنترل به کدام گوشه فرم بچسبد. همچنین تعیین کنیم که کنترل‌ها چگونه تغییر سایز دهند. اجازه بدهید که خاصیت Anchor را به کنترل‌هایمان اختصاص دهیم تا رفتار طبیعی تری هنگام تغییر سایز فرم از خود نشان دهند. خاصیت Anchor یک مقدار از نوع شمارشی System.Windows.Forms.AnchorStyles قبول می‌کند.

AnchorStyle	توضیح
Bottom	کنترل به قسمت پایینی مکان یا کنترلی که در آن قراردارد، می‌چسبد.
Left	کنترل به قسمت چپ مکان یا کنترلی که در آن قراردارد، می‌چسبد.
Right	کنترل به قسمت راست مکان یا کنترلی که در آن قراردارد، می‌چسبد..
Top	کنترل به بالای مکان یا کنترلی که در آن قراردارد، می‌چسبد.
None	کنترل به هیچ قسمتی از مکان یا کنترلی که در آن قراردارد، نمی‌چسبد.

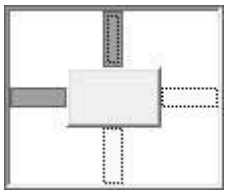
این خاصیت در همه کنترل‌ها در حالت پیش‌فرض ترکیبی از مقادیر Top و Left است. وقتی مقادیر AnchorStyles را به خاصیت Anchor تخصیص می‌دهیم، فاصله کنترل از گوشه خاصی از فرم با وجود تغییر اندازه فرم حفظ می‌شود. به عنوان مثال، اجازه بدهید که کنترل دکمه زیر را به سمت راست فرم بچسبانیم.



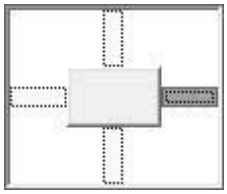
ویژوال استودیو و ویژوال سی‌شارپ دارای ابزاری برای تعیین AnchorStyles می‌باشند. کنترلی را که می‌خواهید به یکی از گوشه‌ها بچسبانید، انتخاب کنید. به پنجره Properties رفته و خاصیت Anchor را پیدا کنید.



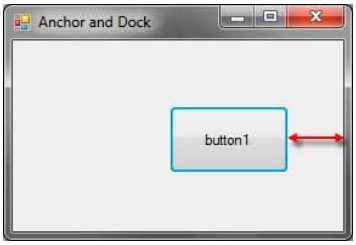
بر روی دکمه ای که به شکل یک فلش رو به پایین است (شکل بالا) کلیک کنید تا شکل زیر ظاهر شود:

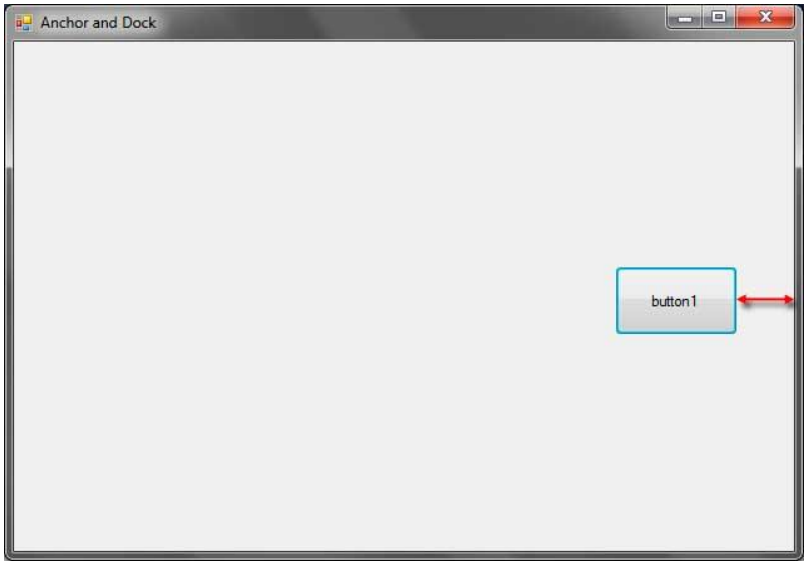


مربعی که در وسط شکل زیر قرار دارد نماینده کنترل است. مربعی که به رنگ خاکستری است گوشه‌ای را نشان می‌دهد که قرار است کنترل به آن بچسبد. شما به راحتی و با کلیک بر روی این مربع‌ها می‌توانید مقادیر AnchorStyle مربوط به کنترل را اضافه یا حذف کنید. حال برای چسباندن دکمه مثال بالا به سمت چپ فرم بر روی مربع سمت راست کلیک می‌کنیم تا رنگ آن مانند شکل زیر خاکستری شود.

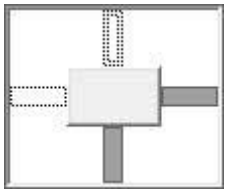


برنامه را اجرا کرده و اندازه فرم را تغییر می‌دهیم. اندازه فاصله دکمه از گوشه راست فرم را در حالت عادی و در حالتی که فرم را تغییر اندازه می‌دهیم، مقایسه می‌کنیم.

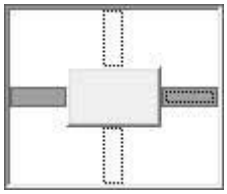




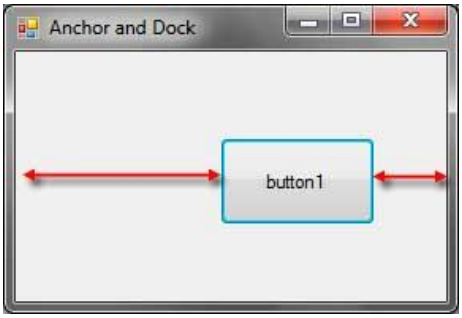
همانطور که مشاهده می‌کنید، این فاصله قبل و بعد از تغییر اندازه فرم یکسان است. حال اجازه بدهید که مقدار Bottom را هم اضافه کنیم.

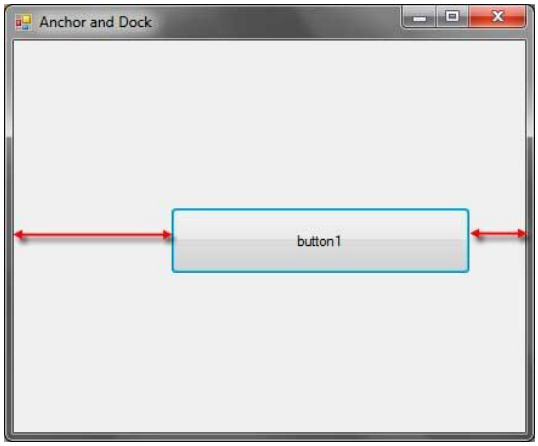


با ترکیب دو مقدار Right و Bottom، کنترل همیشه در فاصله یکسانی از گوشه سمت راست و چپ کنترل فرم قرار می‌گیرد. وقت آن رسیده که مقادیری را که مخالف همدیگر هستند، را اضافه کنیم و مشاهده کنیم که چه اتفاقی برای کنترل می‌افتد. به عنوان مثال مقدار Left مخالف مقدار Right است. انتضار ما این است که کنترل با فاصله یکسانی از سمت چپ و راست فرم قرار بگیرد.



برنامه را اجرا کرده و فرم را تغییر اندازه دهید.





اندازه دکمه در دو جهت مخالف هم تغییر می‌کند. اگر خاصیت Anchor کنترل را در چهار جهت مشخص کنیم اندازه دکمه به صورت افقی و عمودی در همه جهات تغییر می‌کند. اما وقتی مقدار خاصیت Anchor را برابر AnchorStyles.None قرار دهیم، کنترل رفتار متفاوتی از خود نشان می‌دهد. در این حالت وقتی کنترل فرم را تغییر اندازه دهیم، کنترل به اندازه نصف تغییر اندازه فرم و در جهت تغییر اندازه آن حرکت می‌کند.

به عنوان مثال، اگر فرم را به اندازه ۱۰۰ پیکسل و در جهت راست تغییر اندازه دهیم، کنترل به اندازه ۵۰ پیکسل به سمت راست حرکت می‌کند و اگر فرم را به اندازه ۱۰۰ پیکسل در جهت راست و ۵۰ پیکسل در جهت پایین تغییر اندازه دهیم، کنترل ۵۰ پیکسل در جهت راست و ۲۵ پیکسل در جهت پایین حرکت می‌کند.

خاصیت Dock

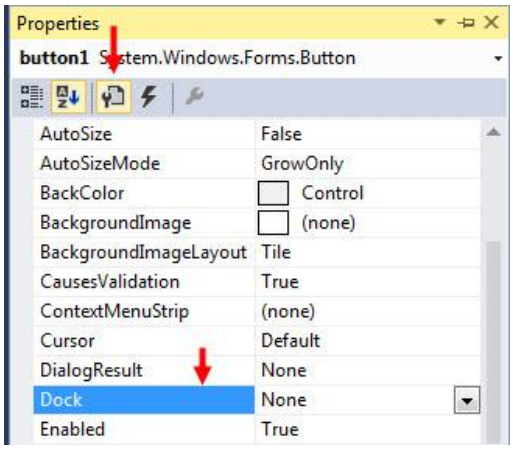
خاصیت Dock به شما اجازه می‌دهد که کنترل مورد نظرتان را به هر گوشه‌ای از فرم یا کنترل در بر گیرنده آن بچسبانید. با استفاده از این روش کنترل‌ها در هنگام تغییر اندازه فرم دست نخورده باقی می‌مانند (یعنی ظاهر کلی فرم به هم نمی‌خورد و کنترل‌ها با تغییر اندازه فرم جا به جا

می‌شوند). این خاصیت یک مقدار شمارشی از نوع System.Windows.Forms.DockStyle قبول می‌کند. در جدول زیر مقادیر موجود این نوع شمارشی ذکر شده‌اند:

مقدار	توضیح
Bottom	کنترل، به قسمت پایین کنترلی که در آن قرار دارد می‌چسبد.
Fill	کنترل، تمام کنترلی را که در آن قرار دارد را پر می‌کند.
Left	کنترل، به قسمت چپ کنترلی که در آن قرار دارد می‌چسبد.
Right	کنترل، به قسمت راست کنترلی که در آن قرار دارد می‌چسبد.
Top	کنترل، به قسمت بالای کنترلی که در آن قرار دارد می‌چسبد.

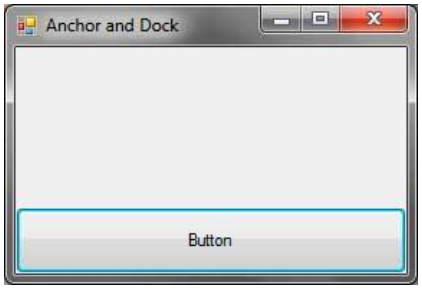
حال اجازه دهید که یک کنترل button را به گوشه‌های مختلف فرم بچسبانیم.

برای این کار، کنترل را انتخاب کرده و به پنجره Properties می‌رویم.



خاصیت Dock را یافته و بر روی دکمه بازشوند آن کلیک می‌کنیم. شکلی به صورت زیر نمایش داده می‌شود که شما با استفاده از آن تعیین می‌کنید که کنترل موردنظر به کدام قسمت فرم بچسبد.

مربع مرکزی بدین معناست که شما می‌خواهید از مقدار DockStyle.Fill استفاده کنید. اشکال زیر تأثیر نحوه چسباندن کنترل به گوشه‌های مختلف را نشان می‌دهد:



Bottom

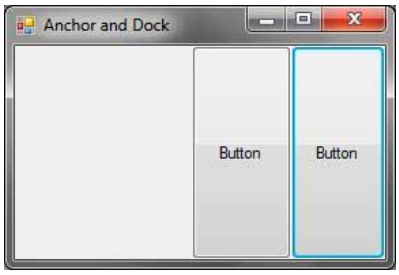
Top

Left

Right

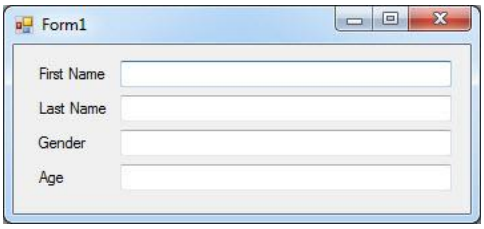
Fill

اگر بخواهید از یک مقدار شمارشی یکسان برای چندین کنترل استفاده کنید، کنترل‌های مورد نظر یا در کنار هم و یا بر روی یکدیگر قرار می‌گیرند.



خاصیت TabIndex

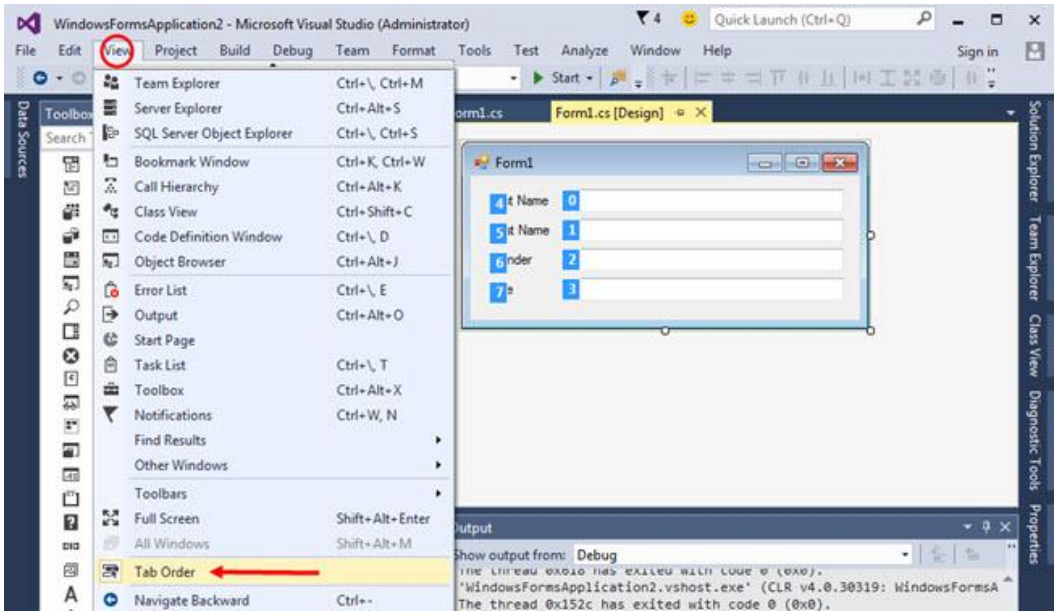
شما با استفاده از کلید TAB می‌توانید در بین کنترل‌های مختلف فرم حرکت کنید. هر کنترل بصری در روی فرم دارای یک خصوصیت به نام TabIndex است که ترتیب فعال شدن آنها را در هنگام فشردن کلید Tab مشخص می‌کند. بهتر است بعد از طراحی فرم، خصوصیت TabIndex تمامی کنترل‌های بصری فرم را بررسی کنید تا از ترتیب فعال شدن آنها در هنگام فشار دادن کلید Tab مطمئن شوید. یک فرم با چهار جعبه متن در نظر بگیرید.



فرض کنید خاصیت TabIndex انها نا مرتب است و مقادیر آنها به صورت زیر است:

TabIndex	TextBox
1	txtFirstName
4	txtLastName
3	txtGender
2	txtAge

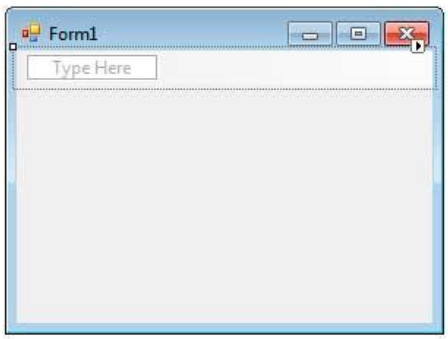
هنگامی که برنامه را اجرا می‌کنید، کنترلی که دارای کمترین مقدار از این خصوصیت است، یعنی کنترل txtFirstName است، ابتدا فعال می‌شود. اگر کلید Tab را فشار دهید دومین کنترل با نام txtAge (چهارمین کنترل در شکل بالا) فعال می‌شود. یک کاربر با تجربه از خصوصیت TabIndex به درستی استفاده می‌کند. اگر از این خصوصیت به درستی استفاده نشود، کاربر نمی‌تواند به درستی تشخیص دهد که کنترل بعدی که فعال می‌شود، کدام است. تعیین ترتیب فعال شدن کنترل‌ها به مکان قرار گیری آنها بر روی فرم بستگی دارد. از آنجایی که TextBoxها در فرم بالا به صورت عمودی قرار گرفته‌اند، بهتر است که ترتیب فعال شدن آنها از بالا به پایین باشد، یعنی خاصیت TabIndex اولی را برابر ۱، دومی را برابر ۲ و... قرار دهیم. در نسخه‌های جدید ویژوال استودیو مانند Visual Studio 2017 یک قابلیت جدید به نام Tab Order وجود دارد که به صورت زیر می‌توانید فقط با چند کلیک ساده ترتیب انتقال فوکوس بین کنترل‌ها را تنظیم کنید:



تنظیم کردن این خصوصیت در کنترل‌هایی که لازم نیست فوکوس بگیرند، الزامی نیست، مانند کنترل Label .

اضافه کردن منو به فرم

نوار منو، در اکثر برنامه به چشم می‌خورد. این نوار شامل دستورات مختلفی است که کاربر از آنها استفاده می‌کند و در ویژوال استودیو با استفاده از کنترل MenuStrip می‌توان آن را به برنامه اضافه کرد. در شکل زیر یک کنترل MenuStrip را از جعبه ابزار بر روی فرم کشیده‌ایم:

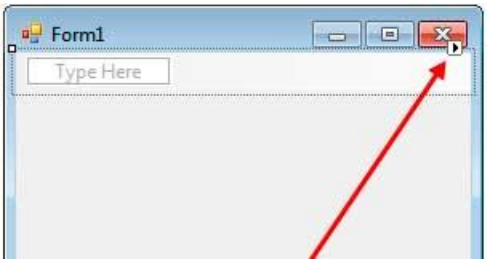


MenuStrip همانند کنترل Timer جزو کنترل‌های غیر بصری بوده و در قسمت Components Tray قرار می‌گیرد. در زیر برخی از خواص پر کاربرد MenuStrip ذکر شده است:

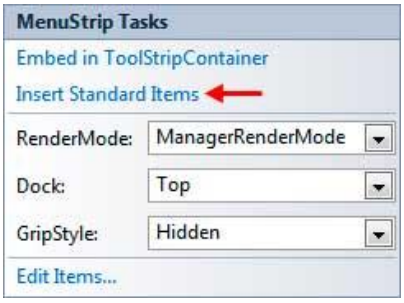
خاصیت	توضیح
Dock	مشخص می‌کند که کنترل منو به کدام گوشه فرم بچسبد. مقدار پیش‌فرض آن Top است
GripStyle	به شما اجازه می‌دهد که مکان منو را تغییر دهید.
Items	کلکسیونی از منوهای اصلی را در بر می‌گیرد.
Stretch	تعیین می‌کند که کنترل منو در سرتاسر در برگیرنده خود کشیده شود.

اضافه کردن منوهای استاندارد

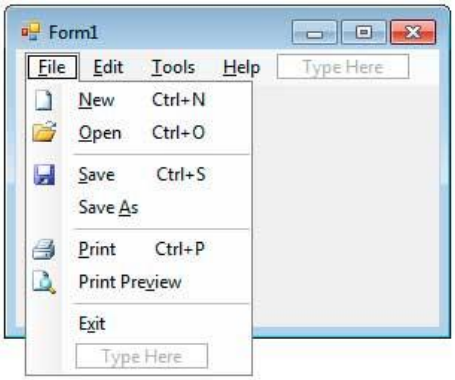
ویژوال استودیو برای اضافه کردن آیتم‌های استاندارد به MenuStrip روشی را پیشنهاد می‌دهد. برای انجام این کار، مانند شکل زیر بر روی فلش کلیک کرده تا صفحه‌ای ظاهر شود:



از صفحه باز شده بروی گزینه Insert Standard Items کلیک کنید:

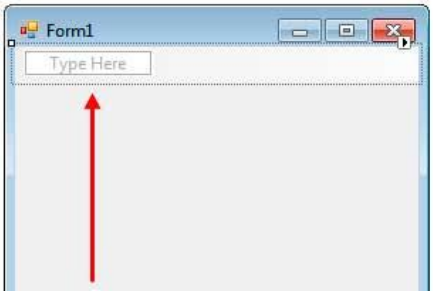


با انجام این کار ویژوال استودیو به صورت خودکار گزینه‌ها و آیکون‌ها مربوط به آنها را مانند شکل زیر به منو اضافه می‌کند:

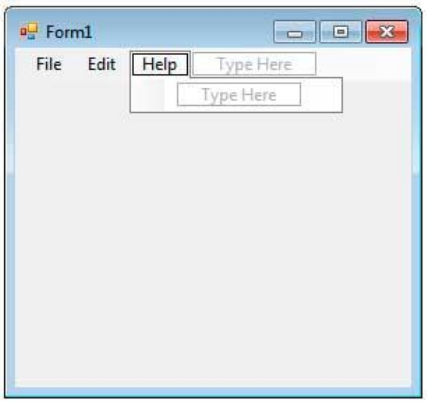


اضافه کردن منوهای دلخواه

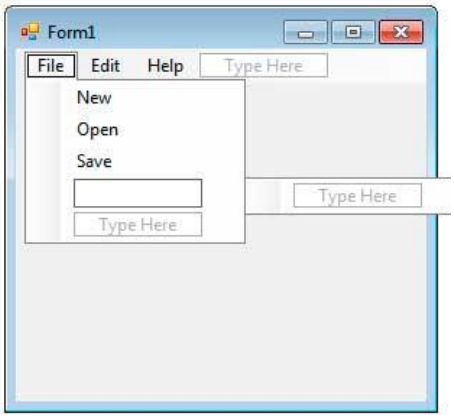
ممکن است در بعضی از شرایط به منوهای خاص نیاز نداشته باشید. شما می‌توانید با استفاده از ابزارهای ویژوال استودیو منوی دلخواهی را ایجاد کنید. ابتدا یک کنترل `MenuStrip` را به فرم اضافه کرده و با کلیک بر روی قسمتی که با فلش نشان داده شده است، یک منو اضافه و عنوان آنرا تایپ کنید.



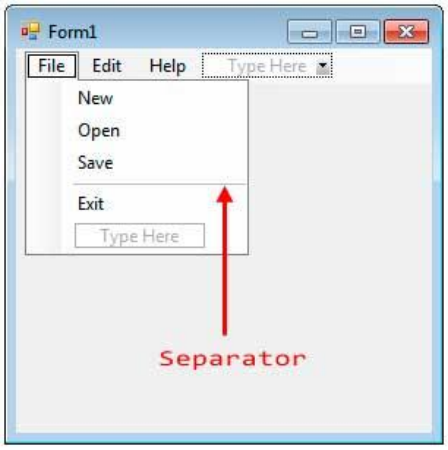
به این نکته توجه نمایید که در هنگام تایپ عنوان منو جعبه دیگری در کنار آن ظاهر می‌شود و شما می‌توانید منوهای دیگری را اضافه نمایید. به عنوان مثال، منوهایی مانند `File`، `Edit`، `Help` در کنترل `MenuStrip` ایجاد کنید.



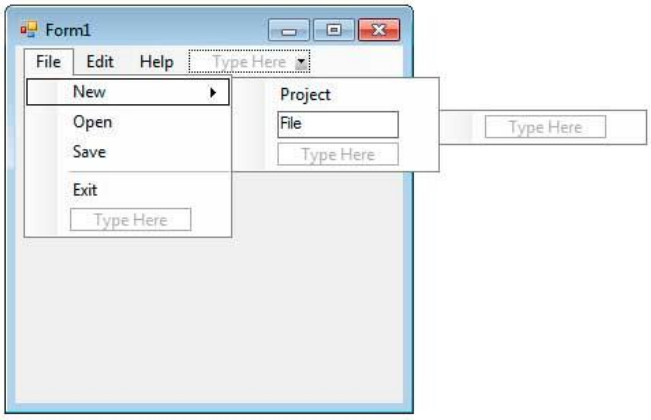
برای اضافه کردن زیر منو بر روی یکی از منوها کلیک کرده تا جعبه‌ای برای اضافه کردن زیرمنو به آن ظاهر شود.



برای ایجاد یک جدا کننده یک کاراکتر خط تیره (-) را تایپ کنید.



همچنین شما می‌توانید برای هر زیر منو مانند شکل زیر، زیرمنوهای دیگری را تعریف نمایید



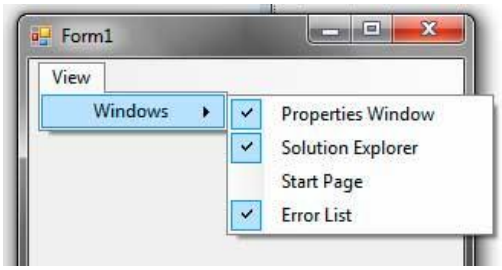
هر کدام از منوهایی که اضافه می‌کنید از نوع ToolStripMenuItem هستند. همچنین نوع جدا کننده ToolStripSeparator هست از ToolStripSeparator فقط برای تقسیم بندی منوهای مرتبط با هم استفاده می‌شود. هر ToolStripMenuItem به خاصیت Items کنترل MenuStrip و هر زیر منو به خاصیت DropDownItems از ToolStripMenuItem اضافه می‌شود. ویژوال استودیو به طور خودکار بر اساس

متنی که برای هر یک از ToolStripMenuItem انتخاب کرده‌اید نامی به آنها اختصاص می‌کند. در زیر برخی از خواص ToolStripMenuItem را مشاهده می‌کنید.

خاصیت	توضیح
Checked	تعیین می‌کند که آیا آیتم تیک خورده است یا نه.
CheckOnClick	تعیین می‌کند که آیتم قابلیت تیک خوردن داشته باشد یا نه.
CheckState	تعیین می‌کند که به طور پیش‌فرض آیتم تیک داشته باشد یا نه.
DropDownItems	زیر منوهای آیتم در این خاصیت قرار می‌گیرند.
Enabled	آیتم را فعال یا غیر فعال می‌کند.
Image	عکسی را به آیتم اختصاص می‌دهد.
ShortcutKeys	کلید میانبر آیتم را تعیین می‌کند.
ShowShortcutKeys	تعیین می‌کند که کلید میانبر در کنار آیتم نشان داده شود یا نه.
Text	عنوان آیتم را تعیین می‌کند.
ToolTipText	وقتی که ماوس را بر روی آیتم قرار می‌دهید، متنی که برای این خاصیت تعیین کرده‌اید نمایش داده می‌شود.

منوهای تیک دار و غیر تیک دار

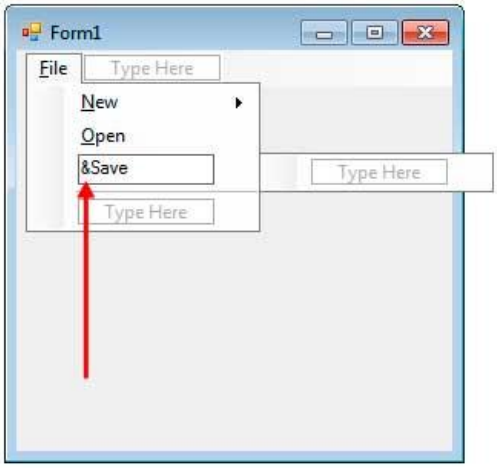
خاصیت CheckOnClick به هر منو این قابلیت را می‌دهد که قابل تیک خوردن باشد یا نه. به عنوان مثال، شما می‌توانید منویی به شکل زیر ایجاد کنید. برای اینکار باید خاصیت CheckOnClick زیر منوهای آن را برابر true قرار دهید. با کلیک بر روی هر آیتم یک علامت تیک در سمت چپ آن نمایش داده می‌شود.



با استفاده از خاصیت‌های Checked و CheckState می‌توانید تعیین کنید که یک آیتم تیک داشته باشد یا خیر.

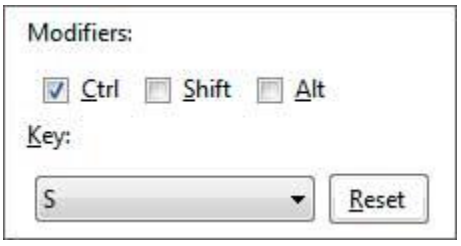
اضافه کرده کلیدهای میانبر به منو

ساده‌ترین راه برای انجام این کار اضافه کرده کاراکتر & در ابتدای نام آیتم است.

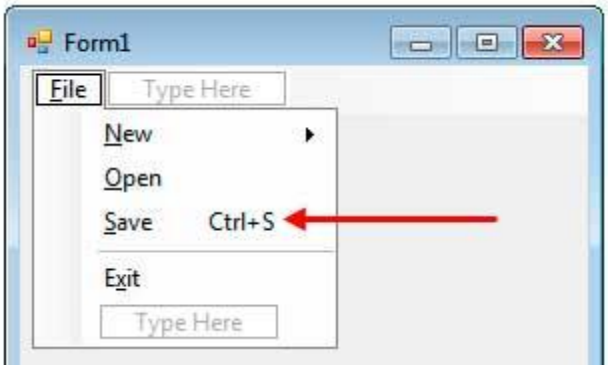


حرف بعد از کاراکتر & به عنوان کلید میانبر در نظر گرفته می‌شود. به عنوان مثال در زیر منوی &Save، حرف S و در E&xit حرف x کلید میانبر است. در قسمت طراحی کلیدهای میانبر به صورت زیر خط دار نمایش داده می‌شوند. در زمان اجرا علامت زیر خط با فشار دادن کلید Alt نمایان می‌شود. برای استفاده از این نوع کلیدهای میانبر باید کلید Alt را ابتدا فشار دهید.

برای فعال کردن یک منو شما باید کلیدهای ترکیبی (کلید میانبر + Alt) را فشار دهید. به عنوان مثال، برای استفاده از منوی File از کلیدهای ترکیبی Alt + F استفاده کنید. برای ایجاد کلیدهای میانبر پیچیده تر باید از خاصیت ShortcutKeys استفاده نمایید. یک آیتم را انتخاب و در پنجره Properties خصوصیت ShortcutKeys را پیدا کنید. بر روی فلش کوچکی که در کنار این خصوصیت قرار دارد کلیک کرده تا پنجره‌ای به شکل زیر نمایان شود.

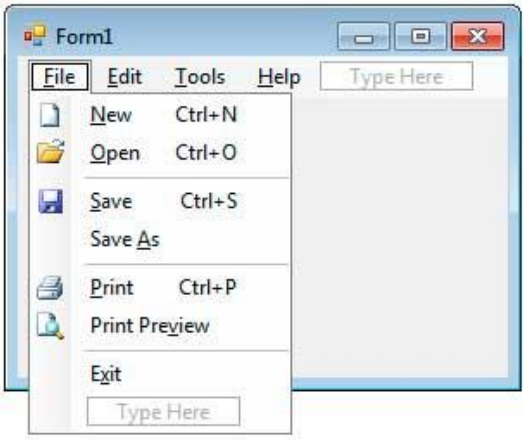


هنگامی که یک کلید ترکیبی را انتخاب می‌کنید، عنوان آن در صورتی در کنار منو (در سمت راست) نمایش داده می‌شود که خاصیت ShowShortcutKeys را برابر true قرار دهید.



اضافه کردن آیکن به منوها

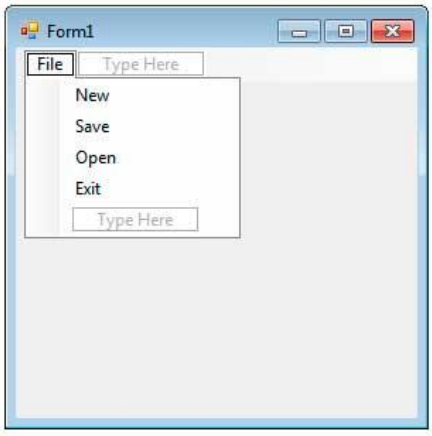
شما می‌توانید برای هر منو آیتم یک تصویر کوچک که در سمت چپ آن نمایش داده می‌شود را، انتخاب کنید.



برای اینکار از خاصیت Image استفاده می‌شود. بر روی دکمه کوچک کنار خاصیت Image در پنجره Properties کلیک کنید. منبعی که عکس در آن قرار دارد را انتخاب و سپس مسیر آنرا مشخص نمایید. اگر سایز عکس بیش از اندازه باشد، به صورت خودکار بر اساس مقدار خاصیت ImageScaling اندازه آن تغییر می‌کند.

اضافه کردن عملکرد به منوها

هنگامی که بر روی یک منو آیتم کلیک می‌کنید رویداد Click آن اتفاق می‌افتد. برای اضافه کردن قابلیتی مشخص به منو آیتم باید کدهای خود را در این رویداد قرار دهیم. برای اضافه کردن کنترل کننده رویداد کلیک به هر منو آیتم، در محیط طراحی بر روی منو آیتم مورد نظر دوبار کلیک کرده تا ویژوال استودیو به صورت خودکار آن را ایجاد کند. برای منو آیتم‌هایی که مقدار خاصیت CheckOnClick آن برابر true است، می‌توانید از رویدادهای CheckedChange و CheckStateChange آنها را استفاده کنید. مانند شکل زیر یک فرم دیگر ساخته و یک کنترل منو به آن اضافه کنید.



بر روی آیتم Exit دوبار کلیک کنید تا ویژوال استودیو کنترل کننده رویداد پیش‌فرض که همان کلیک است را تولید کند. سپس کدهای زیر را در آن قرار دهید.

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

متد استاتیک Exit() از کلاس Application به سادگی برنامه را می‌بندد.

ساخت نوار ابزار

نوار ابزار شامل دکمه‌ها و اجزای مفیدی برای راحتی کار کاربر می‌باشد. مانند منو، نوار ابزار هم در تعداد زیادی از نرم افزارهای مشهور مانند Microsoft Office 2003 به چشم می‌خورد. در زیر نوار ابزار Microsoft Office 2003 را مشاهده می‌کنید:

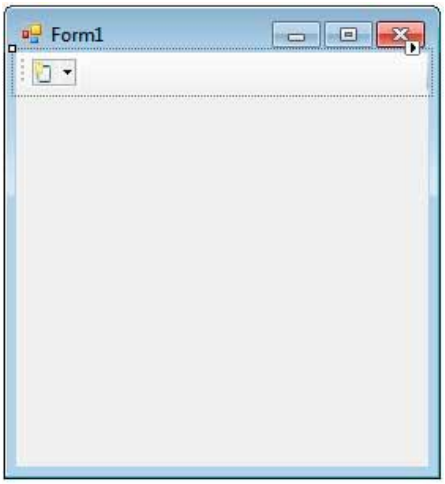
کنترل ToolStrip

دکمه‌ها و اجزای یک نوار ابزار در داخل کنترل ToolStrip قرار می‌گیرند. برای جا به جا کردن کنترل ToolStrip از نقطه چین‌های عمودی که در سمت چپ آن قرار دارند، استفاده می‌شود. در زیر برخی از خواص مهم کنترل ToolStrip توضیح داده شده‌اند:

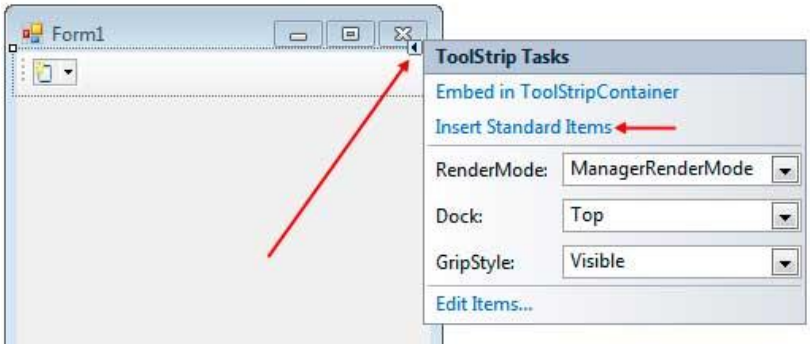
خصوصیت	توضیح
CanOverflow	آیتم‌هایی که داخل کنترل Toolstrip فضای کافی برای نمایش ندارند در یک قسمت مخفی قرار می‌گیرند که این فضا با یک دکمه در دسترس قرار می‌گیرد.
GripStyle	تعیین می‌کند که دستگیره‌ی جابه جایی Toolstrip نمایش داده شود یا خیر.
Items	آیتم‌هایی که در کنترل Toolstrip قرار می‌دهید در داخل این خصوصیت قرار می‌گیرند.
LayoutStyle	نحوه‌ی نمایش آیتم‌ها را تعیین می‌کند.
ShowItemToolTips	تعیین می‌کند tool tip برای آیتم نمایش داده شود یا نه.
Stretch	جهت نمایش ToolStripPanel را تعیین می‌کند.

اضافه کردن آیتم‌های استاندارد به نوار ابزار

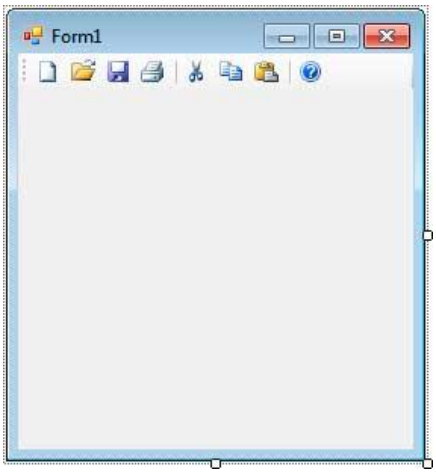
قبل از ایجاد یک نوار ابزار دلخواه، اجازه دهید ببینیم که ویژوال استودیو چگونه به طور خودکار آیتم‌های استاندارد که شامل دکمه‌هایی مانند Save، Open، Copy و Paste می‌باشند را ایجاد می‌کند. یک برنامه ویندوزی جدید ایجاد کرده و نام آنرا ToolBarDemo بگذارید. از قسمت ToolBox یک کنترل ToolStrip را بر روی فرم قرار دهید.



بر روی فلش کوچک واقع در سمت راست کنترل ToolStrip کلیک کرده تا پنجره‌ای به شکل زیر نمایش داده شود:



در پنجره باز شده بر روی گزینه Insert Standard Items کلیک کرده تا ویژوال استودیو به طور خودکار دکمه‌های استاندارد را به صورت زیر ایجاد کند:



خاصیت Items

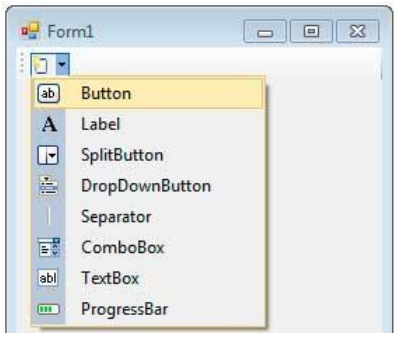
کنترل‌هایی که داخل ToolStrip اضافه می‌کنید، در داخل این خصوصیت قرار می‌گیرند. انواع مختلفی از آیتم‌ها را می‌توان در داخل کنترل ToolStrip قرار داد و همگی آنها از کلاس ToolStripItem ارث بری می‌کنند.

نوع آیتم	توضیح
ToolStripButton	یک آیتم که نمایش دهنده یک دکمه هست. با کلیک بر روی این آیتم می‌توان عملیات مختلفی را اجرا کرد.
ToolStripLabel	یک آیتم که نمایش دهنده یک متن غیر قابل تغییر است. از این آیتم می‌توان برای نمایش عکس هم استفاده کرد.
ToolStripSplitButton	یک آیتم که نمایش دهنده دکمه و فلش رو به پایین در کنار آن است. با کلیک بر روی این دکمه یک منوی باز شونده نمایش داده می‌شود. دکمه خود می‌تواند عملیات جداگانه‌ای را انجام دهد.
ToolStripDropDownButton	مانند آیتم ToolStripSplitButton، این آیتم نیز یک دکمه با فلش رو به پایین را نمایش می‌دهد، ولی در این آیتم دکمه کار خاصی را انجام نمی‌دهد و صرفاً منو را نمایش می‌دهد.
ToolStripComboBox	این آیتم یک کامبو باکس را نمایش می‌دهد.
ToolStripProgressBar	یک نوار پیشرفت به کنترل ToolStrip اضافه می‌کند.
ToolStripSeparator	یک جدا کننده افقی یا عمودی را نمایش می‌دهد.
ToolStripTextBox	یک کنترل TextBox را نمایش می‌دهد.

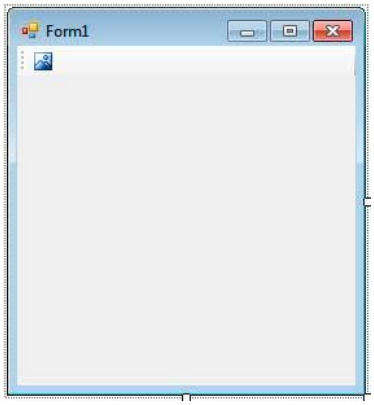
در ادامه نحوه‌ی استفاده از آیتم‌های بالا توضیح داده شده است.

آیتم ToolStripButton

این آیتم رایج‌ترین آیتمی است که شما در یک نوار ابزار می‌بینید. زمانی که بر روی این آیتم کلیک می‌کنید رویداد Click آن همچون یک دکمه معمولی اتفاق می‌افتد. کنترل ToolStrip قبلی را حذف و یک نمونه جدید از آن را روی فرم خود قرار دهید. اگر شما کنترل ToolStrip را انتخاب کنید، یک آیکن با فلش کوچک رو پایین در کنار آن، نمایان می‌شود. با کلیک بر روی این آیکن، لیستی از آیتم‌هایی که می‌توانید به کنترل اضافه کنید، نمایش داده می‌شود.



بر روی Button کلیک کنید تا یک کنترل ToolStripButton به ToolStrip اضافه شود.



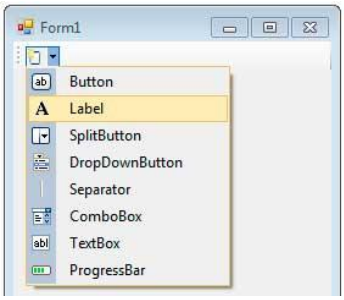
این کنترل از یک آیکن پیش‌فرض استفاده می‌کند. برای تغییر دادن این آیکن به خاصیت Image از پنجره Properties رفته و عکس دلخواه خود را انتخاب کنید. در قسمت طراحی فرم، بر روی دکمه دوبار کلیک کنید تا کنترل کننده رویداد کلیک آن به طور خودکار توسط ویژوال استودیو ایجاد شود. کد زیر را به کنترل کننده رویداد اضافه کنید:

```
MessageBox.Show("Button was clicked!");
```

کد بالا باعث می‌شود هنگامی که بر روی دکمه کلیک کنید یک پیغام ساده نمایش داده شود. در واقع، کدهایی که وظیفه خاصی را انجام می‌دهند مانند تغییر استایل فونت جعبه متن و... در این رویداد قرار می‌دهید.

آیتم ToolStripLabel

این آیتم به شما اجازه می‌دهد که یک متن غیر قابل تغییر را به ToolStrip اضافه می‌کند. همچنین به شما اجازه می‌دهد که یک آیکن کوچک را در کنار آن نمایش دهید. برای اضافه کردن یک ToolStripLabel، بر روی آیکن کوچک کلیک کرده و سپس Label را انتخاب کنید.

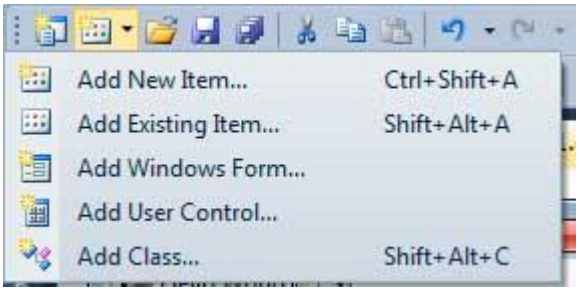


هنگامی که این آیتم را اضافه کردید، می‌توانید از خاصیت Text آن برای تغییر عنوان آن استفاده کنید.

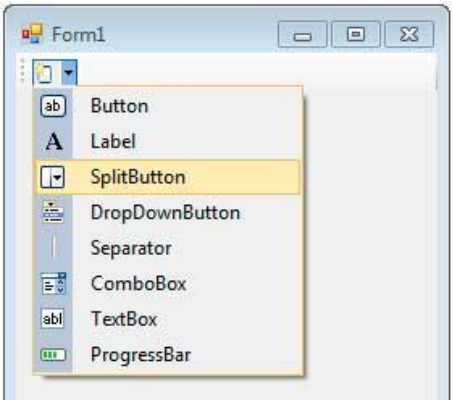
با استفاده از خاصیت Image می‌توان یک عکس را در کنار آن نمایش داد

آیتم ToolStripSplitButton

این آیتم یک دکمه که قابل کلیک کردن است را نمایش می‌دهد. به علاوه اینکه یک فلش رو به پایین در کنار آن قرار دارد که با کلیک بر روی آن یک منوی باز شونده شامل دستوراتی که مرتبط با دکمه هستند، نمایان می‌شود. یک نمونه از این آیتم، دکمه Add New Item در نوار ابزار Visual Studio است.



هنگامی که بر روی دکمه اصلی کلیک می‌کنید، پنجره‌ی Add New Item به جای منو، نمایان می‌شود. برای اضافه کردن کنترل ToolStripSplitButton بر روی new item د ToolStrip کلیک کرده و SplitButton را انتخاب کنید.

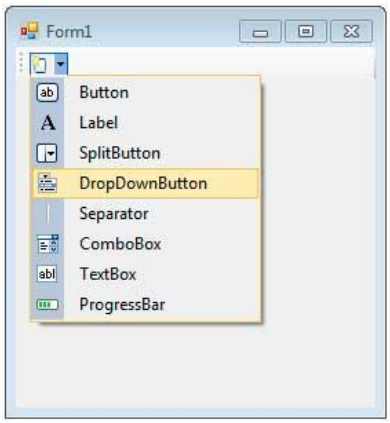


برای اضافه کردن منو به این کنترل، بر روی فلش کوچک کنار آن کلیک کنید و لیست منوها را بنویسید.

دکمه و منو آیتم‌های مرتبط با آن می‌توانند عملیات مختلفی را هنگام کلیک بر روی آنها، انجام دهند. به این نکته توجه فرمایید، که کنترل کننده رویداد دکمه با کنترل کننده منوهای زیر مجموعه آن متفاوت است. همانند کنترل ToolStripButton، می‌توانید برای این کنترل آیکن کوچکی را با استفاده از خاصیت Image تعیین کنید.

آیتم ToolStripDropDownButton

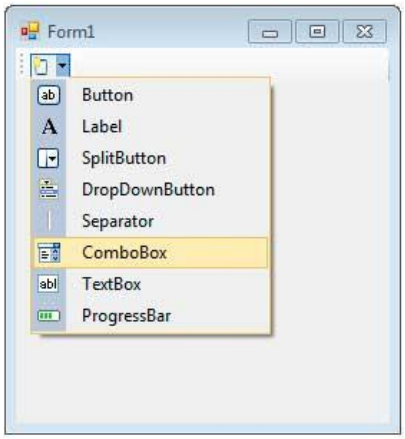
این آیتم بسیار شبیه به آیتم ToolStripSplitButton است، از این جهت که هنگامی که بر روی آن کلیک می‌کنید، یک منوی بازشونده نمایش داده می‌شود. تنها تفاوت بین آنها این است که DropDownButton یک تصویر ساده است، و هنگامی که بر روی آن کلیک می‌کنید تنها منو را نمایش می‌دهد. پس هدف DropDownButton فقط نمایش منو است. برای اضافه کردن این آیتم در لیست بازشونده بر روی DropDownButton کلیک کنید.



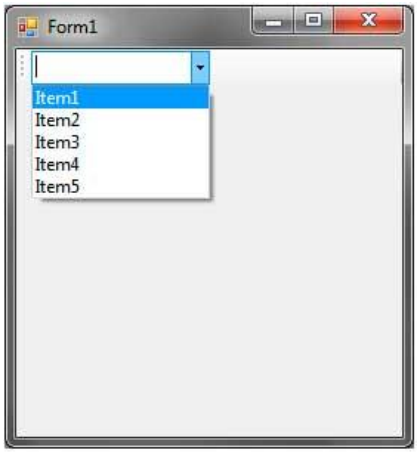
شیوه اضافه کردن منو به این آیتم شبیه به کنترل ToolStripSplitButton است. همچنین می‌توانید به روش مشابه با سایر کنترل‌ها برای این آیتم یک آیکن کوچک انتخاب کنید.

آیتم ToolStripComboBox

این آیتم به شما اجازه می‌دهد که یک کامبو باکس را به ToolStrip اضافه کنید. این آیتم شبیه به کنترل ComboBox است. برای اضافه کردن بخش‌هایی به این کنترل از خاصیت Items استفاده کنید. همچنین می‌توانید از رویداد SelectedIndexChanged استفاده کنید.



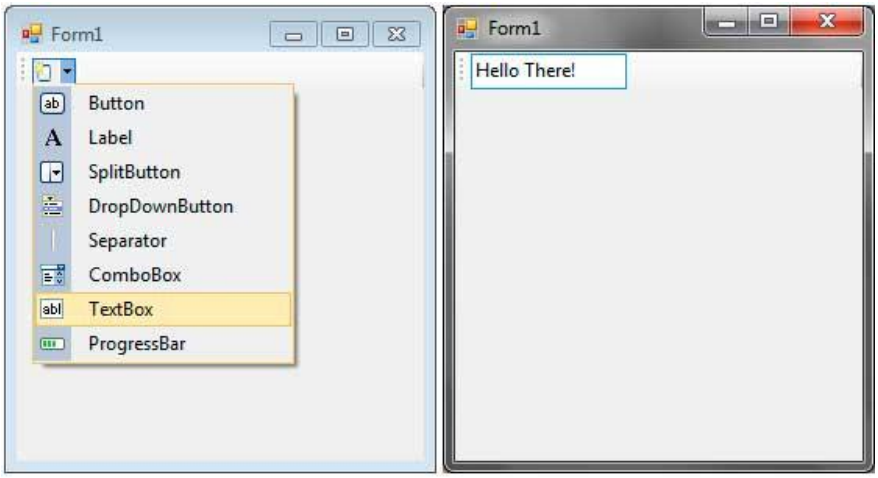
در شکل زیر، یک کامبو باکس با تعدادی آیتم داخل آن نمایش داده شده است.



رویداد پیش فرض این کنترل کلیک است، پس برای استفاده از رویداد SelectedIndexChanged به قسمت رویدادها در پنجره Properties رفته و بر روی قسمت جلوی نام این رویداد دوبار کلیک کنید تا کنترل کننده رویداد آن به صورت خودکار برای شما نوشته شود.

آیتم ToolStripTextBox

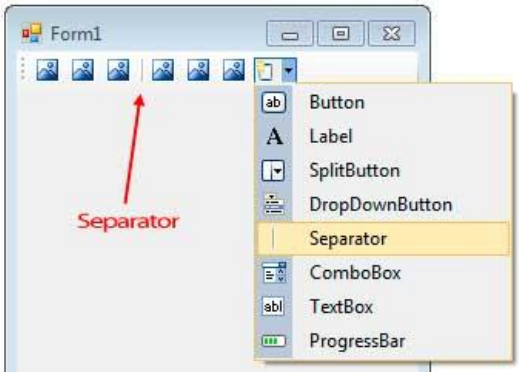
این آیتم نمایانگر یک جعبه متن است. این کنترل شبیه به کنترل TextBox است و خاصیت بسیار مهم آن Text است که هنگام تغییر آن رویداد TextChanged اتفاق می‌افتد.



به این نکته توجه کنید که رویداد پیش‌فرض این آیتم کلیک است. اگر می‌خواهید یک کنترل کننده رویداد به آن اضافه کنید، به پنجره Properties رفته و در قسمت رویدادها، رویداد TextChanged را پیدا و بر روی آن دوبار کلیک کنید تا کنترل کننده رویداد آن اضافه شود.

آیتم ToolStripSeparator

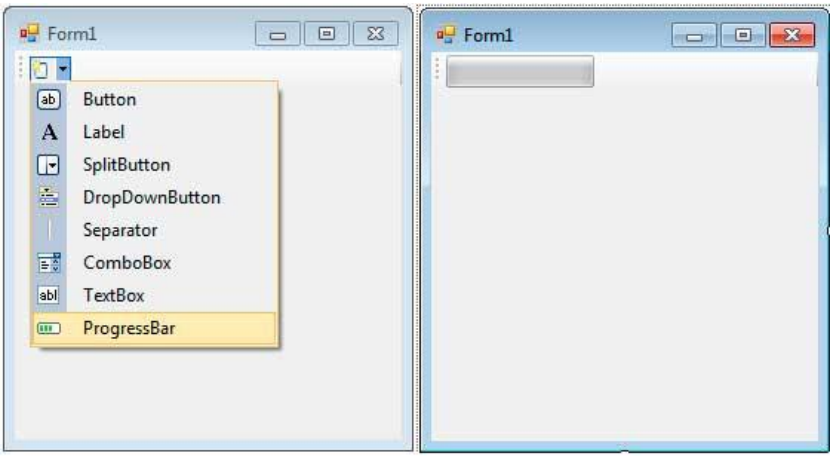
تنها هدف این آیتم جدا کردن آیتم‌ها، به قسمت‌های مرتبط با هم است.



همانطور که در شکل بالا می‌بینید، این آیتم کنترل ToolStrip را به قسمت‌های مختلفی تقسیم می‌کند.

آیتم ToolStripProgressBar

از این کنترل برای نمایش روند پیشرفت یک عملیات مانند ذخیره کردن، حذف کردن یا هر عملیاتی که کاربر نیاز به صبر کردن دارد استفاده می‌شود. با استفاده از این آیتم یک نوار پیشرفت به کاربر نمایش داده می‌شود. بدون وجود این کنترل کاربر ممکن است تصور کند که برنامه قفل کرده یا کار نمی‌کند.

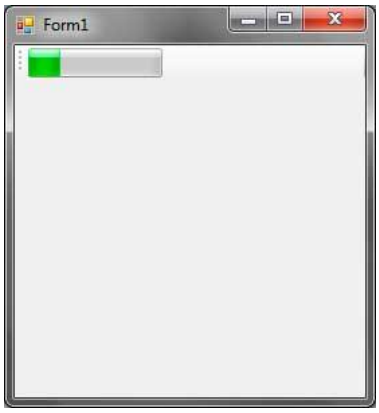


دو خاصیت مهم ToolStripProgressBar عبارت‌اند از Step و Value. خاصیت Value تعیین کننده وضعیت جاری نوار پیشرفت است و خاصیت Step تعیین می‌کند که با فراخوانی متد ()PerformStep چه مقدار به خاصیت Value اضافه شود.

به عنوان مثال، مقدار این خاصیت را برابر 1 قرار داده و یک کنترل Timer از قسمت Components جعبه ابزار بر روی فرم قرار دهید. مقدار خاصیت Enabled تایمر را به true تغییر داده و بر روی آن دوبار کلیک کنید تا کنترل کننده رویداد Tick ایجاد شود. سپس کد زیر را درون آن قرار دهید:

```
toolStripProgressBar1.PerformStep();
```

کد بالا باعث می‌شود هر ۱۰۰ میلی ثانیه یک بار (این مقدار برابر خاصیت interval تایمر است) مقدار خاصیت value کنترل ToolStripProgressBar (که برابر خاصیت Step است) یک واحد افزایش یابد. برنامه را اجرا و نتیجه رامشاهده کنید.

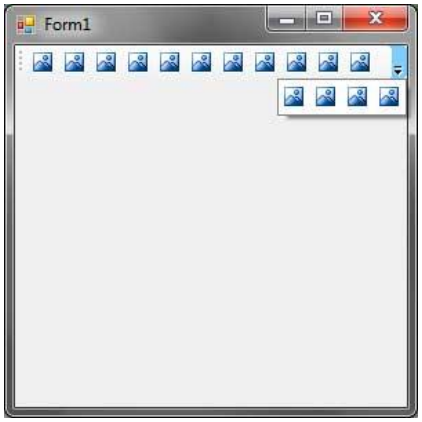


اضافه کردن ToolTip به آیتم‌های ToolStrip

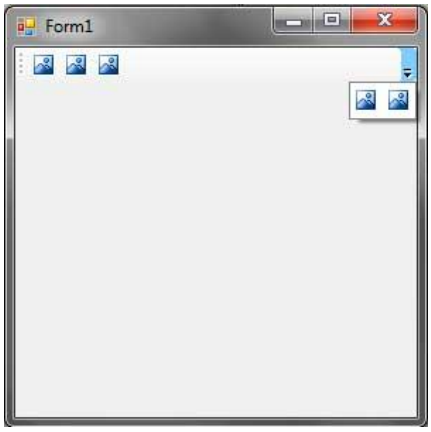
هر آیتم دارای خاصیتی به نام ToolTipText است که، مشخص کننده متنی است که هنگام قرار دادن ماوس بر روی آیتم نمایان می‌شود. به این نکته توجه کنید که ابتدا مقدار خاصیت ShowItemToolTips کنترل ToolStrip را به true تغییر دهید.

استفاده از خاصیت‌های CanOverflow و Overflow

هنگامی که شما تعداد زیادی آیتم به کنترل ToolStrip اضافه کنید برای آیتم‌های بعدی چه اتفاقی افتد؟ به عبارت دیگر برای آیتم‌های بعدی فضای کافی وجود ندارد. به طور پیش‌فرض مقدار خاصیت CanOverflow برابر true است. این کار سبب می‌شود که هنگامی که فضای کافی در کنترل ToolStip وجود ندارد، آیتم‌ها در منوی مخفی که با یک دکمه کوچک در دسترس است، قرار گیرند.

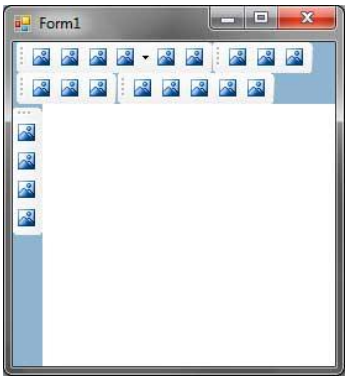


همچنین می‌توانید مقدار خاصیت Overflow از هر آیتم را برابر Always قرار دهید. این کار سبب می‌شود که حتی زمانی که فضای کافی در کنترل وجود داشته باشد آیتم در منوی مخفی قرار گیرد.



کنترل ToolStripContainer

از این کنترل (System.Windows.Forms.ToolStripContainer) به عنوان یک دربرگیرنده برای نوار ابزارها استفاده می‌شود. می‌توانید آرایش و مکان نوار ابزارها را داخل این کنترل تعیین کنید. این کنترل متشکل از تعدادی ToolStripPanel است که کنترل‌های ToolStrip مختلف داخل آنها قرار می‌گیرند. به طور پیش‌فرض، چهار پنل در هر طرف ToolStripContainer وجود دارد.

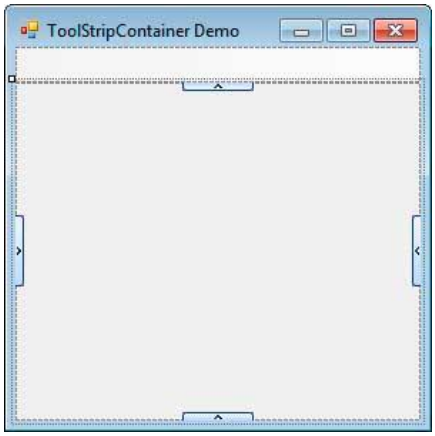


در زیر تعدادی از خصوصیت‌های مهم کنترل توضیح داده شده است.

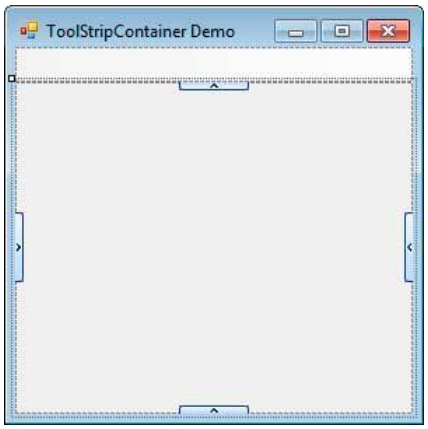
خاصیت	توضیح
BottomToolStripPanel	کنترل ToolStripPanel پایینی را بر می‌گرداند.
BottomToolStripPanelVisible	تعیین می‌کند که ToolStripPanel پایینی نمایش داده شود یا نه.
ContentPanel	پنل مرکزی کنترل را بر می‌گرداند.

LeftToolStripPanel	ToolStripPanel سمت چپ را بر می‌گرداند.
LeftToolStripPanelVisible	تعیین می‌کند که ToolStripPanel چپ نمایش داده شود یا نه.
RightToolStripPanel	کنترل ToolStripPanel سمت راست را بر می‌گرداند.
RightToolStripPanelVisible	مشخص می‌کند که ToolStripPanel سمت راست نمایش داده شود یا نه.
TopToolStripPanel	کنترل ToolStripPanel بالایی را بر می‌گرداند.
TopToolStripPanelVisible	مشخص می‌کند که ToolStripPanel بالایی نمایش داده شود یا نه.

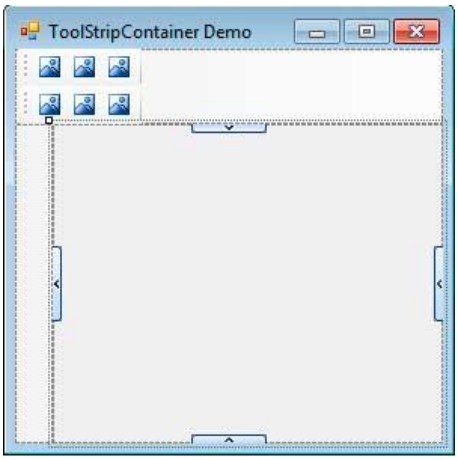
اجازه دهید در عمل از این کنترل استفاده کنیم. یک پروژه Windows Forms Application ایجاد کنید و نام آن را ToolStripContainerDemo بگذارید. یک ToolStripContainer از قسمت ToolBox بر روی فرم قرار دهید. مقدار خاصیت Dock کنترل را برابر Fill قرار دهید.



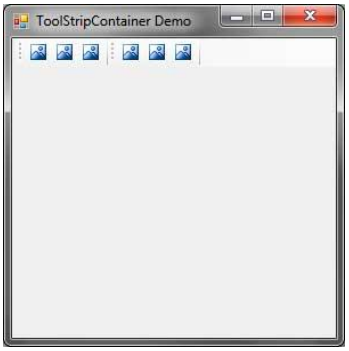
در حالت پیش‌فرض فقط ToolStripPanel بالایی نمایش داده می‌شود. شما می‌توانید ToolStripهای مختلفی را در داخل این پنل قرار دهید. فلش‌های کناری به شما این امکان را می‌دهند که پنل‌های مختلفی را اضافه کنید. به عنوان مثال با کلیک بر روی فلش سمت چپ یک ToolStripPanel جدید ایجاد می‌شود که می‌توانید ToolStripهای مختلفی را در آن قرار دهید.



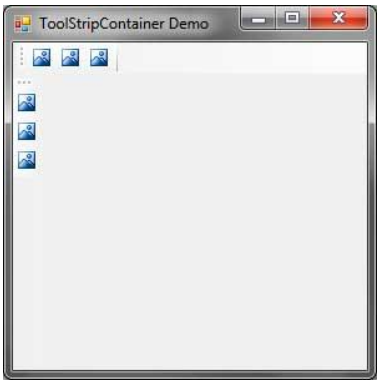
اجازه دهید دو ToolStrip را در پنل بالایی قرار دهیم. همچنین به هر کدام از آنها سه ToolStripButtons اضافه کنیم.



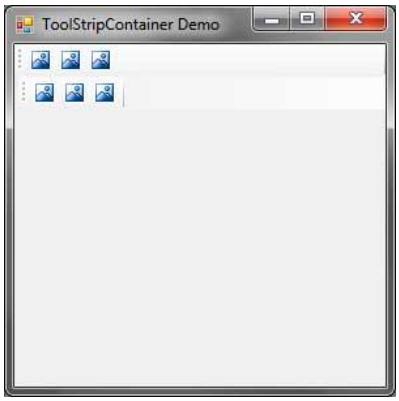
به این نکته توجه کنید که هنگامی که یک ToolStrip جدید اضافه می‌کنید، کنترل جدید در زیر یا جلوی کنترل قبلی قرار می‌گیرد. همچنین سایز پنل برای جای دادن ToolStripها به طور خودکار تغییر می‌کند. برنامه را اجرا و نوار ابزارها را مشاهده کنید. با کلیک کردن یا کشیدن دستگیره جابه جایی هر ToolStrip، می‌توان مکان یا آرایش آن را تغییر داد. به عنوان مثال می‌توانید دومین ToolStrip را به بالای اولین ToolStrip کشیده تا در کنار آن قرار گیرد.



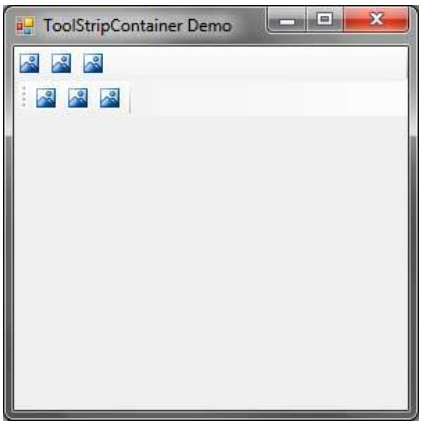
یاد آوری می‌کنیم که یک ToolStripPanel به ناحیه سمت چپ ToolStripContainer اضافه کرده‌ایم. اگر این ناحیه خالی باشد در زمان اجرا مخفی است. ولی شما می‌توانید یه کنترل ToolStrip را مانند شکل زیر در آن قرار دهید.



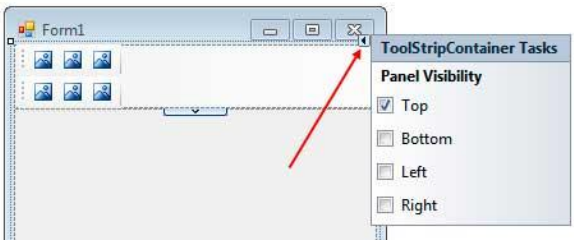
به قسمت طراحی فرم برگردید. اگر می‌خواهید که کنترل ToolStrip تمامی سطر یا ستون را اشغال کند، مقدار خاصیت Stretch آن را برابر true قرار دهید.



برای غیر فعال کردن امکان تغییر مکان ToolStrip باید خاصیت GripStyle آنرا برابر Hidden قرار دهید.

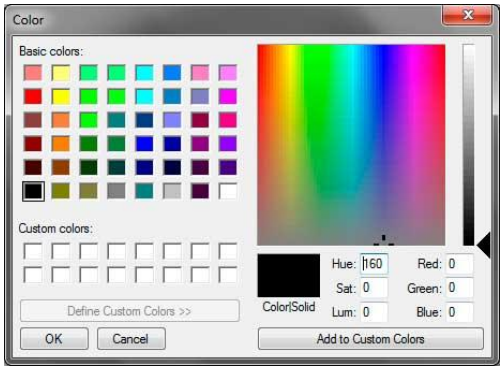


همچنین می‌توانید ToolStripPanelsهای که نیاز ندارید را غیر فعال کنید. برای این کار ToolStripContainer را انتخاب، روی دکمه کوچک بالا سمت راست آن کلیک کنید تا پنجره Smart Task نمایان شود. تیک پنل هایی که نیاز ندارید را بر دارید.

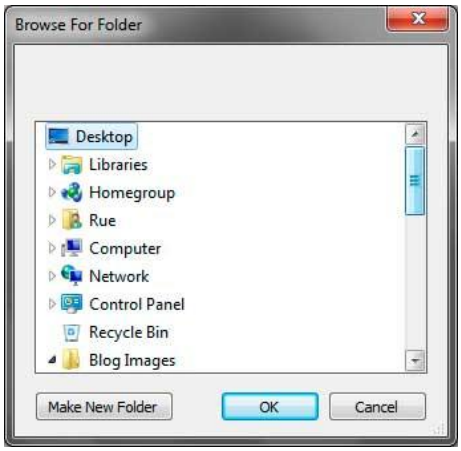


کادرهای محاوره‌ای

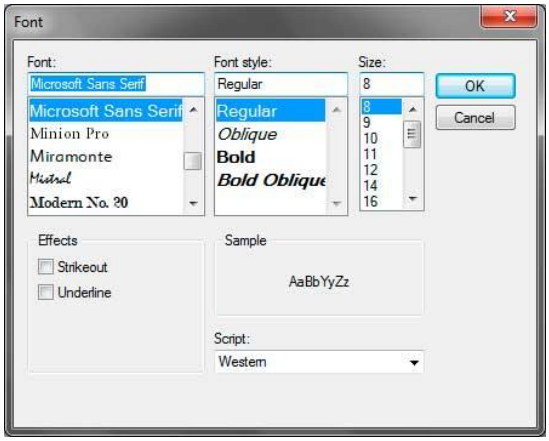
کادرهای محاوره‌ای پنجره‌هایی هستند که دارای وظایف مشخصی مانند ذخیره یا باز کردن یک فایل، انتخاب یک رنگ و یا چاپ یک سند می‌باشند. دات‌نت فریم ورک دارای کنترل‌هایی محاوره‌ای می‌باشد که به شما اجازه می‌دهند با فراخوانی کادرهای محاوره‌ای امور خاصی را انجام دهید. کادرهای محاوره‌ای را می‌توان با استفاده از خواص این کنترل‌ها سفارشی کرد.



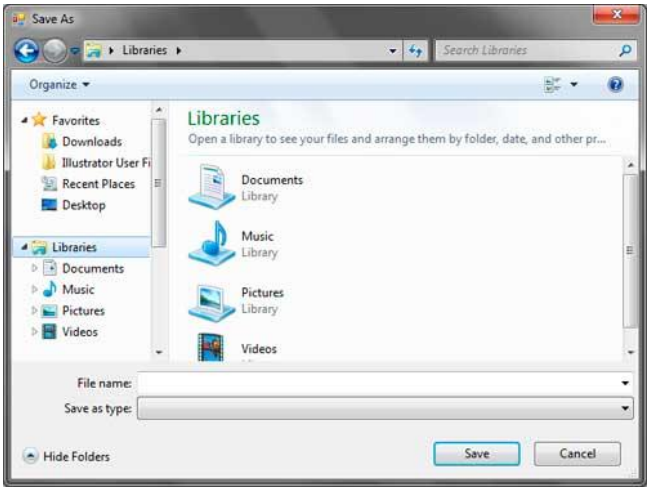
کادر محاوره‌ای انتخاب رنگ



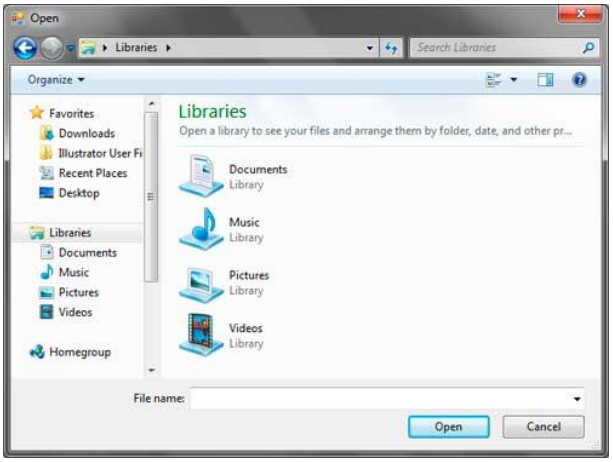
کادر محاوره‌ای جستجوی پوشه‌ها



کادر محاوره‌ای انتخاب سبک فونت



کادر محاوره‌ای ذخیره فایل‌ها

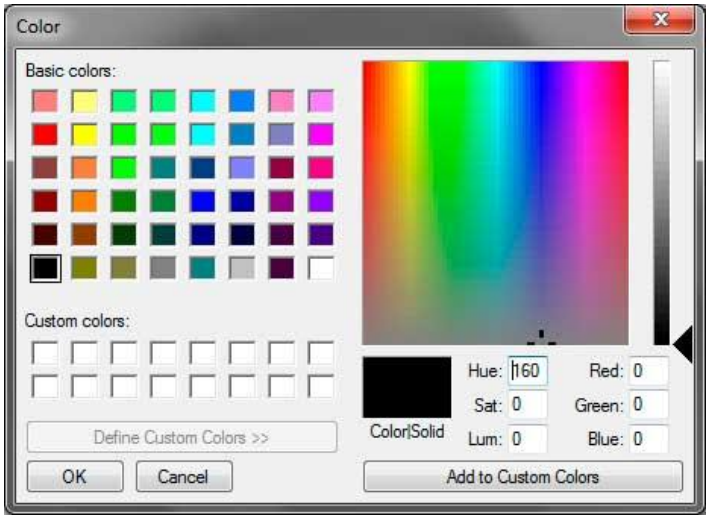


کادر محاوره‌ای باز کردن فایل‌ها

بنابراین می‌توانید با استفاده از این کادرهای محاوره‌ای برنامه‌های تحت ویندوز استانداردتری ایجاد کنید. این کادرها معمولاً در تمام برنامه‌های ویندوزی مورد استفاده قرار می‌گیرند. به این نکته توجه کنید که ظاهر کادرهای محاوره‌ای در نسخه‌های مختلف ویندوز متفاوت است. عکس‌های بالا ظاهر این کادرها را در ویندوز ۷ نشان می‌دهد.

کنترل ColorDialog

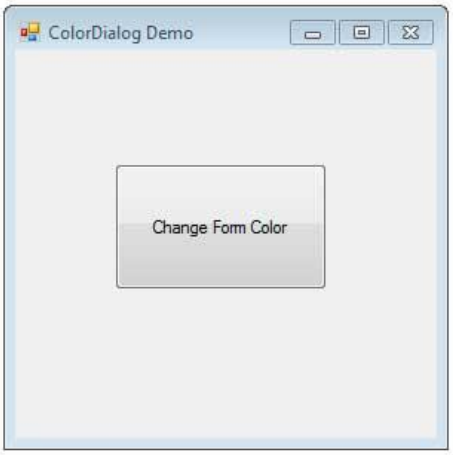
از کنترل ColorDialog وقتی استفاده می‌شود که شما بخواهید یک رنگ انتخاب کنید. به عنوان مثال وقتی بخواهید رنگ یک فونت یا رنگ پس زمینه فرم را تغییر دهید، می‌توانید از این کنترل استفاده کنید.



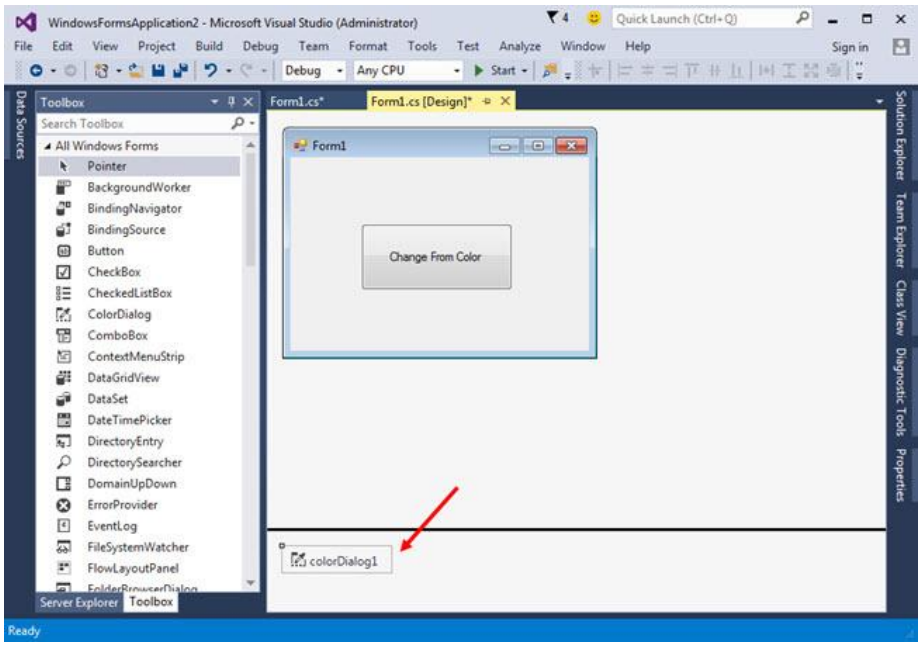
برخی خواص مفید کنترل ColorDialog در زیر ذکر شده است.

خاصیت	خاصیت
AllowFullOpen	مشخص می‌کند که آیا کاربر می‌تواند یک رنگ سفارشی انتخاب کند یا نه.
Color	رنگی که کاربر انتخاب می‌کند را بر می‌گرداند.
CustomColors	کلکسیونی از رنگ‌هایی که کاربر انتخاب کرده است را نشان می‌دهد.

پنجره ColorDialog از رنگ‌های از پیش تعریف شده تشکیل شده است. شما می‌توانید با کیلک بر روی دکمه Define Custom Colors تعداد بیشتری رنگ مشاهده کنید، به طوری که در پنجره ظاهر شده هر رنگی که دوست دارید را، انتخاب نمایید. همچنین می‌توانید از گزینه‌های Hue، Sat، Lum و همچنین RGB برای انتخاب رنگ مورد نظر استفاده کرد. فلش سیاه رنگی که در سمت راست پنجره اصلی قرار دارد، به شما اجازه تنظیم شفافیت یک رنگ خاص را می‌دهد. با استفاده از دکمه Add Custom Colors رنگ مورد نظرتان را به پالت Custom Colors برای استفاده‌های بعدی اضافه نمایید. برای نشان دادن عملکرد کادر محاوره‌ای ColorDialog یک فرم ساده به صورت زیر ایجاد می‌کنیم:



یک کنترل ColorDialog به فرم اضافه می‌کنیم. همانطور که مشاهده می‌کنید، این کنترل به قسمت پایینی بخش طراحی اضافه می‌شود.

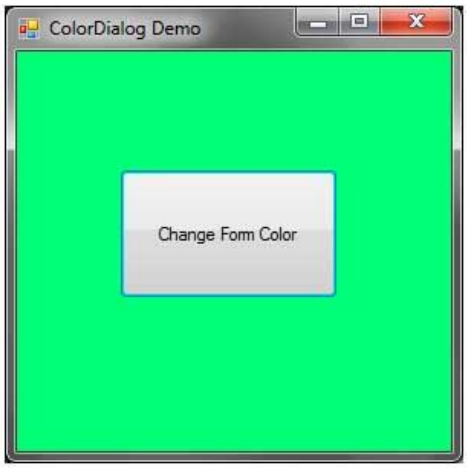


برای تغییر خواص کنترل ColorDialog بر روی آن کلیک کرده و به پنجره Properties بروید. بر روی کنترل Button دوبار کلیک کنید و یک کنترل کننده رویداد برای رویداد Click آن ایجاد نمایید. کد زیر را برای کنترل کننده رویداد بنویسید:

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result = colorDialog1.ShowDialog();

    if (result == DialogResult.OK)
    {
        this.BackColor = colorDialog1.Color;
    }
}
```

خط اول (خط ۲) متد استاتیک ShowDialog() مربوط به نوع شمارشی ColorDialog را فراخوانی می‌کند. این متد باعث ظاهر شدن کادر محاوره‌ای می‌شود. در نتیجه کاربر می‌تواند یک رنگ را انتخاب کند. این متد همچنین یک مقدار System.Windows.Forms.DialogResult بر می‌گرداند که نشان می‌دهد آیا کاربر بر روی دکمه OK کلیک کرده است یا دکمه Cancel.

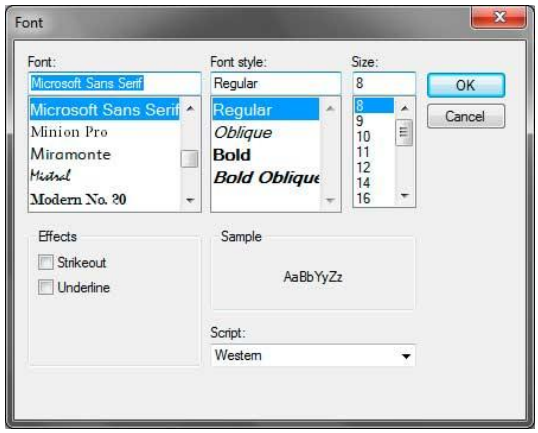
اگر کاربر بر روی یک رنگ کلیک کرده و دکمه OK را بزند کادر بسته می‌شود و رنگی که کاربر انتخاب کرده در خاصیت Color کنترل ذخیره می‌شود. با آزمایش مقدار result می‌توان تشخیص داد که کاربر بر روی دکمه OK کلیک کرده است. در صورتی که دکمه OK توسط کاربر فشار داده شود، رنگ پس زمینه فرم به رنگی در می‌آید که توسط کاربر در خاصیت Color انتخاب شده است.



برنامه را اجرا کرده و بر روی دکمه کلیک کنید تا کادر محاوره‌ای باز شود. یک رنگ را انتخاب کرده و بر روی ok کلیک کنید. مشاهده می‌کنید که رنگ فرم به رنگی که شما انتخاب کرده‌اید تغییر می‌کند.

کنترل FontDialog

کنترل FontDialog کنترلی است که از آن برای انتخاب انواع مختلف فونت و خواص وابسته به فونت استفاده می‌شود.

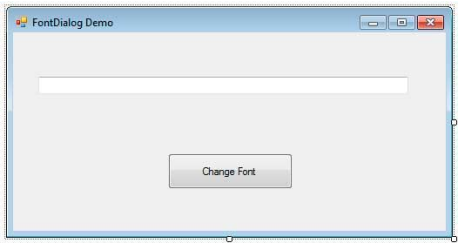


با استفاده از FontDialog می‌توانید نوع، اندازه و رنگ فونت را تغییر دهید. همچنین می‌توانید قبل از اعمال تغییر یک پیش نمایش از تغییرات را مشاهده نمایید. برخی از خواص مفید این کنترل در جدول زیر آمده است.

خاصیت	توضیح
Color	رنگ انتخاب شده توسط کاربر را نشان می‌دهد.
Font	برای تعیین نوع فونت به کار می‌رود.
MaxSize	حداکثر اندازه فونت را مشخص می‌کند.

MinSize	حداقل اندازه فونت را مشخص می‌کند.
ShowApply	نشان می‌دهد که آیا کادر محاوره‌ای شامل دکمه Apply باشد یا نه.
ShowColor	نشان می‌دهد که آیا کادر محاوره‌ای دارای کادری برای انتخاب رنگ باشد یا نه.
ShowEffects	نشان می‌دهد که آیا کادر محاوره‌ای دارای کادری برای زیر خط دار کردن فونت باشد یا نه.

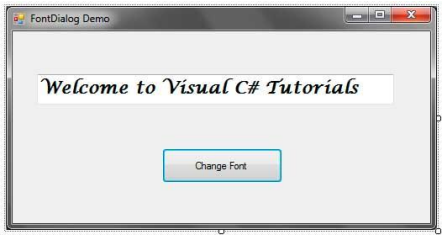
به این نکته توجه کنید که بخش انتخاب رنگ در کادر محاوره‌ای FontDialog در حالت پیش‌فرض مخفی می‌باشد. برای اینکه به کاربر اجازه انتخاب رنگ بدهید، مقدار خاصیت ShowColor را برابر true کنید. خاصیت ShowApply اجازه اضافه کردن دکمه Apply را به کادر FontDialog می‌دهد. در این صورت شما با اجرای رویداد Apply تغییرات را بر روی فونت مورد نظر انجام می‌دهید، بدون اینکه کادر محاوره‌ای بسته شود. به مثال ساده زیر در مورد استفاده از این کنترل توجه کنید. یک فرم شبیه به فرم زیر ایجاد کنید.



ما می‌خواهیم فونت داخل کنترل textbox را با استفاده از کنترل Fontdilog تغییر دهیم. کنترل FontDialog را برروی فرم می‌کشیم. از آنجاییکه کنترل FontDialog یک کنترل غیر بصری است در قسمت پایینی بخش طراحی قرار می‌گیرد. بر روی دکمه دوبار کلیک کنید و در کنترل کننده رویداد ایجاد شده کد زیر را وارد کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result = fontDialog1.ShowDialog();

    if (result == DialogResult.OK)
    {
        textBox1.Font = fontDialog1.Font;
    }
}
```

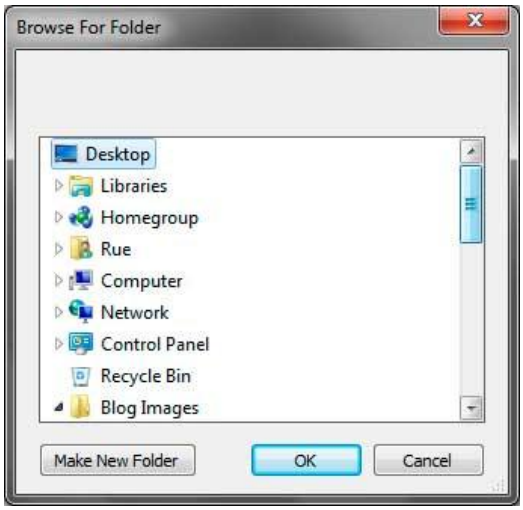
برای نشان داده کادر محاوره‌ای به کاربر ابتدا متد استاتیک ()ShowDialog این کنترل (FontDialog) را فراخوانی می‌کنیم. کاربر اکنون می‌تواند فونت‌های مختلفی را انتخاب کند و سپس دکمه OK را فشار دهد. با استفاده از مقدار برگشتی از متد ()ShowDialog تست می‌کنیم که آیا کاربر دکمه OK را فشار داده است یا نه. اگر کاربر دکمه OK را فشار داده باشد فونت داخل کنترل textbox تغییر خواهد کرد.



برنامه را اجرا کرده و یک متن را در داخل textbox بنویسید. بر روی دکمه کلیک کنید تا پنجره Font Dialog باز شود و سپس یک فونت را به دلخواه انتخاب کنید. در نهایت بر روی دکمه OK کلیک کنید تا فونت textbox تغییر کند.

کنترل FolderBrowserDialog

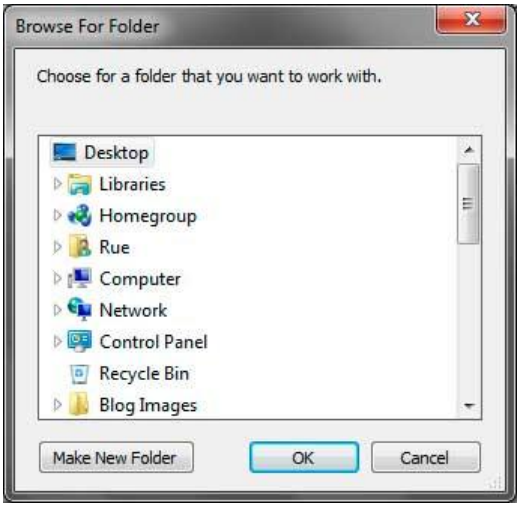
کنترل FolderBrowserDialog به شما اجازه می‌دهد که به دنبال یک فایل یا پوشه در داخل سیستم بگردید. این کنترل از یک نمایش درختی برای نمایش پوشه‌ها استفاده می‌کند. مرورگر این کنترل در حالت پیش‌فرض دسکتاپ و محتویات آن را نمایش می‌دهد. می‌توانید با کلیک بر روی فلش کوچکی که در کنار هر کدام از زیر مجموعه‌های دسک تاپ قرار دارد به زیرپوشه‌ها و فایل‌های آنها دست یافت (شکل زیر). همچنین می‌توان یک پوشه جدید در داخل پوشه انتخاب شده با استفاده از دکمه Make New Folder ایجاد کرد.



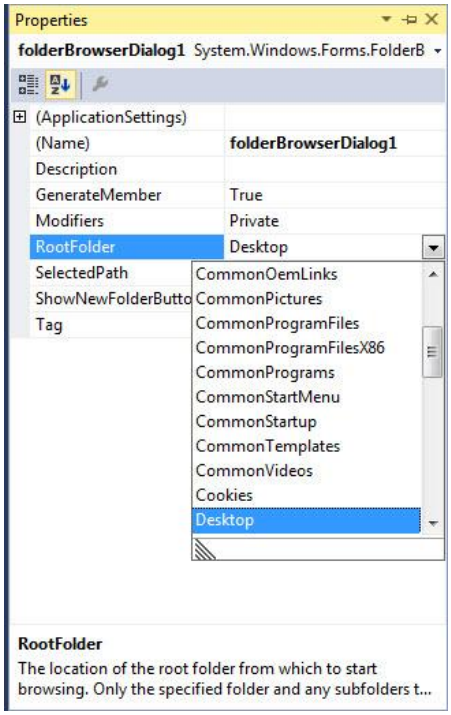
در زیر برخی از خواص مفید کنترل FolderBrowserDialog آمده است.

خاصیت	توضیح
Description	به شما اجازه می‌دهد که یک متن توضیحی به بالای نمایش درختی FolderBrowserDialog اضافه کنید
RootFolder	یک پوشه را به عنوان پوشه اصلی کنترل نمایش می‌دهد. مثلاً دسکتاپ، پوشه اصلی و بقیه زیر پوشه.
SelectedPath	پوشه یا مسیری که کاربر انتخاب کرده است.
ShowNewFolderButton	نشان دادن یا مخفی کردن دکمه Make New Folder.

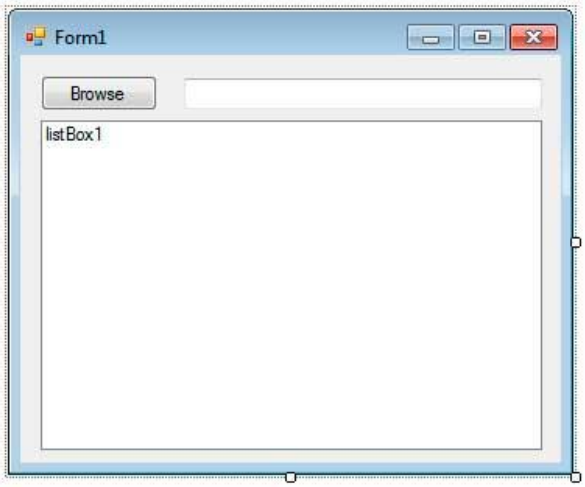
خاصیت Description به شما اجازه می‌دهد که یک توضیح به قسمت بالای نمایش درختی کادر محاوره‌ای کنترل اضافه کنید.



اگر نمی‌خواهید از دسک تاپ به عنوان ریشه اصلی استفاده نمایید (و بقیه پوشه‌ها زیر مجموعه آن باشند) می‌توانید از خاصیت RootFolder استفاده نمایید. در پنجره properties خاصیت RootFolder را یافته و بر روی دکمه drop down کناری آن کلیک کنید. در این صورت مسیرهای از پیش تعریف شده‌ای که می‌توانید آنها را انتخاب کنید به شما نمایش داده می‌شود.



با استفاده از خاصیت SelectedPath می‌توان پوشه یا مسیر انتخاب شده را بدست آورد. مسیر به عنوان یک رشته برگردانده می‌شود. می‌خواهیم با یک مثال کارکرد کنترل FolderBrowserDialog را به شما نشان دهیم. برنامه‌ای که به عنوان مثال در نظر گرفته‌ایم به کاربر اجازه می‌دهد که یک پوشه را انتخاب کرده و سپس مسیر آن را در داخل جعبه متن (text box) نمایش دهد. محتویات پوشه هم در داخل یک لیست باکس فهرست می‌شوند. یک فرم جدید مانند شکل زیر ایجاد کنید. و کنترل‌های button، text box و listbox را در داخل آن قرار دهید.



از گروه Dialogs واقع در Toolbox کنترل FolderBrowserDialog را اضافه کنید. مشاهده می‌کنید که کنترل در قسمت پایینی محیط طراحی قرار می‌گیرد. بر روی دکمه Browse دو بار کلیک کرده و در داخل کنترل کننده رویداد آن کد زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result = folderBrowserDialog1.ShowDialog();

    if (result == DialogResult.OK)
    {
        //Show the path using the text box
        textBox1.Text = folderBrowserDialog1.SelectedPath;

        //Obtain information about the path
        DirectoryInfo selectedPath = new DirectoryInfo(textBox1.Text);

        //Clear the list box first
        listBox1.Items.Clear();

        //Check if there are directories then add a label
        if (selectedPath.GetDirectories().Length > 0)
            listBox1.Items.Add("== Directories ==");

        //Show all the directories using the ListBox control
        foreach (DirectoryInfo dir in selectedPath.GetDirectories())
        {
            //Show only the name of the directory
            listBox1.Items.Add(dir.Name);
        }

        //Check if there are files then add a label
        if (selectedPath.GetFiles().Length > 0)
            listBox1.Items.Add("== Files ==");

        //Show all the directories using the ListBox control
        foreach (FileInfo file in selectedPath.GetFiles())
        {
            listBox1.Items.Add(file.Name);
        }
    }
}
```

این نکته فراموش نشود که لازم است در بالای کد فضای نام System.IO را وارد کنیم:

```
using System.IO;
```

ابتدا با استفاده از متد ShowDialog() کنترل FolderBrowserDialog را فراخوانی می‌کنیم. حال کاربر می‌تواند یک پوشه را انتخاب کرده و دکمه OK را بزند. می‌توانید تست کنید که آیا کاربر دکمه OK را زده است یا نه. برای تشخیص اینکار می‌توانید یک مقدار را به وسیله متد ShowDialog() برگشت دهید (خط ۵). اگر کاربر دکمه OK را فشار داد، سپس مسیر انتخاب شده توسط او در داخل جعبه متن نمایش داده شود.

```
DirectoryInfo selectedPath = new DirectoryInfo(textBox1.Text);
```

این خط با استفاده از مسیر انتخاب شده توسط کاربر یک شیء از DirectoryInfo ایجاد می‌کند. شیء DirectoryInfo در داخل فضای نام System.IO قرار دارد و شامل خواصی است که مشخص می‌کنند که، مسیر پوشه انتخاب شده کجاست و محتویات آن چیست. چون با انتخاب یک مسیر یا پوشه زیر مجموعه‌های آن در داخل لیست باکس فهرست می‌شوند، لازم است برای ورود آیتم‌های جدید، لیست باکس پاک شود. این کار را با استفاده از کد زیر انجام می‌دهیم.

```
if (selectedPath.GetDirectories().Length > 0)
listBox1.Items.Add("== Directories ==");

foreach (DirectoryInfo dir in selectedPath.GetDirectories())
{
   listBox1.Items.Add(dir.Name);
}
```

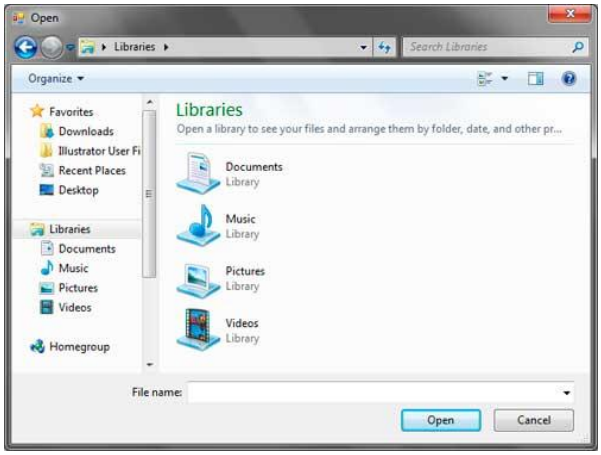
سپس چک می‌کنیم که آیا در داخل پوشه انتخاب شده پوشه‌های دیگری وجود دارند یا نه؟ این کار را با استفاده از خاصیت Length کلاس DirectoryInfo و برگرداندن مقدار بوسیله فراخوانی متد GetDirectories() انجام می‌دهیم. اگر وجود داشت یک کنترل Lable ایجاد کرده و آن را به لیست باکس مان اضافه می‌کنیم. با استفاده از یک حلقه foreach و متد GetDirectories() همه زیر پوشه‌های، پوشه جاری را می‌شماریم. سپس نام هر یک از زیر پوشه‌ها را به لیست باکس اضافه می‌کنیم.

```
//Check if there are files then add a label
if (selectedPath.GetFiles().Length > 0)
    listBox1.Items.Add("== Files ==");

//Show all the directories using the ListBox control
foreach (FileInfo file in selectedPath.GetFiles())
{
    listBox1.Items.Add(file.Name);
}
```

از تکنیک مشابهی برای نوشتن نام همه فایل‌های داخل یک پوشه انتخاب شده استفاده می‌کنیم. اما در اینجا ما از متد GetFiles() و اشیاء کلاس FileInfo استفاده می‌کنیم. شیء FileInfo شامل اطلاعات خاصی درباره یک فایل مشخص است. حال برنامه را اجرا کرده و یک پوشه انتخاب می‌کنیم. بر روی دکمه OK کلیک کرده تا برنامه، پوشه‌هایی را که در داخل پوشه انتخاب شده وجود دارند را به شما نشان دهد.

کنترل OpenFileDialog

کنترل System.Windows.Forms.OpenFileDialog به شما اجازه می‌دهید که یک فایل (مثلاً یک فایل متنی) را انتخاب کرده و محتویات آن را بخوانید. این دیالوگ به شما امکان پیمایش فایل‌های موجود بر روی سیستم را می‌دهد. شما می‌توانید نام یک فایل را در مسیر جاری یا اینکه مسیر کامل آن را در جعبه متنی که در جلوی متن File Name و پایین پنجره قرار دارد، بنویسید.



در جدول زیر لیستی از خاصیت‌های مهم این کنترل توضیح داده شده است.

خاصیت	توضیح
AddExtention	تعیین می‌کند که هنگامی که کاربر نام یک فایل بدون پسوند را تایپ می‌کند یک پسوند به طور خودکار به آن اضافه شود یا نه؟
CheckFileExists	تعیین می‌کند زمانی که کاربر نام یک فایل را که وجود ندارد می‌نویسد یک پنجره‌ی اخطار به او نمایش داده شود یا نه
CheckPathExists	تعیین می‌کند زمانی که کاربر مسیری که وجود ندارد را می‌نویسد یک پنجره‌ی اخطار به او نمایش داده شود یا نه
DefaultExt	پسوند پیش‌فرض. زمانی که کاربر نام فایل را بدون پسوند بنویسد به طور خودکار به آخر فایل اضافه می‌شود.
FileName	فایلی که توسط کاربر انتخاب شده است.
FileNames	مجموعه‌ای از فایل‌ها که توسط کاربر انتخاب می‌شود.
Filter	با استفاده از این خاصیت می‌توانید لیست فایل‌هایی (پسوندهایی) که کاربر می‌تواند انتخاب کند را مشخص کنید.
FilterIndex	اگر پسوندهای متفاوتی در دسترس باشد، با استفاده از این خصوصیت می‌توانید تعیین کنید که کدام پسوند به طور پیش‌فرض هنگام باز شدن پنجره نمایش داده شود.

InitialDirectory	دایرکتوری پیشفرضی که هنگام نمایش پنجره در آن قرار داریم را تعیین می‌کند.
Multiselect	تعیین می‌کند که امکان انتخاب چندین فایل توسط کاربر وجود دارد یا نه.
Title	عنوان پنجره را تعیین می‌کند.

خاصیت AddExtension تعیین می‌کند زمانی که کاربر پسوند فایل را نمی‌نویسد، یک پسوند که در خاصیت DefaultExt مشخص شده است به آن اضافه کند یا نه؟ توصیه می‌شود که مقدار دو خاصیت CheckFileExists و CheckPathExists را برابر true قرار دهید تا زمانی که فایل یا مسیر مشخصی وجود ندارد، به کاربر اخطاری نمایش داده شود. خصوصیت InitialDirectory تعیین کننده پوشه پیشفرضی است که هنگام نمایش دیالوگ در آن قرار داریم. خصوصیت Title نمایانگر متنی است که در عنوان دیالوگ قرار دارد. خصوصیت FileName مسیر فایلی که کاربر توسط دیالوگ انتخاب کرده است را بر می‌گرداند. همچنین با استفاده از خاصیت MultiSelect امکان انتخاب چندین فایل در دیالوگ را می‌دهید. لیست فایل‌های انتخاب شده در خصوصیت FileNames قرار می‌گیرد.

فیلتر کردن فایل‌ها

شما می‌توانید فقط فایل‌هایی که دارای نوع مشخصی هستند را در پنجره نمایش دهید. برای این کار باید یک الگو که یک رشته است را در خاصیت Filter مشخص کنید. به عنوان مثال، می‌توانیم فقط فایل‌هایی که دارای پسوند.txt هستند را در پنجره نمایش دهیم. خاصیت Filter به یک الگوی خاص نیاز دارد.

```
Description1|FilterPattern1|Description2|FilterPattern2|...DescriptionN|FilterPatternN
```

ابتدا باید شرحی از نوع فایل را مشخص کنیم. سپس کاراکتر | و الگوی مشخصی را بعد از آن قرار می‌دهیم. به عنوان مثال الگوی زیر فقط لیست فایل‌های متنی را نمایش می‌دهد.

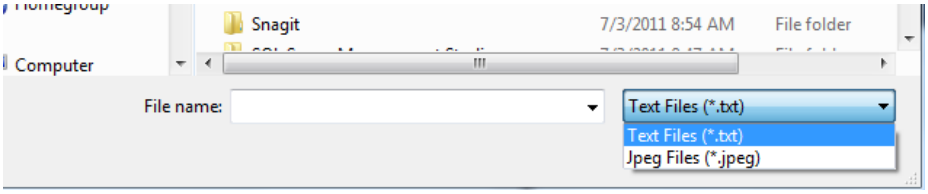
```
Text Files|*.txt
```

شرح نوع فایل Text Files و الگوی آن *.txt است. کاراکتر * یک کاراکتر جایگزین است که بیانگر هر نامی است. قسمت.txt مشخص کننده یک پسوند خاص است. الگو تعیین می‌کند که فایل‌های با پسوند.txt در پنجره نمایش داده شود. همچنین از کاراکترهای جایگزین برای بسیاری از هدف‌های دیگر نیز استفاده کنید.

به عنوان مثال، الگوی m*.txt تمامی فایل‌های متنی که با حرف m شروع می‌شوند را تطبیق می‌دهد. الگوی r*.txt تمامی فایل‌های متنی که نام آنها به حرف r ختم می‌شود را تطبیق می‌دهد. الگوی *.* تمامی انواع فایل‌ها را تطبیق می‌دهد. الگوی *.t* تمامی فایل‌های که پسوند آنها با حرف t شروع می‌شود را تطبیق می‌دهد. شما می‌توانید چندین فیلتر را مشخص کنید. به عنوان مثال الگوی زیر متشکل از چندین فیلتر است.

```
Text Files|*.txt|Bitmap Files|*.bmp|All Files|*.*
```

در جلوی جعبه متنی که نام فایل را می‌نویسید یک کامبو باکس وجود دارد که لیست فیلترهای مختلف در آن قرار دارد.



همچنین این امکان وجود دارد که برای چندین پسوند یک توضیح نوشته شود. به عنوان مثال، فایل‌های تصویری دارای پسوندهای مختلفی از جمله bmp، jpeg، png هستند. شما به راحتی می‌توانید پسوندهای مختلف را با سمیکالن از یکدیگر جدا کنید.

```
Image Files|*.bmp;*.jpeg;*.png;*.gif
```

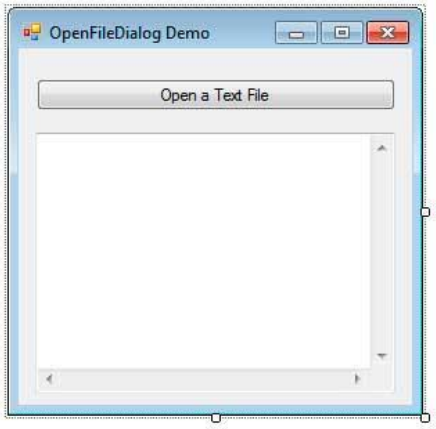
هنگامی که این فیلتر را انتخاب می‌کنید، تمامی فایل‌هایی که حداقل با یکی از الگوهای آن مطابقت داشته باشد، نشان داده می‌شود.

مثالی از نحوه کاربرد OpenFileDialog

در ادامه یک پروژه جدید از نوع Windows Application درست می‌کنیم، که از قابلیت‌های کنترل OpenFileDialog استفاده می‌کند. کاربر با استفاده از برنامه یک فایل متنی را انتخاب می‌کند، سپس برنامه محتوای آن فایل را در یک جعبه متن چند خطی نشان می‌دهد. ابتدا به این نکته توجه کنید که فضای نامی System.IO را به برنامه اضافه کنید.

```
using System.IO;
```

یک فرم به شکل زیر طراحی کنید. از جعبه متنی استفاده کنید که خاصیت MultiLine آن برابر true باشد. مقادیر خاصیت‌های Scrollbars و WordWrap را به ترتیب برابر both و false تغییر دهید. یک کنترل OpenFileDialog را ازجعبه ابزار بر روی فرم قرار دهید.



بر روی دکمه دو بار کلیک کنید تا کنترل کننده رویداد Click ایجاد شود. دوباره یادآوری می‌شود که فضای نامی System.IO را به لیست فضای نامی کلاس اضافه کنید. از کد زیر در داخل کنترل کننده رویداد استفاده کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "Text Files|*.txt";
    openFileDialog1.FileName = String.Empty;

    DialogResult result = openFileDialog1.ShowDialog();
```

```
    if (result == DialogResult.OK)
    {
        Stream fs = openFileDialog1.OpenFile();
        StreamReader reader = new StreamReader(fs);
        textBox1.Text = reader.ReadToEnd();
        reader.Close();

    }
}
```

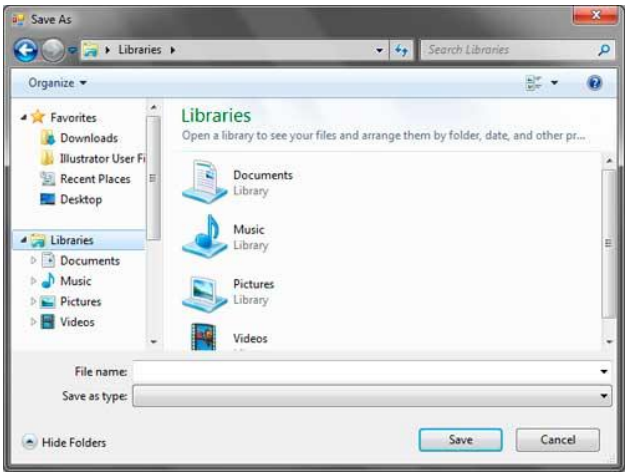
در خط اول با استفاده از خاصیت Filter یک فیلتر را اضافه می‌کنیم. با استفاده از الگوی آن مشخص می‌کنیم که در پنجره فقط لیست فایل‌های متنی نمایان شود. در خط دوم به خاصیت FileName مقدار empty را نسبت می‌دهیم تا در ابتدا هیچ فایلی انتخاب نشود. سپس با استفاده از متد ()ShowDialog پنجره را نمایش می‌دهیم. این متد خروجی از نوع DialogResult دارد.

هنگامی که کاربر یک فایل مجاز را انتخاب و بر روی دکمه Open کلیک می‌کنید خروجی متد برابر DialogResult.OK می‌شود. با استفاده از دستور شرط خروجی متد را بررسی می‌کنیم. خروجی متد ()OpenFile از کنترل OpenFileDialog را در یک شیء از نوع Stream ذخیره می‌کنیم. شیء Stream به فایل انتخاب شده ارجاع دارد، از این شیء در ساخت شیءای از کلاس StreamReader، که کار خواندن از فایل را به عهده دارد، استفاده می‌کنیم.

با استفاده از متد ()ReadToEnd کلاس StreamReader محتوای کامل فایل را خوانده و در خاصیت Text کنترل textBox1 نمایش می‌دهیم. برنامه را اجرا و بر روی دکمه کلیک کنید. یک فایل متنی را از پنجره نمایش داده شده انتخاب کنید. هنگامی که فایلی که مسیر آن توسط کاربر نوشته شده موجود نباشد، یک پیغام خطا به او نمایش داده می‌شود. فراموش نکنید که ابتدا خصوصیات CheckFileExists و CheckPathExists را برابر true قرار دهید. اگر فایل انتخاب شده موجود و مجاز باشد محتوای کامل آن در جعبه متنی قرار می‌گیرد.

کنترل SaveFileDialog

کنترل System.Windows.Forms.SaveFileDialog به شما اجازه می‌دهد که یک فایل را ذخیره یا متنی را در یک فایل بنویسید. این کنترل همچنین به شما اجازه می‌دهد که در بین پوشه‌های موجود در سیستم خود یک پوشه را انتخاب کرده، سپس نام فایلی که قصد دارید اطلاعاتی را در آن قرار دهید را بنویسید. زمانی که نام فایلی که در پوشه جاری پنجره قرار دارد را بنویسید، یک اخطار نمایش داده می‌شود مبنی بر اینکه اطلاعات موجود در فایل دوباره نویسی می‌شود (overwrite). همچنین می‌توانید مسیر کامل فایل را در داخل جعبه متنی که در قسمت پایین آن قرار دارد، بنویسید.



در زیر تعدادی از خاصیت‌های مفید این کنترل را مشاهده می‌کنید.

خاصیت	توضیح
AddExtention	تعیین می‌کند که هنگامی که کاربر نام یک فایل بدون پسوند را تایپ می‌کند یک پسوند به طور خودکار به ان اضافه شود یا نه؟
CheckFileExists	تعیین می‌کند زمانی که کاربر نام یک فایل را که وجود ندارد می‌نویسد یک پنجره‌ی اخطار به او نمایش داده شود یا نه.
CheckPathExists	تعیین می‌کند زمانی که کاربر مسیری که وجود ندارد را می‌نویسد یک پنجره‌ی اخطار به او نمایش داده شود یا نه.
DefaultExt	زمانی که کاربر نام فایل را بدون پسوند بنویسد، یا یک فایل بدون پسوند را انتخاب کند، به طور خودکار یک پسوند پیش‌فرض به نام فایل اضافه می‌شود.
FileName	فایلی که توسط کاربر انتخاب شده است.
Filter	با استفاده از این خاصیت می‌توانید لیست فایل‌های داخل پنجره را فیلتر کنید.
FilterIndex	اگر فیلترهای متفاوتی در دسترس باشد، با استفاده از این خصوصیت می‌توانید تعیین کنید که کدام فیلتر به پنجره اعمال شود.
InitialDirectory	دایرکتوری پیشفرضی که هنگام نمایش پنجره در آن قرار داریم را، تعیین می‌کند.
OverwritePrompt	زمانی که فایل مورد نظر شما در پنجره موجود باشد، یک پیغام به شما نمایش داده می‌شود.
Title	عنوان پنجره را مشخص می‌کند.

خاصیت AddExtension به طور خودکار زمانی که کاربر فایل بدون پسوندی را بنویسد، پسوندی که در خاصیت DefaultExt قرار دارد را به آن اضافه می‌کند. توصیه می‌شود که مقدار دو خاصیت CheckFileExists و CheckPathExists را برابر true قرار دهید تا زمانی که فایل یا مسیر مشخصی وجود ندارد، به کاربر اخطاری نمایش داده شود. خصوصیت InitialDirectory تعیین کننده پوشه پیشفرضی است که، هنگام نمایش دیالوگ در آن قرار داریم. خصوصیت Title نمایانگر متنی است که در عنوان دیالوگ قرار دارد.

مشخص کردن نوع فایل

می‌توان نوع فایل را تعیین کرد. برای این کار از خاصیت Filter استفاده می‌شود. این خاصیت یک رشته حاوی الگوی خاص را، به عنوان مقدار می‌پذیرد. به عنوان مثال، تعیین می‌کنیم که فایلی که قصد ذخیره آن را داریم از نوع متنی می‌باشد. خاصیت Filter باید دارای یک الگوی خاص باشد.

```
Description1|Extension1|Description2|Extention2|...DescriptionN|ExtentionN
```

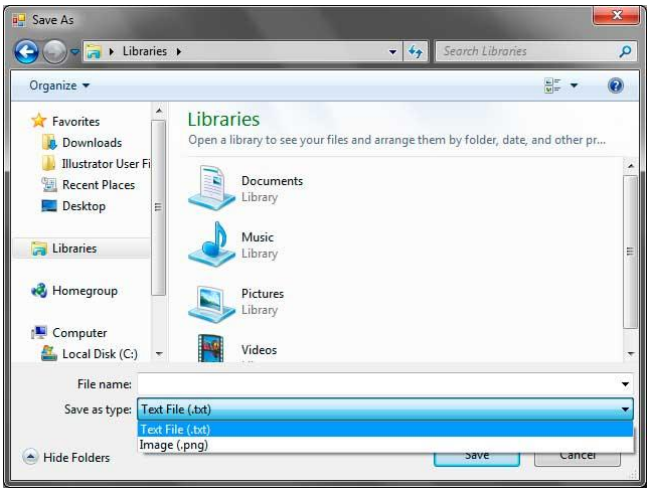
ابتدا برای نوع فایل یک توضیح می‌نویسیم. در ادامه یک کاراکتر | و سپس نوع فایل را می‌نویسیم. به عنوان مثال الگوی زیر به شما اجازه می‌دهد که فایل را فقط به صورت متنی ذخیره کنید.

```
Text Files|.txt
```

شرح فایل در مثال بالا برابر Text Files و پسوند آن برابر .txt است. این امکان وجود دارد که چندین پسوند را مشخص کنید. به عنوان نمونه، الگوی زیر به شما اجازه می‌دهد که یک فایل را به صورت متنی با پسوند .txt یا به صورت تصویری با پسوند .png ذخیره کنید.

```
Text Files|*.txt|Image|*.png
```

نوع فایل را از کامبو باکسی که در زیر جعبه متن نام فایل قرار دارد، انتخاب می‌کنید.

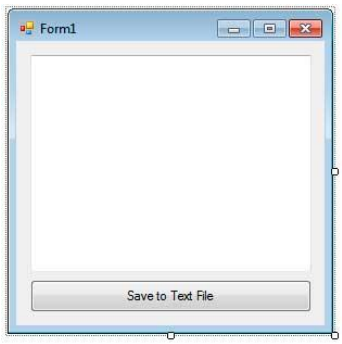


یک مثال از کاربرد کنترل SaveFileDialog

اجازه دهید یک برنامه، که در آن از قابلیت‌های کنترل SaveFileDialog استفاده شده است را طراحی کنیم. برنامه به کاربر اجازه می‌دهد که محتوای یک جعبه متن چند خطی را در یک فایل متنی که به وسیله این کنترل انتخاب می‌شود قرار دهد. ابتدا توجه کنید که کد زیر را به لیست فضای نام‌های برنامه اضافه کنید.

```
using System.IO;
```

یک فرم شبیه فرم زیر طراحی کنید. از جعبه متنی که خاصیت mutiline آن برابر true است، استفاده کنید. یک کنترل SaveFileDialog را از قسمت Dialogs جعبه ابزار بر روی بکشید.



بر روی دکمه دوبار کلیک کنید تا کنترل کننده رویداد کلیک آن ایجاد شود. به شکل زیر کدهای مشابه را در کنترل کننده رویداد قرار دهید.

```
private void button1_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "Text File|.txt";
    saveFileDialog1.FileName = String.Empty;
    saveFileDialog1.DefaultExt = ".txt";

    DialogResult result = saveFileDialog1.ShowDialog();

    if (result == DialogResult.OK)
    {
        FileStream fs = new FileStream(saveFileDialog1.FileName, FileMode.Create);

        StreamWriter writer = new StreamWriter(fs);
        writer.Write(textBox1.Text);
        writer.Close();

    }
}
```

خط اول تعیین می‌کند که کاربر فقط می‌تواند محتوای جعبه متنی را در یک فایل متنی قرار دهد. خط دوم مقدار empty را به خصوصیت FileName نسبت می‌دهد تا در زمان نمایش پنجره، هیچ فایلی به طور پیش‌فرض انتخاب نشود. مقدار خاصیت DefaultExt را برابر .txt قرار داده‌ایم تا اگر کاربر پسوند فایل مورد نظر خود را فراموش کند برنامه به طور خودکار این پسوند را به انتهای نام فایل اضافه کند.

سپس با استفاده از متد ()ShowDialog پنجره را نمایش می‌دهیم. خروجی این متد از نوع DialogResult است. هنگامی که کاربر بر روی دکمه Save کلیک می‌کند خروجی متد برابر DialogResult.OK می‌شود. با استفاده از دستور if خروجی متد را بررسی می‌کنیم. یک شیء از کلاس FileStream ساخته و خاصیت‌های FileName و FileMode آنرا تنظیم کرده‌ایم.

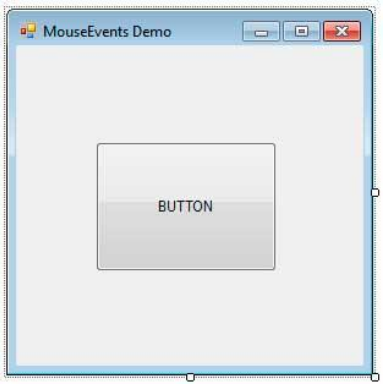
سپس یک شیء از کلاس StreamWriter ساخته و شیء FileStream قبلی را به آن انتقال می‌دهیم. با استفاده از متد ()Write متن موجود در جعبه متن را در استریم نوشته و سپس با استفاده از متد ()close استریم را می‌بندیم. برنامه را اجرا و متنی را داخل جعبه متن بنویسید. بر روی دکمه کلیک کنید و یک پوشه را انتخاب کنید. سپس نام یک فایل را بنویسید و بر روی دکمه Save پنجره کلیک کنید. اگر فایلی با همان نام نوشته شده توسط شما موجود باشد، یک پیغام به شما نمایش داده می‌شود. در صورت تأیید پیغام، محتوای آن فایل کاملاً حذف و محتوای جدید به جای آن قرار می‌گیرد. اگر مقدار خاصیت OverwritePrmpt را برابر false قرار دهید، پیغام خطا نمایش داده نمی‌شود.

رویدادهای ماوس

می‌توان از چندین رویداد خاص برای واکنش نشان دادن به اعمالی که با ماوس انجام می‌شود، استفاده نمود. همانطور که می‌دانید رویداد کلیک زمانی اتفاق می‌افتد که بر روی ماوس کلیک شود. چندین رویداد پیشرفته برای اداره اعمال ماوس وجود دارد. برخی از این رویدادها دارای یک نماینده (delegate) از نوع MouseEventHandler می‌باشند، که آن نیز به نوبه خود دارای پارامتر MouseEventArgs که شامل جزییاتی در مورد اعمال ماوس است، می‌باشد. در زیر رویدادهای ماوس که می‌توانید از آنها استفاده کنید آمده است:

رویداد	توضیح
MouseClick	وقتی روی می‌دهد که با ماوس بر روی کنترل کلیک شود.
MouseDoubleClick	وقتی روی می‌دهد که با ماوس بر روی کنترل دو بار کلیک شود.
MouseDown	وقتی روی می‌دهد که اشاره گر ماوس بر روی کنترل قرار گرفته و دکمه ماوس به سمت پایین فشار داده شود.
MouseEnter	وقتی روی می‌دهد که ماوس وارد کنترل شود.
MouseHover	وقتی روی می‌دهد که با ماوس بر روی کنترل مکث کنیم.
MouseLeave	وقتی روی می‌دهد که ماوس کنترل را ترک کند.
MouseMove	وقتی روی می‌دهد که ماوس بر روی کنترل حرکت کند.
MouseUp	وقتی روی می‌دهد که ماوس بر روی کنترل قرار دارد و دکمه آن رها می‌شود.
MouseWheel	وقتی روی می‌دهد که دکمه وسط ماوس که به شکل چرخ است به سمت بالا یا پایین حرکت کند.

حال اجازه بدهید با برخی از این رویدادها بیشتر آشنا شویم. یک فرم ویندوزی جدید ایجاد کرده و یک کنترل دکمه (button) را در وسط آن قرار دهید (شکل زیر).



دو کنترل کننده رویداد را به رویدادهای MouseEnter و MouseLeave دکمه اضافه کنید. می‌خواهیم کاری کنیم که با ورود ماوس به محدوده کنترل دکمه، اندازه دکمه بزرگ شده و با دور شدن از آن اندازه دکمه به حالت اول برگردد. کنترل دکمه را انتخاب کرده و سپس به پنجره Properties رفته و بر روی آیکون جرقه که پنجره رویدادهای کنترل مذکور است کلیک می‌کنیم. در پنجره رویدادها، رویداد MouseEnter را پیدا کرده و بر روی آن دو بار کلیک می‌کنیم تا کنترل کننده رویدادی برای آن ایجاد شود. کد زیر را وارد می‌کنیم.

```
private void button1_MouseEnter(object sender, EventArgs e)
{
    button1.Height += 30;
    button1.Width += 30;
    button1.Top -= 15;
    button1.Left -= 15;
}
```

وقتی که ماوس وارد کنترل می‌شود طول و عرض کنترل به اندازه ۳۰ پیکسل افزایش می‌یابد. در این حالت برنامه را اجرا کنید و در حالت اجرا چندین بار با ماوس بر روی کنترل بروید. مشاهده می‌کنید که کنترل فقط بزرگ و بزرگ‌تر می‌شود و به حالت اول بر نمی‌گردد. اگر فقط خطوط ۳ و ۴ را درج می‌کردیم، دکمه با هر بار تغییر اندازه، دیگر در وسط فرم قرار نمی‌گرفت. ولی اضافه کردن خطوط ۵ و ۶ باعث می‌شود که این کنترل با هر بار تغییر اندازه در وسط فرم قرار بگیرد. حال نوبت آن رسیده که با خروج ماوس از کنترل، اندازه آن به حالت اول برگردد. در پنجره رویدادها، رویداد MouseLeave را پیدا کرده و بر روی آن دو بار کلیک می‌کنیم تا کنترل کننده رویدادی برای آن ایجاد شود. کد زیر را وارد می‌کنیم.

```
private void button1_MouseLeave(object sender, EventArgs e)
{
    button1.Height -= 30;
    button1.Width -= 30;
    button1.Top += 15;
    button1.Left += 15;
}
```

کد بالا برعکس رویداد MouseEnter عمل می‌کند. رویداد MouseClick نسخه بهبود یافته رویداد Click است. این رویداد به شما اجازه می‌دهد جزییاتی در مورد رویداد Click از جمله مکان کنترلی که با ماوس بر روی آن کلیک شده است را به دست آورید.

اجازه بدهید که یک کنترل کننده رویداد، به رویداد MouseClick کنترل دکمه، اضافه کنیم. ابتدا از پنجره Properties به قسمت رویدادها رفته و رویداد MouseClick پیدا کنید و بر روی آن دو بار کلیک کنید. با استفاده از کد زیر طول و عرض مکانی که شما بر روی آن کلیک می‌کنید نسبت به کوشه سمت چپ بالای کنترل در اختیار شما قرار داده می‌شود:

```
private void button1_MouseClick(object sender, MouseEventArgs e)
{
    MessageBox.Show(String.Format("Clicked at point ({0}, {1})", e.X, e.Y));
}
```

کنترل کننده رویداد با استفاده از شیء MouseEventArgs به مختصات X و Y نقطه ای که شما بر روی آن کلیک کرده‌اید، دست می‌یابد. حال با هر بار کلیک بر روی دکمه، یک پیغام، که نشان دهنده مختصات مکانی است که کلیک کرده‌اید، به شما نشان داده می‌شود. رویداد MouseDown زمانی اتفاق می‌افتد که دکمه ماوس بر روی کنترل فشار داده شود. رویداد MouseUp زمانی اتفاق می‌افتد که دکمه ماوس که بر روی کنترل فشار داده شده رها شود. برای روشن شدن عملکرد این رویداد، یک کنترل کننده رویداد به رویداد MouseDown اضافه می‌کنیم. فقط به یاد داشته باشید که رویداد MouseClick با رویداد MouseDown تداخل دارد. پس رویداد MouseClick را یا حذف کنید و یا به حالت توضیحات در آورید. کد زیر را به کنترل کننده رویداد MouseDown اضافه کنید:

```
private void button1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
        MessageBox.Show("Left Click");
    else if (e.Button == MouseButtons.Right)
        MessageBox.Show("Right Click");
    else
        MessageBox.Show("Middle Click");
}
```

کد بالا از شیء MouseEventArgs برای دسترسی به خاصیت‌های کنترل Button استفاده می‌کند و شامل جزییاتی است که نشان می‌دهد کاربر بر روی کدام دکمه ماوس کلیک کرده است، راست یا چپ؟ نوع شمارشی MouseButton دارای سه مقدار Left، Right و Middle که نشان دهنده سه دکمه ماوس هستند می‌باشد. از رویداد MouseWheel برای تشخیص اینکه آیا دکمه وسط ماوس (که غالباً با شکل یک چرخ با سمت بالا و پایین حرکت می‌کند) حرکت کرده است یا نه استفاده می‌شود. این رویداد در پنجره Properties موجود نیست و برای دسترسی به آن باید کد نوشت. در سازنده فرم، درست زیر متد InitializeComponent() کد زیر را تایپ کنید. دقیقاً بعد از تایپ += دو بار کلید tab را فشار دهید تا کنترل کننده رویداد مربوط به MouseWheel برای شما تولید شود.

```
public Form1()
{
    InitializeComponent();
    button1.MouseWheel+=new MouseEventHandler(button1_MouseWheel);
}
```

کد زیر را نیز برای رویداد MouseWheel بنویسید:

```
void button1_MouseWheel(object sender, MouseEventArgs e)
{
    button1.Height += e.Delta / 60;
    button1.Width += e.Delta / 60;
    button1.Top -= e.Delta / 120;
    button1.Left -= e.Delta / 120;
}
```

در کد بالا از خاصیت Delta از کلاس MouseEventArgs استفاده کرده‌ایم. فرض کنید که کلیک وسط ماوس دارای چرخ دنده است که به سمت عقب و جلو حرکت می‌کنند و مقدار هر یک از آنها برابر با ثابت WHEEL_DATA و ۱۲۰ است. ما مقدار Delta را بر ۶۰ تقسیم کرده‌ایم، بنابراین با هر

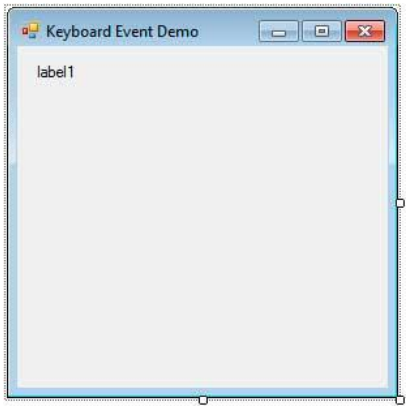
بار حرکت دکمه وسط ماوس، طول و عرض کنترل به جای ۱۲۰ پیکسل، ۲ پیکسل افزایش می‌یابد. با حرکت دکمه ماوس به بالا مقادیر مثبت و با حرکت به پایین آن مقادیر منفی اعمال می‌شوند. دو خط آخر هم برای این است که مکان دکمه در وسط صفحه حفظ شود. در این دو خط مقدار Delta بر ۱۲۰ تقسیم شده است در نتیجه با هر بار چرخش چرخ دنده‌ها، قسمت بالا و چپ دکمه ۱ پیکسل حرکت می‌کند. برنامه را اجرا کنید، از آنجاییکه فوکوس بر روی کنترل button است، با حرکت دکمه وسط ماوس به اندازه کنترل بزرگ و با حرکت آن به سمت عقب اندازه آن کوچک می‌شود. کل کد:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {
        //*************************************************************
        public Form1()
        {
            InitializeComponent();
            button1.MouseWheel+=new MouseEventHandler(button1_MouseWheel);
        }

        void button1_MouseWheel(object sender, MouseEventArgs e)
        {
            button1.Height += e.Delta / 60;
            button1.Width += e.Delta / 60;
            button1.Top -= e.Delta / 120;
            button1.Left -= e.Delta / 120;
        }

        //*************************************************************
        private void button1_MouseEnter(object sender, EventArgs e)
        {
            button1.Height += 30;
            button1.Width += 30;
            button1.Top -= 15;
            button1.Left -= 15;
        }
        //*************************************************************
        private void button1_MouseLeave(object sender, EventArgs e)
        {
            button1.Height -= 30;
            button1.Width -= 30;
            button1.Top += 15;
            button1.Left += 15;
        }
        //*************************************************************
        private void button1_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
                MessageBox.Show("Left Click");
            else if (e.Button == MouseButtons.Right)
                MessageBox.Show("Right Click");
            else
                MessageBox.Show("Middle Click");

        }
        //*************************************************************
        private void button1_MouseClick(object sender, MouseEventArgs e)
        {
            MessageBox.Show(String.Format("Clicked at point ({0}, {1})", e.X, e.Y));
        }
    }
}
```

رویدادهای کیبورد

اگر بخواهید یکی از رویدادهای مربوط به فشرده شدن دکمه‌های کیبورد را کنترل کنید باید رویدادهای KeyPress، KeyDown و KeyUp را کنترل کنید. رویداد KeyDown زمانی اتفاق می‌افتد که یکی از دکمه‌های کیبورد به سمت پایین فشار داده شود و رویداد KeyUp زمانی اتفاق می‌افتد که دکمه فشرده شده رها می‌شود. رویداد KeyPress تلفیقی از دو رویداد بالا است.

می‌خواهیم یک برنامه ایجاد کنیم، به طوریکه هر زمانی که یکی از دکمه‌های کیبورد فشرده شد، مقدار آن به خاصیت Text کنترل Lable اضافه شود. یک برنامه ویندوزی جدید ایجاد کنید و نام آن را KeyBoardEvents بگذارید. سپس یک کنترل Label به آن اضافه نمایید.



خاصیت Text کنترل Lable را پاک کنید. بر روی کنترل form کلیک کرده و از پنجره Properties رویداد KeyPress را یافته و بر روی آن دو بار کلیک کنید تا یک کنترل کننده رویداد ایجاد شود. در داخل کنترل کننده رویداد Form1_KeyPress کد زیر را وارد کنید:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private int charCount = 0;

        private void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            charCount++;

            if (charCount > 30)
            {
                label1.Text += "\r\n";
                charCount = 0;
            }
            else
            {
                label1.Text += e.KeyChar;
            }
        }
    }
```

```
29  }
```

رویداد KeyPress هر وقت که یکی از دکمه‌های کیبورد فشرده شود، اتفاق می‌افتد. در خط ۱۲ یک متغیر به نام charCount تعریف و مقدار دهی اولیه شده است و از آن برای تشخیص تعداد کاراکترهای خط جاری و رفتن به خط بعد استفاده می‌شود. کنترل کننده رویداد تعداد کاراکترهای تایپ شده را به وسیله افزایش یک واحدی charCount در خط ۱۶ تشخیص می‌دهد. در خط ۱۸ تست می‌شود که آیا مقدار charCount از ۳۰ بیشتر است یا نه؟ اگر بیشتر بود، یک خط جدید با استفاده از دستور "\r\n" ایجاد و بقیه کاراکترها تایپ شده در آن نمایش داده می‌شوند. سپس در خط ۲۱ بار دیگر مقدار charCount را به دلیل اینکه در ابتدای یک خط جدید هستیم، صفر می‌کنیم.

اگر مقدار charCount کوچک‌تر یا مساوی ۳۰ باشد، به سادگی کاراکتر تایپ شده توسط کاربر با استفاده از خاصیت KeyPressEventArgs.KeyChar به متن اضافه می‌شود. حال برنامه را اجرا کنید. مشاهده می‌کنید که با زدن هر دکمه کیبورد، متن کنترل lable تغییر کرده و کاراکترهایی را که شما وارد می‌کنید، به آن اضافه می‌شود. وقتی که رویدادهای KeyDown و KeyUp را کنترل می‌کنید، می‌توانید از آرگومان KeyEventArgs که شامل خاصیت‌های زیادی درباره دکمه فشرده شده است، استفاده نمایید. خواص این آرگومان به شرح زیر است:

خاصیت	توضیح
Alt	تشخیص می‌دهد که آیا دکمه Alt فشرده شده است یا نه؟
Control	تشخیص می‌دهد که آیا دکمه Control فشرده شده است یا نه؟
KeyCode	کد اسکی دکمه فشرده شده را می‌گیرد. و با استفاده از آن تشخیص می‌دهد که دکمه خاصی فشرده شده است یا نه؟
KeyData	بسیار شبیه به خاصیت KeyCode است با این تفاوت که فشرده شدن دکمه‌های ترکیبی را نیز تشخیص می‌دهد.
KeyValue	مقدار عددی دکمه فشرده شده را بر می‌گرداند.
Modifier	تشخیص می‌دهد که کدام دکمه‌های ترکیبی (SHIFT, CTRL, ALT) فشرده شده‌اند؟
Shift	می‌گوید که آیا دکمه Shift فشرده شده است یا نه؟
SuppressKeyPress	به شما اجازه می‌دهد که از ورود اطلاعات از طریق کیبورد توسط کاربر جلوگیری کنید.

به عنوان مثال در کد زیر از خاصیت SupressKeyPress استفاده شده است، که در آن فقط اجازه ورود اعداد به کاربر داده شده است (کاربر نمی‌تواند هیچ کاراکتر و یا نشانه‌ای را وارد کند). یک کنترل TextBox را به فرمتان اضافه کرده و یک کنترل کننده رویداد را به رویداد KeyDown اضافه نمایید.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (!(e.KeyCode >= Keys.D0 && e.KeyCode <= Keys.D9 && !e.Shift))
    {
       e.SuppressKeyPress = true;
    }
}
```

شرط داخل دستور if می‌گوید که اگر دکمه تایپ شده توسط کاربر یک دکمه غیر عددی بود یا کلید shift فشرده شد، توسط خاصیت SuppressKeyPress نادیده گرفته شود. خاصیت KeyCode که از آن در شرط استفاده کرده‌ایم شامل مقادیری از نوع شمارشی Keys است. دکمه‌های عددی به وسیله مقایر D0 تا D9 نمایش داده شده‌اند. در ضمن لازم است که چک کنیم که آیا دکمه Shift فشرده شده است یا نه؟ از آنجاییکه نتیجه فشردن دکمه Shift همراه با یک عدد ممکن است منجر به تولید یک علامت مثلاً @ (ترکیب دکمه Shift و عدد ۲) شود، بنابراین باید از خاصیت Shift کلاس KeyEventArgs استفاده کنیم.

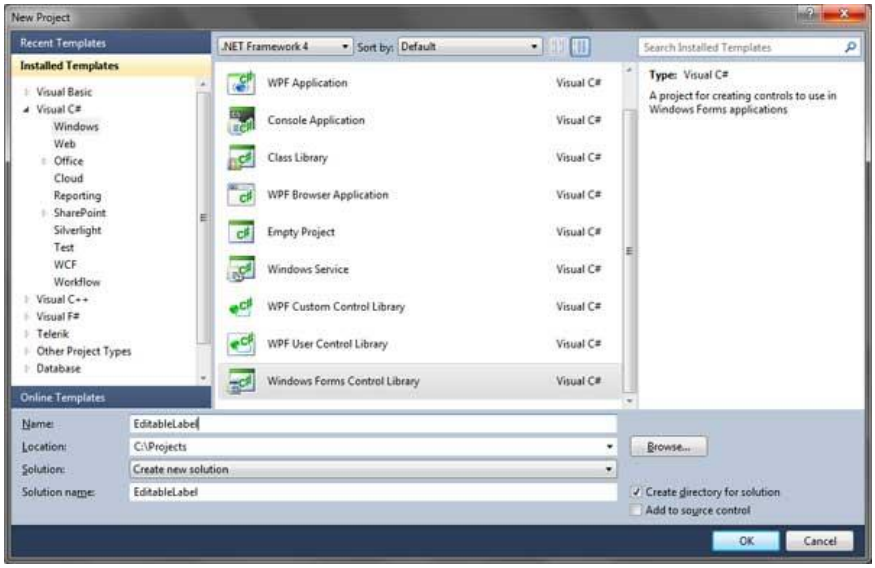
UserControl

ویژوال استودیو این توانایی را به شما می‌دهد که یک کنترل به دلخواه خود، علاوه بر کنترل‌های از پیش تعریف شده در آن، ایجاد کنید. ممکن است که شما دوست داشته باشید یک کنترل با ویژگی‌های خاص داشته باشید و چنین کنترلی در میان کنترل‌های معمول ویژوال استودیو وجود نداشته باشد. این کنترل‌های سفارشی می‌توانند همراه با دیگر کنترل‌ها در قسمت ToolBox قرار گیرند.

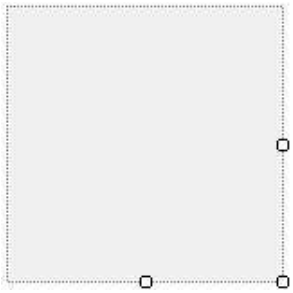
دو نوع کنترل وجود دارد که توسط کاربر تعریف می‌شوند: user control و کنترل‌های سفارشی. user control ترکیبی از کنترل‌های از قبل تعریف شده بوده و ساخت آنها بسیار آسان است. کنترل‌های سفارشی از قبل وجود نداشته و برای ایجاد آنها باید کدنویسی کرد. تمرکز اصلی ما در این درس بر روی User Controls است. چون ایجاد کنترل‌های سفارشی مستلزم آشنایی با مفاهیم پیشرفته برنامه‌نویسی بوده و برای آموزش کسانی که در سطح مبتدی هستند، مناسب نیست.

User controls از کلاس UserControl ارث بری می‌کند. که این کلاس نیز مشتق کلاس Control است، که کنترل‌ها از آن استفاده می‌کنند. همه متدها و خاصیت‌های اجزای بصری یا کنترل‌هایی که در User control به کار رفته‌اند در دسترس شما نیستند چون فقط متدها و خواص کلاس UserControl به شما ارائه می‌شود. ولی می‌توان متدها یا خاصیت‌هایی تعریف کرد که با کنترل‌های User control در ارتباط باشند.

حال یک کنترل به نام EditableLabel ایجاد می‌کنیم. کنترلی که ایجاد می‌کنیم شبیه به یک کنترل lable است و وقتی که کاربر بر روی آن دو بار کلیک می‌کند به شکل یک کنترل Textbox درمی آید که متن lable را در خود دارد. شما می‌توانید متن Textbox را ویرایش کنید و وقتی دکمه enter را فشار دهید Textbox دوباره تبدیل به یک lable ولی با متن ویرایش شده شود. برنامه ویژوال استودیو را باز کرده و یک پروژه جدید را ایجاد کنید. از لیست templates گزینه Windows Forms Control Library را انتخاب کرده و نام آن را EditableLabel بگذارید. (اگر از Visual Studio Express استفاده می‌کنید گزینه Class Library را انتخاب نمایید.) شما می‌توانید یک user control را از منوی Project به برنامه اضافه کنید.



شکل زیر ظاهر می‌شود:



همانطور که می‌بینید شکل بالا شبیه به یک پنجره ویندوزی بدون قاب است. هر کنترلی که به این پنجره یا `user control` اضافه شود جزئی از آن می‌شود. بر روی کادر کلیک کرده و خاصیت `Name` آن را به `EditableLabel` و خاصیت `AutoSize` را به `True` تغییر دهید. یک کنترل `label` را بر روی کادر بکشید و خاصیت `Text` آن را به `Label` و خاصیت `Name` آن را به `labelDisplay` تغییر دهید. اندازه کادر را طوری تغییر دهید که برابر اندازه `label` شود:

اضافه کردن خاصیت‌ها

از آنجاییکه ما یک `user control` ایجاد کرده‌ایم در نتیجه فقط رویدادها یا خاصیت‌های کلاس `UserControl` در دسترس ما قرار دارد. این بدان معناست که ما نمی‌توانیم به خاصیت‌های کنترل `Lable` ی که به فرم اضافه کرده‌ایم، دسترسی داشته باشیم. با وجودیکه کلاس `UserControl` دارای خاصیت `Text` است که از کلاس `Control` به ارث برده اما ما نیاز به توابع بیشتری برای کار با کنترل `Lable` مان داریم.

برای اضافه کردن یک خاصیت به `user control`، نیاز به اضافه کردن خاصیت به کلاس `user control` مان داریم. اگر در محیط طراحی هستید با زدن دکمه `F7` به قسمت کدنویسی بروید و در داخل کلاس `EditableLabel` خاصیت زیر را اضافه کنید:

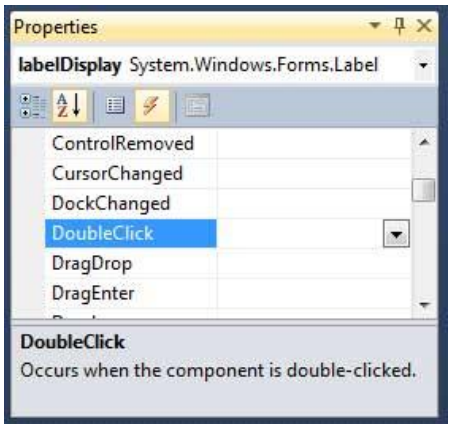
```
[Browsable(true)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
public override string Text
{
    get { return labelDisplay.Text; }
    set { labelDisplay.Text = value; }
}
```

به دو صفت بالای خاصیت توجه کنید. صفت Browsable به خاصیت اجازه می‌دهد که در پنجره Properties محیط Visual Studio نمایش داده شود. اگر این صفت را اضافه نکنید و مقدار آن را برابر true قرار ندهید، فقط در محیط کدنویسی به خاصیت دسترسی خواهید داشت نه در محیط طراحی.

دومین صفت (DesignerSerializationVisibility) به ما اجازه تغییر خاصیت‌های user control در پنجره Property را می‌دهد. به این نکته نیز توجه کنید که ما از کلمه کلیدی override به این دلیل که خاصیت Text در User Control نیز وجود دارد استفاده می‌کنیم.

اضافه کردن کنترل کننده رویداد (Event Handlers)

حال وقت آن رسیده که به user control، کنترل کننده رویداد اضافه کنیم. قبلاً گفتیم که با دوبار کلیک بر روی کنترل EditableLabel مان شکل آن به textbox تغییر می‌کند. بنابراین لازم است که یک کنترل کننده رویداد به رویداد DoubleClick آن اضافه شود. کنترل Lable را انتخاب می‌کنیم. سپس از پنجره Properties به سربرگ رویدادها رفته و رویداد DoubleClick را انتخاب و بر روی آن دو بار کلیک می‌کنیم. با این کار یک کنترل کننده رویداد برای رویداد DoubleClick ایجاد می‌شود.



حال لازم است که یک کنترل TextBox را هم به user control اضافه کنیم. فیلد زیر را در داخل کلاس EditableLabel اضافه می‌کنیم.

```
private TextBox editableTextBox;
```

حال در داخل سازنده کلاس EditableLabel و بعد از متد InitializeComponent کد زیر را اضافه کنید:

```
public EditableLabel()
{
    InitializeComponent();

    editableTextBox = new TextBox();
    this.Controls.Add(editableTextBox);
```

```
    editableTextBox.Hide();

}
```

با کد بالا کنترل TextBox به مجموعه کنترل‌های user control اضافه می‌شود. از آنجاییکه ما می‌خواهیم ابتدا کنترل Lable نمایش داده شود نه textbox، با استفاده از متد ()Hide کنترل textbox را مخفی می‌کنیم. حال اجازه بدهید که به کنترل کننده رویداد کنترل Lable بر گردیم.

```
private void labelDisplay_DoubleClick(object sender, EventArgs e)
{
    editableTextBox.Size = this.Size;
    editableTextBox.Text = labelDisplay.Text;
    labelDisplay.Hide();
    editableTextBox.Show();
    editableTextBox.Focus();
}
```

کد بالا، کنترل کننده رویداد مربوط به رویداد DoubleClick کنترل lable می‌باشد. همانطور که می‌بینید اندازه textbox را برابر با اندازه user control قرار داده‌ایم. سپس متن آن را با متن lable برابر قرار می‌دهیم. در مرحله بعد با استفاده از متد ()Hide کنترل lable را مخفی می‌کنیم.

و در نهایت فوکوس را بر روی textbox قرار می‌دهیم، تا کاربر بتواند آن را ویرایش کند. حال نیاز به یک کنترل کننده رویداد داریم که به کاربر اجازه می‌دهد تغییرات متن textbox را در خاصیت text کنترل lable ذخیره کند. در داخل سازنده کلاس EditableLabel و بعد از متد ()InitializeComponent، کد زیر را وارد کنید:

```
editableTextBox = new TextBox();
this.Controls.Add(editableTextBox);
editableTextBox.KeyDown += new KeyEventHandler(editableTextBox_KeyDown);
editableTextBox.Hide();
```

یک کنترل کننده رویداد را به رویداد KeyDown اضافه می‌کنیم. وقتی که کاربر متن textbox را ویرایش کرد، می‌تواند دکمه Enter را فشار دهد. کنترل کننده رویداد مربوط به رویداد KeyDown را به صورت زیر تعریف می‌کنیم:

```
void editableTextBox_KeyDown(object sender, KeyEventArgs e)
{
   if (e.KeyCode == Keys.Enter)
   {
       labelDisplay.Text = editableTextBox.Text;
       editableTextBox.Hide();
       labelDisplay.Show();
   }
}
```

در کد بالا ابتدا چک می‌کنیم که آیا دکمه Enter توسط کاربر زده شده یا نه؟ سپس متن جاری lable مخفی را برابر متن جدیدی که توسط کاربر نوشته شده است، قرار می‌دهیم. در مرحله بعد textbox را مخفی و کنترل lable با متن جدید را نمایش می‌دهیم. و در آخر وقتی که کنترل lable بسته به طول متن تغییر اندازه داد، user control نیز تغییر اندازه می‌دهد. بر روی labelDisplay در محیط طراحی کلیک کرده و

در پنجره Properties از بخش Event ها رویداد Resize را پیدا کرده و بر روی آن دو بار کلیک و سپس کد زیر را در کنترل کننده رویداد وارد می‌کنیم:

```
private void labelDisplay_Resize(object sender, EventArgs e)
{
    this.Size = labelDisplay.Size;
}
```

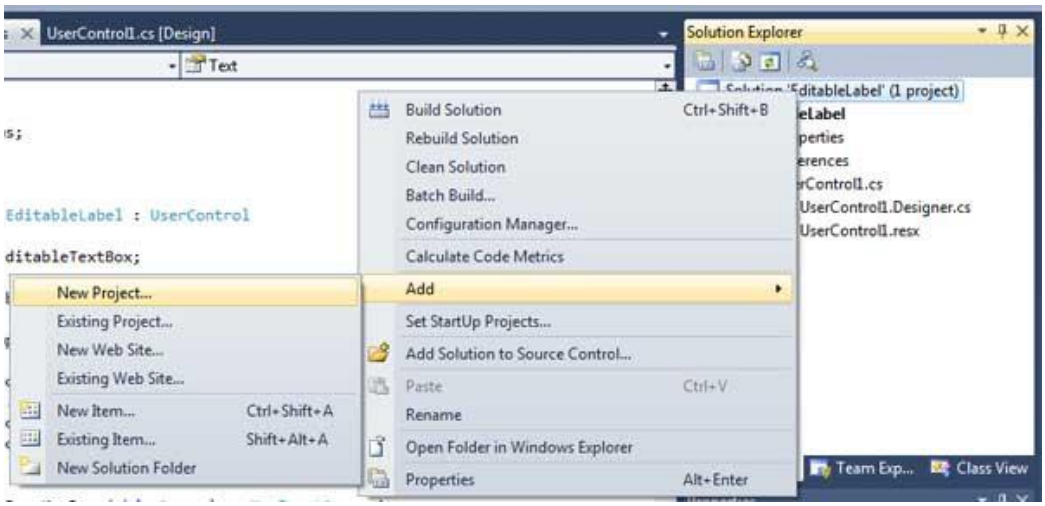
کنترل کننده رویداد به سادگی اندازه user control را برابر اندازه جدید labelDisplay می‌کند.

کامپایل User Control

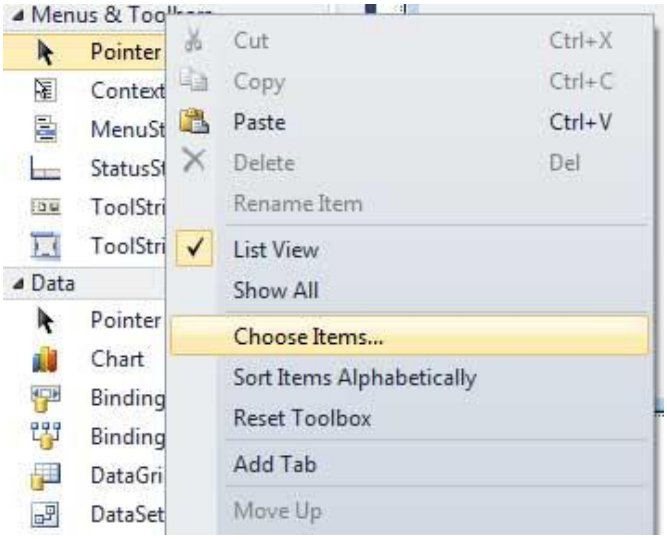
حال نوبت به کامپایل کنترل می‌رسد. برای این کار به قسمت menu رفته و سپس بر روی گزینه Build و بعد از آن Solution Build کلیک می‌کنید. با انجام این عمل یک فایل با پسوند dll. ساخته می‌شود که شامل user control شما می‌باشد.

امتحان کردن کنترل

برای تست کنترل جدید لازم است که یک پروژه جدید ایجاد کنیم. مانند شکل زیر بر روی پروژه‌تان واقع در Solution Explorer راست کلیک کرده و از مسیر New > Project یک پروژه جدید ایجاد کنید.

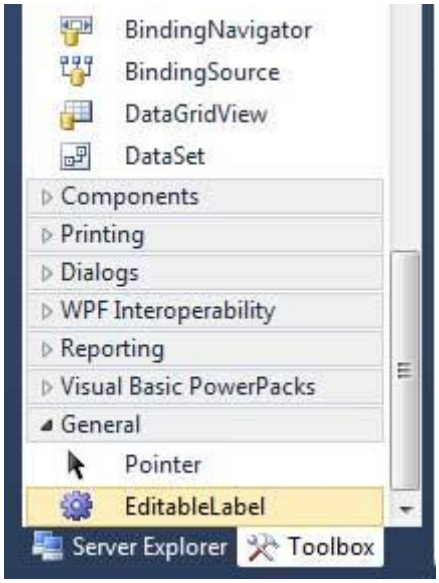


در پروژه ویندوزی جدید ایجاد شده گزینه Windows Forms Application را انتخاب کرده و نام آن را به EditableLabelDemo تغییر دهید. حال لازم است که کنترل جدید (our control) که در ToolBox قرار دارد را به سادگی بر روی فرم بکشیم. وقتی که پنجره ویندوزی جدید ظاهر شد بر روی ToolBox راست کلیک کرده و مانند شکل زیر گزینه "Choose Items..." را انتخاب نمایید:

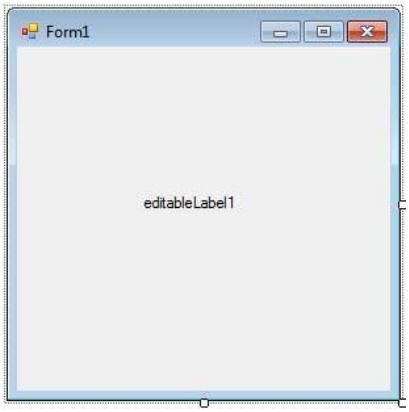


در پنجره ظاهر شده که عنوان آن Choose Toolbox Items می‌باشد بر روی دکمه Browse کلیک کنید. سپس به دنبال فایلdll. ی که شامل user control است بگردید. این فایل در پوشه‌ای که پروژه‌تان در آن ذخیره شده است، قرار دارد. در داخل پوشه پروژه مان که نام آن EditableLabel یک پوشه به نام bin وجود دارد، وارد آن (پوشه bin) می‌شویم.

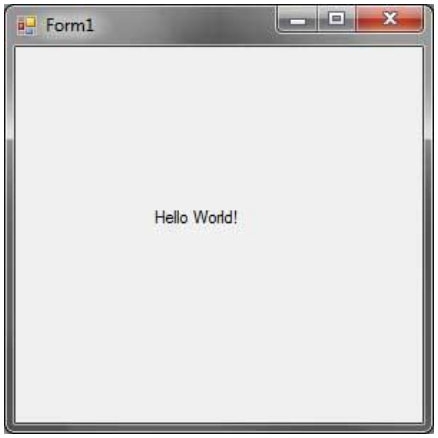
در درون پوشه ممکن است پوشه یا پوشه‌های به نام‌های Debug یا Release وجود داشته باشد که فایل dll در درون آنها باشد. وقتی که فایل EditableLabel.dll را پیدا کردید، آن را انتخاب و بر روی گزینه open کلیک کنید. حال مشاهده می‌کنید که کنترل EditableLabel در درون کنترل‌های قابل انتخاب واقع در Toolbox به نمایش در خواهد آمد.

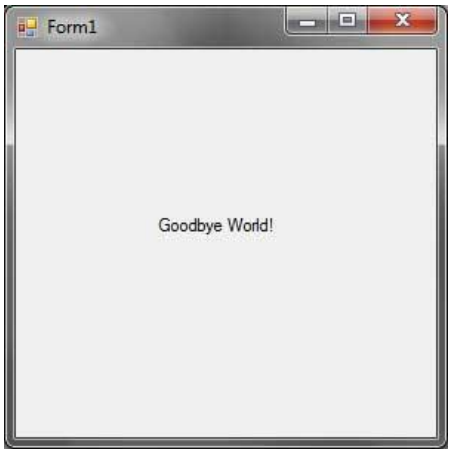


کنترل EditableLabel را بر روی فرم بکشید. با کشیدن کنترل بر روی فرم مشاهده می‌کنید که خاصیت Text این کنترل به خاطر اضافه شدن صفت Browsable، در پنجره Properties قابل مشاهده است.



قبل از اینکه بر روی دکمه F5 کلیک کنید، باید کاری کنید که پروژه EditableLabelDemo به عنوان پروژه پیش‌فرض اجرا شود. در پنجره Solution Explorer بر روی نام پروژه EditableLabelDemo راست کلیک کرده و گزینه Set as StartUp Project را انتخاب نمایید. مشاهده می‌کنید که نام پروژه ضخیم می‌شود. پروژه را اجرا کرده و بر روی label دو بار کلیک کنید. متن را تغییر داده و دکمه Enter را بزنید. همانطور که مشاهده می‌کنید متن lable به متنی که شما در textbox وارد کرده‌اید تغییر می‌کند.





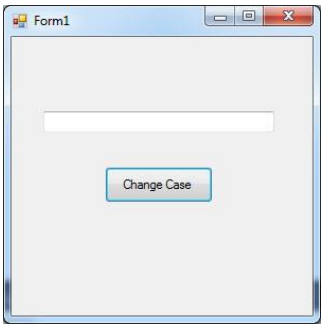
کل کد:

```
using System;
using System.ComponentModel;
using System.Windows.Forms;

namespace EditableLabel
{
    public partial class EditableLabel : UserControl
    {
        public EditableLabel()
        {
            InitializeComponent();

            editableTextBox = new TextBox();
            this.Controls.Add(editableTextBox);
            editableTextBox.Hide();

            editableTextBox = new TextBox();
            this.Controls.Add(editableTextBox);
            editableTextBox.KeyDown += new KeyEventHandler(editableTextBox_KeyDown);
            editableTextBox.Hide();
        }

        private TextBox editableTextBox;
        [Browsable(true)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
        public override string Text
        {
            get { return labelDisplay.Text; }
            set { labelDisplay.Text = value; }
        }

        private void labelDisplay_DoubleClick(object sender, EventArgs e)
        {
            editableTextBox.Size = this.Size;
            editableTextBox.Text = labelDisplay.Text;
            labelDisplay.Hide();
            editableTextBox.Show();
            editableTextBox.Focus();
        }

        void editableTextBox_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                labelDisplay.Text = editableTextBox.Text;
                editableTextBox.Hide();
```

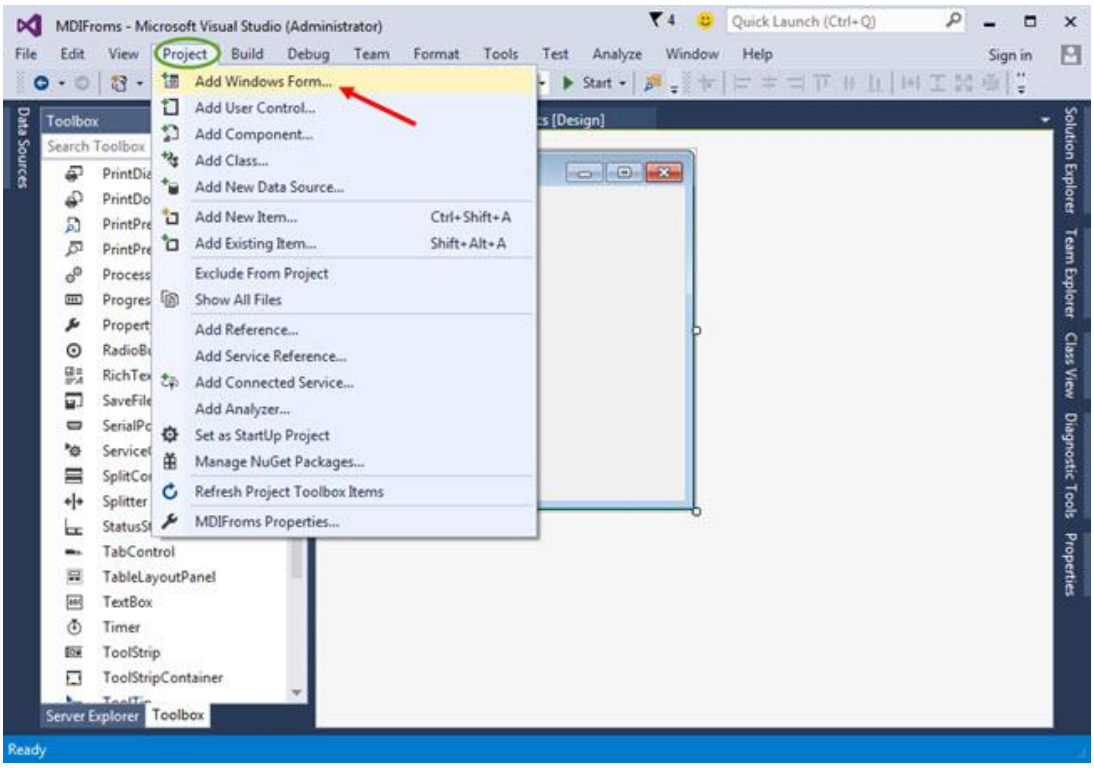
```
                labelDisplay.Show();
            }
        }

        private void labelDisplay_Resize(object sender, EventArgs e)
        {
            this.Size = labelDisplay.Size;
        }


    }
}
```

ایجاد فرم‌های چند گانه در سی‌شارپ

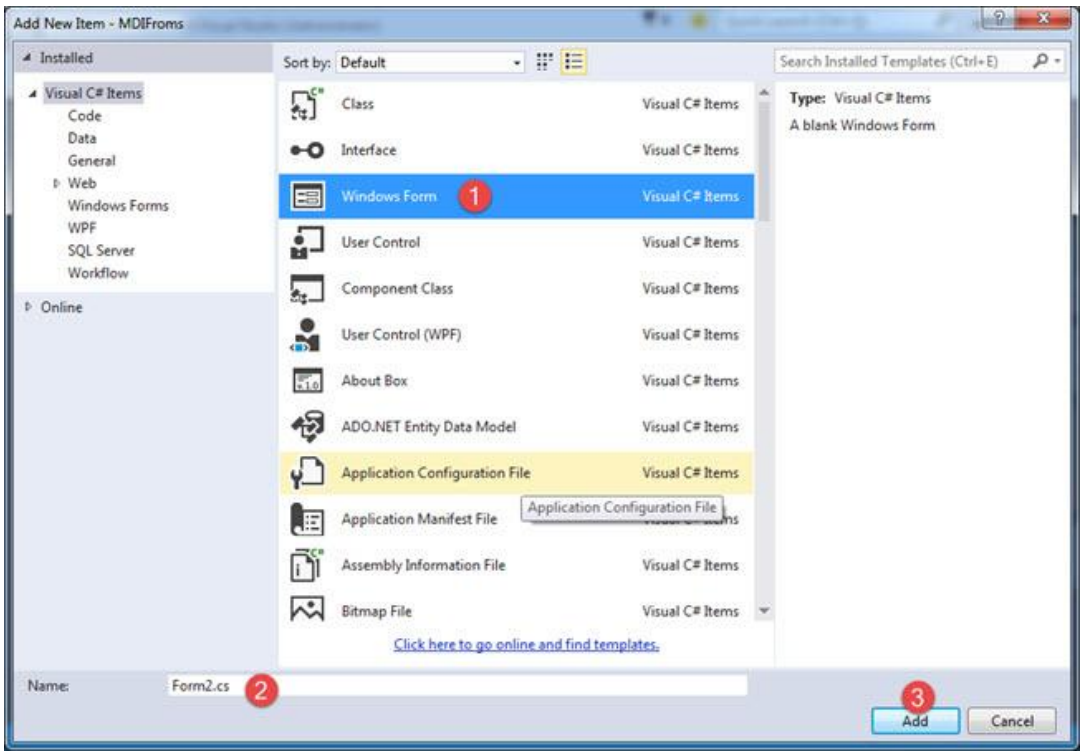
تعداد کمی از برنامه نویسان تنها از یک فرم در برنامه‌هایشان استفاده می‌کنند. بیشتر برنامه‌ها دارای چندین فرم هستند که از طریق فرم اصلی قابل دسترسی می‌باشند. در این درس چگونگی ایجاد برنامه‌هایی با چندین فرم را آموزش می‌دهیم. برنامه ای ساده که دارای یک فرم یک جعبه متن و یک دکمه است مانند شکل زیر، ایجاد می‌کنیم. می‌خواهیم وقتی بر روی دکمه کلیک شد فرم دوم نمایش داده شود. در داخل فرم دوم هم می‌خواهیم به کاربر اجازه تغییر متن داخل جعبه متن در فرم یک را بدهیم. شکل کلی برنامه به صورت زیر است:



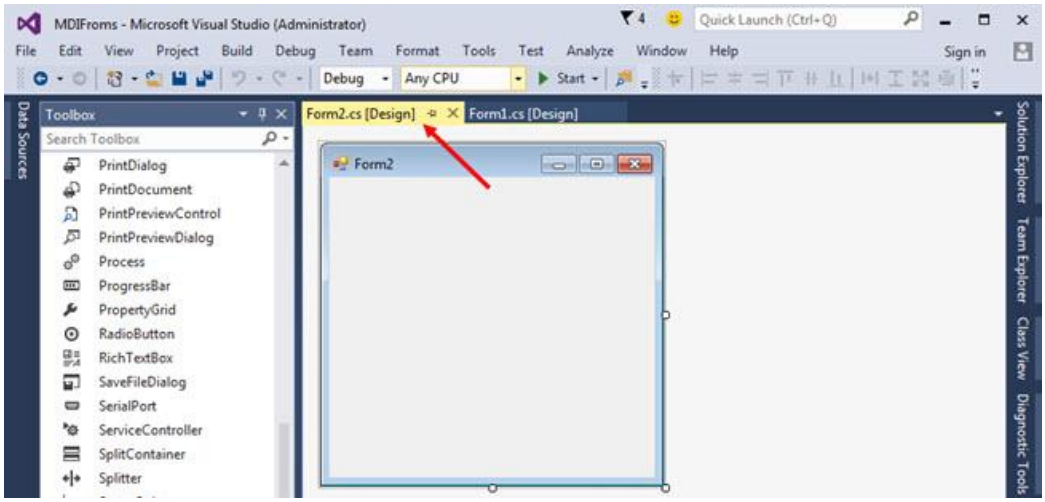
فرم بالا را طراحی کنید. خاصیت Name جعبه متن را به txtChangeCase تغییر دهید. در داخل خاصیت Text جعبه متن هم یک متن پیش‌فرض بنویسید (به صورت حروف کوچک). خاصیت Name دکمه را هم به btnFormTwo تغییر دهید. یک فرم جدید هم به پروژه اضافه کنید. در نوار منو در بالای محیط برنامه سی‌شارپ بر روی گزینه project کلیک کرده و از منوی باز شده گزینه Add New Windows Form را انتخاب کنید:



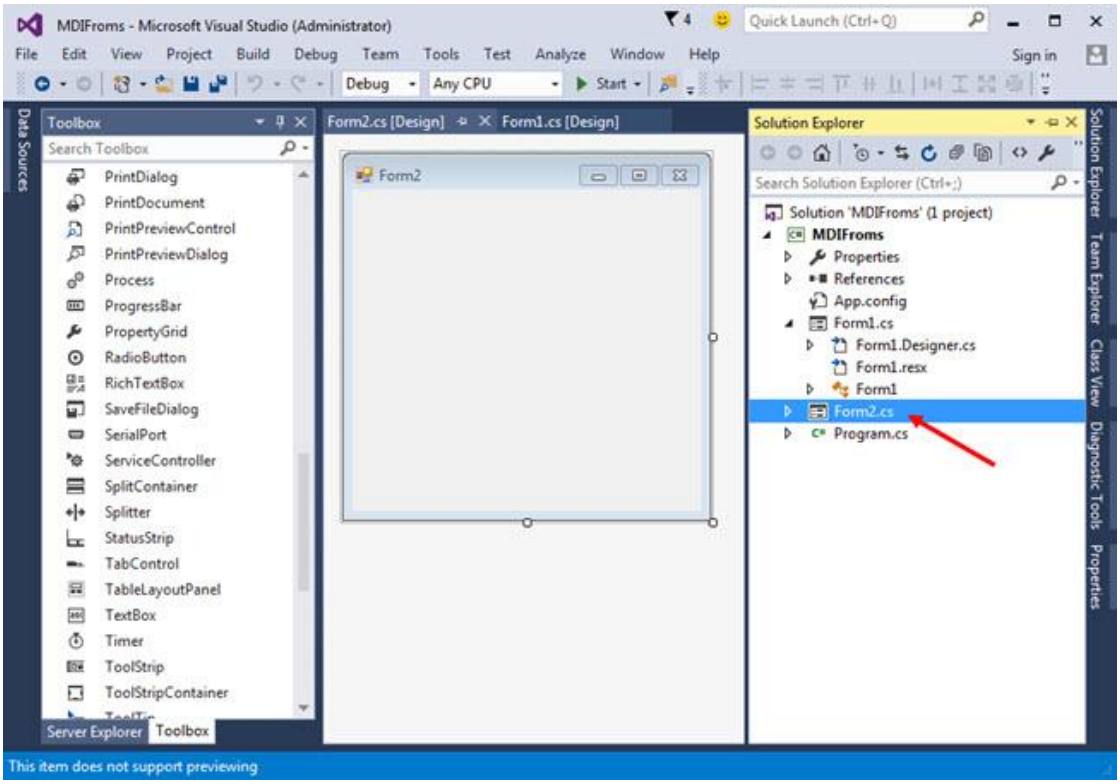
مشاهده می‌کنید که مانند شکل زیر، کادر Add New Item ظاهر می‌شود. نام پیش‌فرض آن (Form2.cs) را تغییر ندهید و بر روی دکمه Add کلیک کنید:



تا مانند شکل زیر فرم دوم به برنامه اضافه شود:



نام فرم جدید به `Solution Explorer` واقع در سمت راست برنامه اضافه می‌شود:



اضافه کردن فرم به برنامه راحت است. موضوع اصلی نمایش آن در برنامه است. برای نمایش فرم دوم باید به این نکته توجه داشته باشید که فرم‌ها، کلاس هستند. در نتیجه برای کار با کلاس `Form2` باید یک شئ از آن ایجاد کنید. بنابراین بر روی دکمه دوبار کلیک کرده تا وارد محیط کدنویسی شوید. برای ایجاد یک شیء از فرم دوم یک متغیر به صورت زیر تعریف می‌کنیم:

```
Form2 secondForm;
```

سپس یک شئ جدید ایجاد می‌کنید:

```
secondForm = new Form2();
```

پیشنهاد می‌کنیم که دو خط کد بالا را به صورت تک خط زیر بنویسید :

```
Form2 secondForm = new Form2();
```

کد بالا یک شیء جدید از کلاس Form2 با نام secondForm ایجاد می‌کند. برای نمایش فرم از متد Show() به صورت زیر استفاده می‌کنیم :

```
secondForm.Show();
```

کد نهایی باید به صورت زیر باشد :

```
private void btnFormTwo_Click(object sender, EventArgs e)
{
	Form2 secondForm = new Form2();
	secondForm.Show();
}
```

برنامه را اجرا و آن را امتحان می‌کنیم. بر روی دکمه کلیک می‌کنیم تا فرم دوم نمایش داده شود. یک مشکل کوچک وجود دارد. اگر بر روی دکمه دوباره کلیک کنید، فرم‌های دیگری ایجاد و نمایش داده می‌شوند. برای رفع این مشکل کد بالا را به صورت زیر تغییر دهید :

```
Form2 secondForm = new Form2();
private void btnFormTwo_Click(object sender, EventArgs e)
{
	secondForm.Show();
}
```

برنامه را دوباره اجرا و بر روی دکمه چندین بار کلیک کنید. در درس بعدی به فرم ایجاد شده قابلیت‌های بیشتری اضافه می‌کنیم.

فرم شرطی (Modal Form) در سی‌شارپ

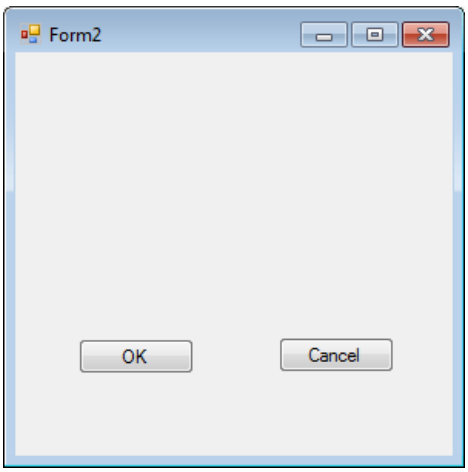
فرم شرطی به فرمی گفته می‌شود که بعد از نمایان شدن شما را مجبور به انجام کارهایی برای ادامه دادن به بقیه کارها می‌کند. فرض کنید به جای استفاده از متد Show() آن را به صورت زیر تغییر دهیم:

```
secondForm.ShowDialog();
```

متدی که الان از آن استفاده می‌کنیم، متد ShowDialog() می‌باشد. این متد یک فرم شرطی (Modal form) را به وجود می‌آورد. فرم شرطی به فرمی گفته می‌شود که بعد از نمایان شدن، شما را مجبور به انجام کارهایی برای ادامه دادن به بقیه کارها می‌کند. مثلاً در حالت قبل (هنگام استفاده از متد Show()) شما به راحتی می‌توانستید بر روی فرم اصلی برنامه کلیک کرده و آن را فعال کنید. اما در این حالت (هنگام استفاده از متد ShowDialog()) برای فعال کردن بر روی فرم اول شما مجبورید، فرم دوم را ببندید. حال برنامه را اجرا و بر روی دکمه کلیک کنید تا فرم دوم ظاهر شود. بعد از ظاهر شدن فرم آن را به گوشه‌ای بکشید و سعی کنید که بر روی دکمه دوباره کلیک کنید. مشاهده می‌کنید که دکمه غیر قابل کلیک است و تا فرم دوم را نبندید این کار امکان پذیر نیست. دو دکمه به فرم دوم اضافه کنید و خاصیت‌های Name و Text آنها را به صورت زیر تغییر دهید :

خاصیت	دکمه اول	دکمه دوم
Name	btnOK	btnCancel
Text	OK	Cancel

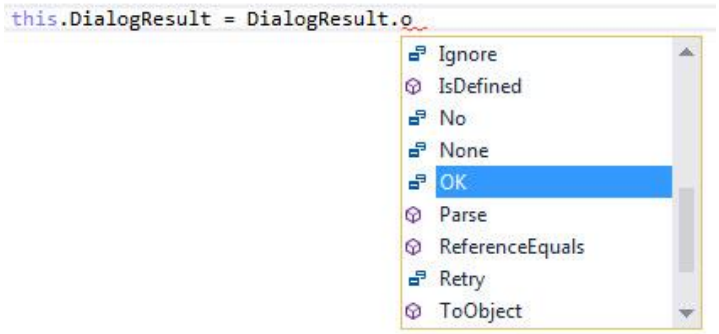
شکل نهایی فرم دوم به صورت زیر است:



بر روی دکمه OK کلیک کرده و کد زیر را بنویسید:

```
this.DialogResult = DialogResult.OK;
```

بعد از تایپ علامت مساوی، IntelliSense ظاهر می‌شود. گزینه DialogResult را انتخاب و علامت نقطه را تایپ کنید تاIntelliSense دوباره ظاهر شود:



از کادر باز شده گزینه OK را انتخاب کنید. حال بر روی دکمه Cancel کلیک کرده و کد زیر را به آن اضافه کنید:

```
this.DialogResult = DialogResult.Cancel;
```

کد نویسی فرم دوم به صورت زیر در می‌آید:

```
public Form2()
{
```

```
        InitializeComponent();
}

private void btnOK_Click(object sender, EventArgs e)
{
        this.DialogResult = DialogResult.OK;
}

private void btnCancel_Click(object sender, EventArgs e)
{
        this.DialogResult = DialogResult.Cancel;
}
```

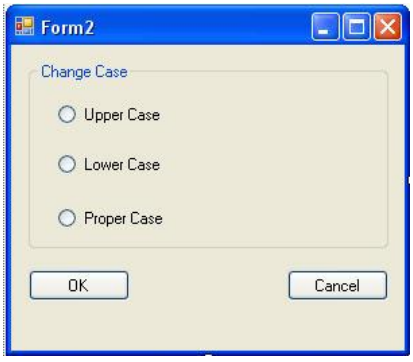
با استفاده از فرم اول می‌توانید بفهمید که کدام دکمه از فرم دوم کلیک شده است Cancel یا OK؟ بر روی دکمه فرم اول دو بار کلیک کرده و کدهای آن را به صورت زیر تغییر دهید:

```
if (secondForm.ShowDialog() == DialogResult.OK)
{
        MessageBox.Show("OK Button Clicked!");
}
```

کد بالا چک می‌کند که آیا دکمه OK کلیک شده است یا نه؟ اگر کلیک شده باشد پیغامی نمایش داده می‌شود. اما برای دکمهCancel مجبور نیستید کدی بنویسید، چون با زدن این دکمه فرم بسته می‌شود و همین کافی است. حال برنامه را اجرا و بر روی دکمه Change Case در فرم اول کلیک کرده تا فرم دوم ظاهر شود. بر روی دکمه OK کلیک کنید تا پیغام برای شما نمایش داده شود. همین کار را با دکمه Cancel انجام داده و نتیجه را مشاهده نمایید.

دریافت مقادیر از دیگر فرم‌ها

فرم دوم را به شکل زیر جهت تغییر در بزرگی و کوچگی حروف جعبه متن موجود در فرم اول تغییر دهید:



قصد ما این است که وقتی دکمه OK کلیک شد، متن موجود در جعبه متن فرم اول، بسته به گزینه انتخاب شده تغییر کند. مشکلی که با آن روبرو می‌شویم این است که، جعبه متن در فرم اول قرار دارد و نمی‌توانیم آن را ببینیم و اگر بخواهیم به آن در فرم دوم دسترسی داشته باشیم با خطا مواجه می‌شویم. راه حل این مشکل این است که جعبه متن را به صورت زیر تعریف کنیم (کدهای زیر را به فرم اول اضافه کنید):

```
public static TextBox tb = new TextBox();
```

کد بالا باعث ایجاد یک شئ TextBox به نام tb می‌شود. این کد را در پایین شئ ایجاد شده از فرم به صورت زیر قرار می‌دهیم:

```
public Form1()
{
       InitializeComponent();
}

Form2 secondForm = new Form2();
public static TextBox tb = new TextBox();

private void btnFormTwo_Click(object sender, EventArgs e)
{
       secondForm.Show();
}
```

به این نکته توجه کنید که ما کد مربوط به نمایش پیغام را حذف کردیم، چون به آن احتیاجی نداریم. حال یک شئ TextBox داریم که می‌توانیم جعبه متن موجود در فرم اول را به آن تخصیص دهیم. در رویداد Load فرم اول کد زیر را اضافه کنید:

```
tb = txtChangeCase;
```

کد نهایی فرم اول به صورت زیر می‌باشد:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    Form2 secondForm = new Form2();
    public static TextBox tb = new TextBox();

    private void btnFormTwo_Click(object sender, EventArgs e)
    {
        secondForm.Show();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        tb = txtChangeCase;
    }
}
```

هنگامی که فرم اول بارگذاری (Load) می‌شود، جعبه متن در فرم دوم قابل دسترسی می‌باشد. بر روی دکمه OK فرم دوم دو بار کلیک کرده و کد زیر را در داخل آن بنویسید:

```
string changeCase = Form1.tb.Text;
```

در کد بالا یک متغیر رشته با نام changeCase ایجاد کرده‌ایم. محتوای این رشته جدید متن موجود در جعبه متن فرم اول است. برای تغییر در بزرگی و کوچکی حروف می‌توانیم از دو متد Uppercase() و Lowercase() به صورت زیر استفاده کنیم:

```
changeCase = changeCase.ToUpper();
changeCase = changeCase.ToLower();
```

متاسفانه سی‌شارپ دارای متدی برای بزرگ کردن فقط حرف اول رشته نمی‌باشد. یعنی اگر بخواهیم رشته‌ای داشته باشیم که حرف اول هر کلمه آن به صورت بزرگ و بقیه حروف به صورت کوچک نمایش داده شوند (مانند This Is Proper Case)، متدی برای این کار وجود ندارد. برای این منظور باید دو فضای نام زیر را به قسمت فضاهای نامی اضافه کنیم:

```
using System.Globalization;
using System.Threading;
```

حال که این دو فضای نامی را اضافه کردیم، کار بعدی ایجاد یک شئ CultureInfo می‌باشد:

```
CultureInfo properCase = Thread.CurrentThread.CurrentCulture;
```

CurrentCulture به ما اطلاعاتی درباره زبان یک کشور خاص ارائه می‌دهد. یک شیء از CultureInfo با نام properCase ایجاد می‌کنیم. این پایان کار نیست. ما به یک شئ TextInfo هم احتیاج داریم و آن را textInfoObject می‌نامیم:

```
TextInfo textInfoObject = properCase.TextInfo;
```

این شئ دارای متدی با نام ()ToTitleCase می‌باشد که ما برای تغییر در بزرگی و کوچکی حروف یک رشته به آن احتیاج داریم.

```
changeCase = textInfoObject.ToTitleCase(changeCase);
```

مقدار changeCase در کد بالا همان متن موجود در فرم اول می‌باشد. برای فهمیدن اینکه کدام گزینه در فرم دوم انتخاب شده به یک سری دستور if…else احتیاج داریم:

```
if (radioButton1.Checked==true)
{
       changeCase = changeCase.ToUpper();
}
else if(radioButton2.Checked==true)
{
       changeCase = changeCase.ToLower();
}
else if (radioButton3.Checked==true)
{
       CultureInfo ProperCase=Thread.CurrentThread.CurrentCulture;
       TextInfo textInfoObject=ProperCase.TextInfo;

       changeCase = textInfoObject.ToTitleCase(changeCase);
}
```

در کدهای بالا ما فقط چک کرده‌ایم که کدام دکمه رادیویی انتخاب شده و سپس با توجه به آن عمل تبدیل حروف را انجام می‌دهیم. برای اضافه کردن متن تبدیل شده به جعبه متن فرم اول کد زیر را می‌نویسیم:

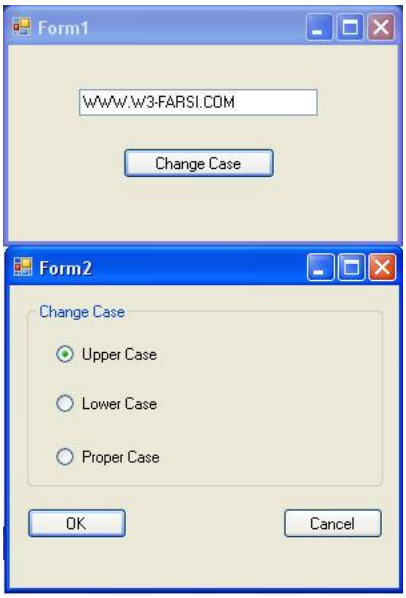
```
Form1.tb.Text = changeCase;
```

کد بالا را درست قبل از کد DialogResult می‌نویسیم. کد کامل دکمه باید به صورت زیر باشد:

```
private void btnOK_Click(object sender, EventArgs e)
{
       string changeCase = Form1.tb.Text;
```

```
        if (radioButton1.Checked==true)
        {
                changeCase = changeCase.ToUpper();
        }
        else if(radioButton2.Checked==true)
        {
                changeCase = changeCase.ToLower();
        }
        else if (radioButton3.Checked==true)
        {
                CultureInfo ProperCase=Thread.CurrentThread.CurrentCulture;
                TextInfo textInfoObject=ProperCase.TextInfo;

                changeCase = textInfoObject.ToTitleCase(changeCase);
        }
        Form1.tb.Text = changeCase;

        this.DialogResult = DialogResult.OK;
}
```

برنامه را اجرا و بر روی دکمه برای نمایش فرم دوم کلیک کنید. بعد از نمایان شدن فرم دوم گزینه Uppre Case را انتخاب کرده و بر روی دکمه OK کلیک کنید.

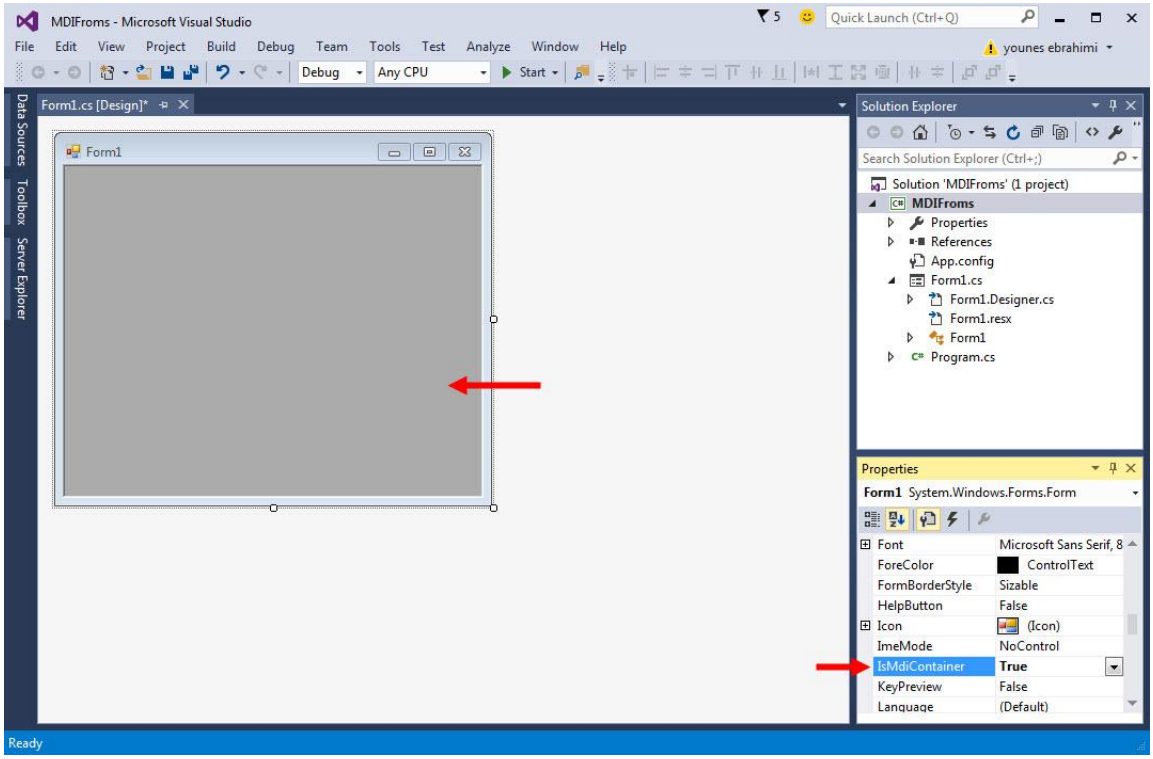


مشاهده می‌کنید که متن موجود در فرم اول به صورت بزرگ نمایش داده می‌شود.

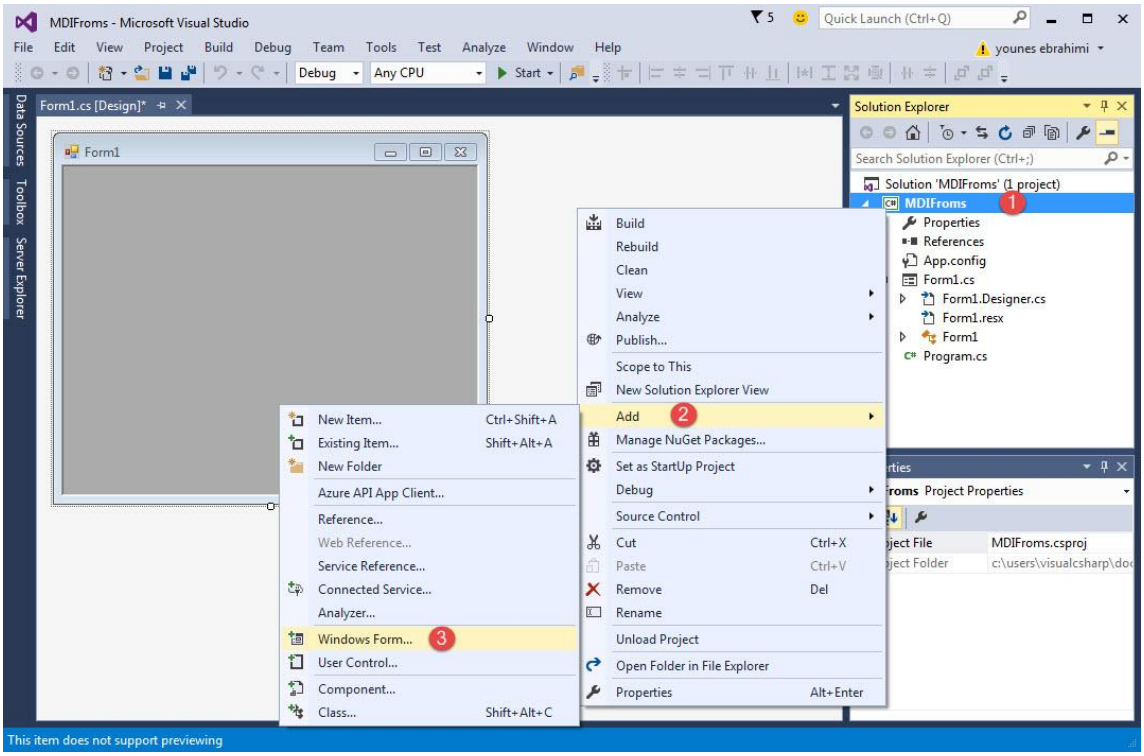
کار با فرم‌های MDI

MDI یا Multiple Document Interface نوعی طراحی رابط گرافیکی چند سندی است؛ که در آن کاربر قادر است از چند فرم (فرزند) در داخل یک فرم (پدر) استفاده کند. مثالی از یک رابط چند سندی نرم‌افزار مایکروسافت Word یا Excel هست. وقتیExcel رو اجرا می‌کنید یک پنجره والد ظاهر می‌شود که در این پنجره پدر می‌توانید هر تعداد سند بعنوان پنجره‌های فرزند ظاهر می‌گردند را باز کنید. در یک برنامه MDI، تمام پنجره‌های فرزند از یک نوار ابزار و یا نوار منو که در پنجره والد ظاهر می‌گردد، بصورت مشترک استفاده می‌کنند. یکی از محدودیت‌های پنجره‌های فرزند اینست که می‌توانند تنها در محدوده‌ی پنجره والد وجود داشته باشند. اجازه دهید که نحوه کار با فرم‌های چند سندی را با یک مثال توضیح

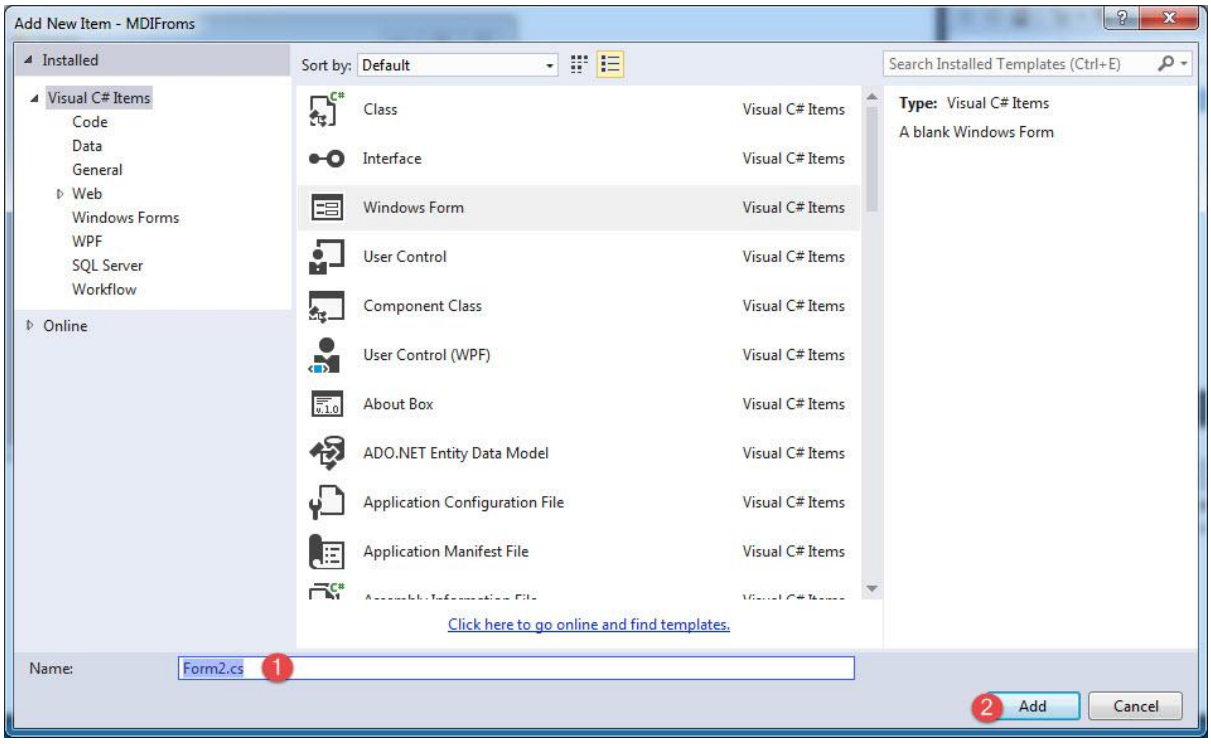
دهیم. یک برنامه ویندوزی ایجاد کرده و نام آن را `MDIFroms` بگذارید. بعد از بالا آمدن برنامه خاصیت `IsMdiContainer` فرم را به `true` تغییر می‌دهید. مشاهده می‌کنید که رنگ پس زمینه فرم به رنگ طوسی پر رنگ در می‌آید :



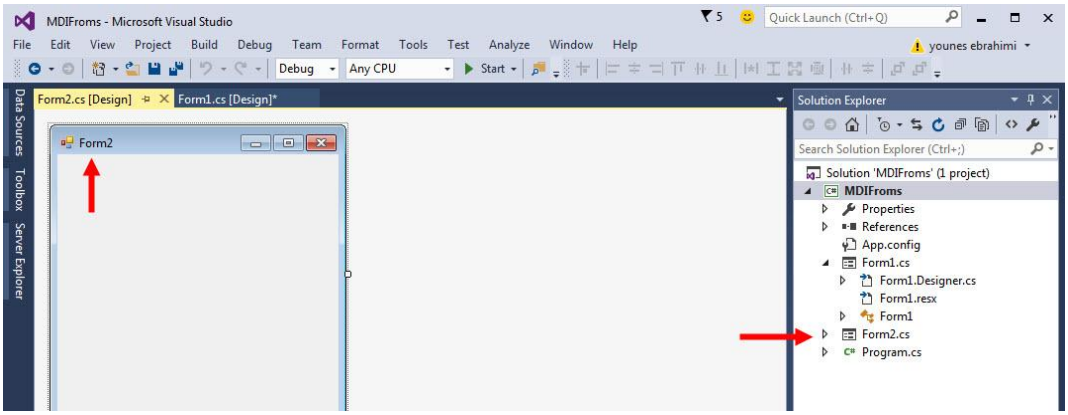
با این کار فرم پدر ایجاد می‌شود. حال نوبت به اضافه کردن فرم فرزند می‌رسد. برای اضافه کردن فرم دوم یا فرم فرزند به برنامه به صورت زیر عمل می‌کنیم :



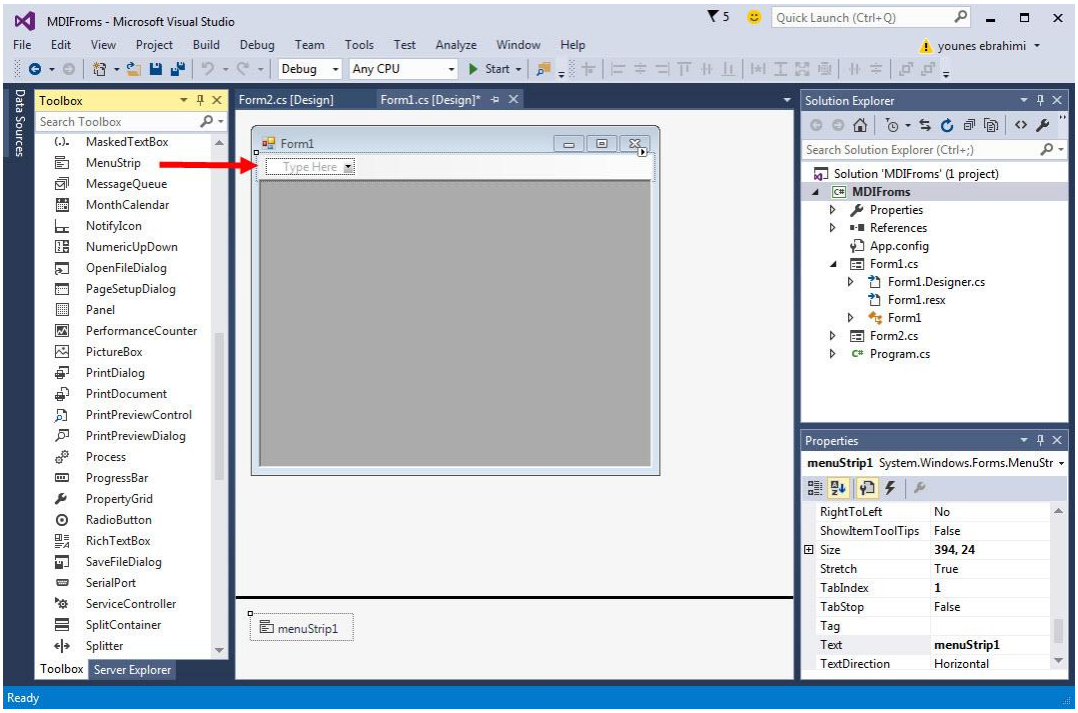
نام فرم را هم در حالت پیش‌فرض رها کرده و دکمه Add را می‌زنیم:

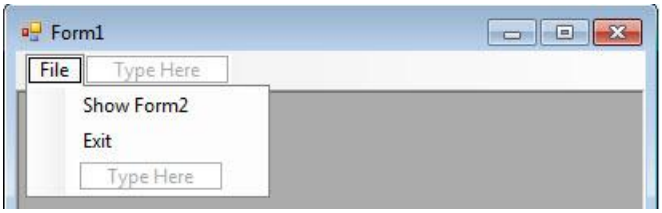


همانطور که در شکل زیر مشاهده می‌کنید، یک فرم به نام Form2 به برنامه اضافه می‌شود:



حال یک منو با استفاده از کنترل MenuStrip به فرم اول یا From1 و چند زیر منو هم به منو اضافه می‌کنیم:

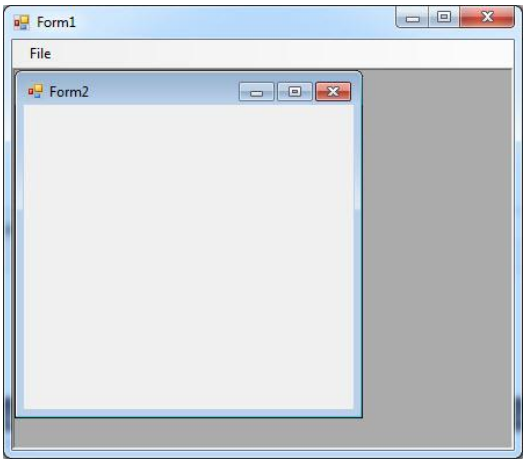




حال بر روی زیر منوی Show Form1 دوبار کلیک کرده و کد زیر را در رویداد کلیک آن می‌نویسیم:

```
private void showForm2ToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 frm2 = new Form2();
    frm2.Show();
    frm2.MdiParent = this;
}
```

به کلمه کلیدی this در کد بالا توجه کنید. چون این کلمه را در داخل فرم اول نوشته‌ایم در نتیجه به همین فرم اشاره می‌کند. و معنای آن این است که فرم والد frm2 همین فرم است. برنامه را اجرا کرده و بر روی زیر منوی Show Form2 کلیک کنید:



همانطور که در شکل بالا مشاهده می‌کنید، فرم دوم در داخل فرم اول نمایش داده و نسبت به گوشه بالا سمت چپ تراز می‌شود. اگر دوباره بر روی گزینه Show Form2 کلیک کنید، مشاهده می‌کنید که یک بار دیگر فرم دوم در داخل فرم اول به نمایش در می‌آید، برای جلوگیری از این کار می‌توانید کد بالا را به صورت زیر تغییر دهید:

```
private void showForm2ToolStripMenuItem_Click(object sender, EventArgs e)
{
    foreach (Form form in Application.OpenForms)
    {
        if (form.GetType() == typeof(Form2))
        {
            form.Activate();
            return;
        }
    }

    Form2 frm2 = new Form2();
    frm2.Show();
    frm2.MdiParent = this;
}
```

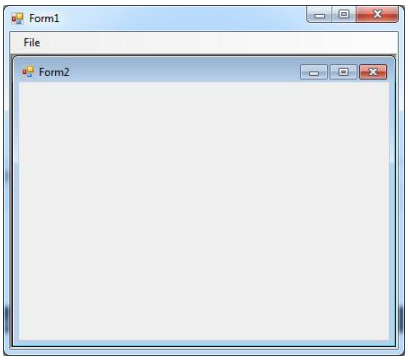
حال اگر بخواهید فرم دوم در وسط فرم اول نمایش داده شود می‌توانید کد زیر را هم به انتهای کدهای بالا اضافه کنید:

```
frm2.StartPosition = FormStartPosition.CenterScreen;
```

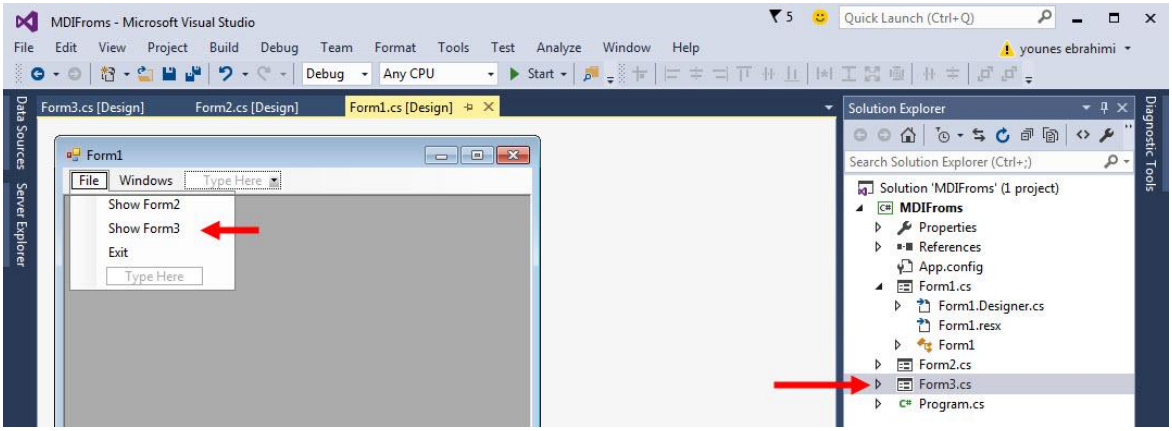
البته به جای CenterScreen می‌توان مقادیر CenterParent، Manual، WindowsDefaultBounds و WindowsDefaultLocation را نوشت و نحوه عملکرد آنها را امتحان کرد. حال اگر بخواهید فرم فرزند کل فرم اول را در بر بگیرد و با تغییر سایز فرم پدر او نیز تغییر اندازه دهد، باید از کد زیر استفاده کنید:

```
frm2.Dock = DockStyle.Fill;
```

حال برنامه را اجرا و نتیجه را مشاهده کنید:



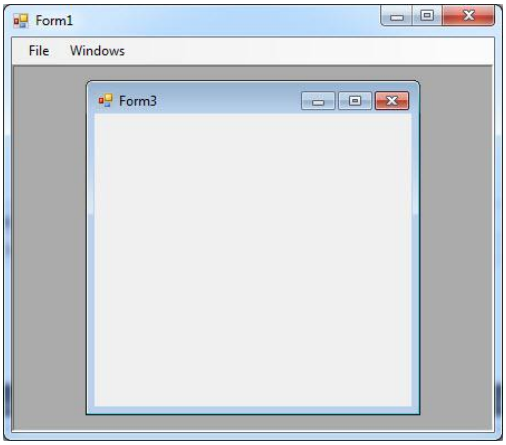
البته این خط بالا را در برنامه فقط برای تست بنویسید و دوباره پاک کنید. اگر بخواهید از چند فرم در داخل فرم پدر استفاده کنید، وضع به همین منوال است. یک فرم دیگر به برنامه بالا و یک زیر منو هم به صورت زیر به منو اضافه کنید:



بر روی زیرمنوی جدید دوبار کلیک کرده و کدهای زیر را در داخل آن بنویسید:

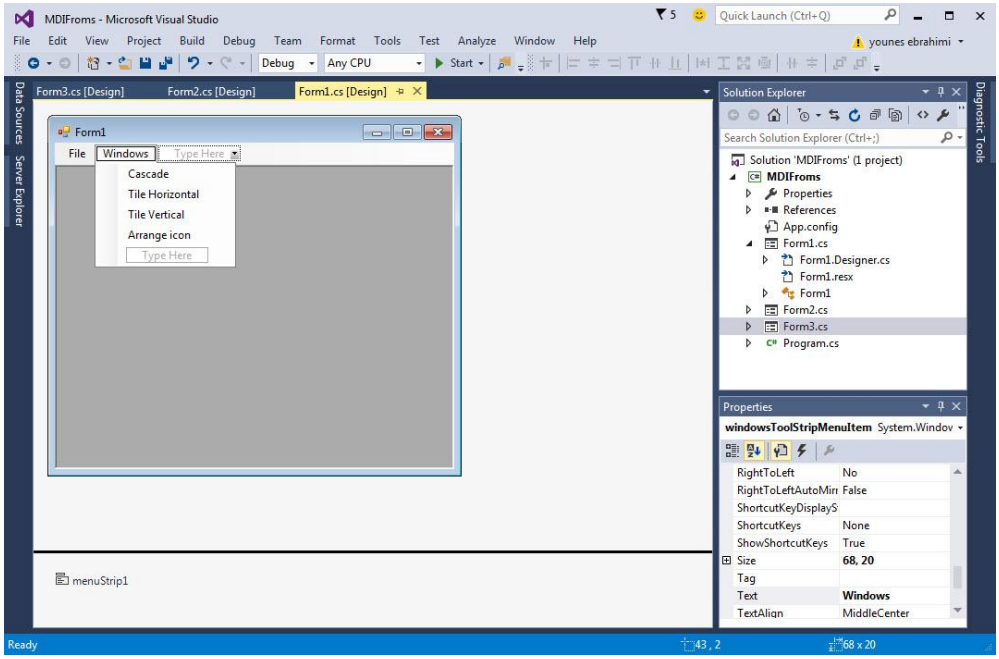
```
private void showForm3ToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form3 frm3 = new Form3();
    frm3.StartPosition = FormStartPosition.CenterScreen;
    frm3.Show();
    frm3.MdiParent = this;

}
```

برنامه را اجرا و هر دو فرم را باز کنید



:

مشاهده می‌کنید که هر دو فرم بر روی هم قرار می‌گیرند چون خاصیت StartPosition بر روی CenterScreen است. حال اگر بخواهید آنها را به صورت مرتب تری در داخل فرم اول نمایش دهید باید چکار کنید؟ یک منو با چند زیر منوی دیگر به فرم اضافه کنید:

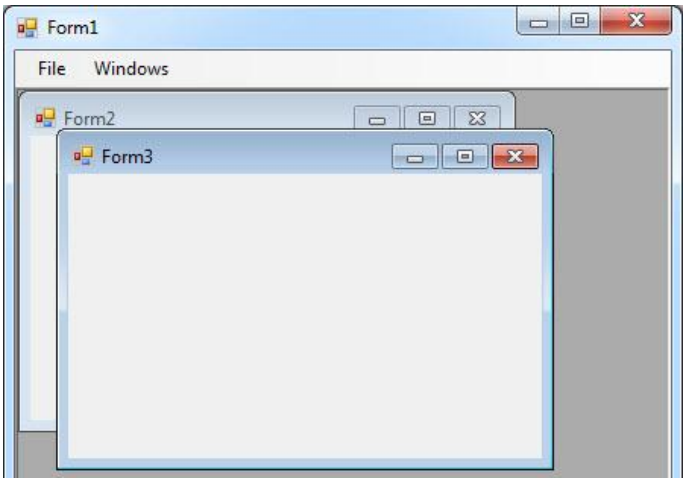


به ترتیب بر روی زیر منوهای بالا کلیک کرده و کدهای زیر را در داخل آنها بنویسید:

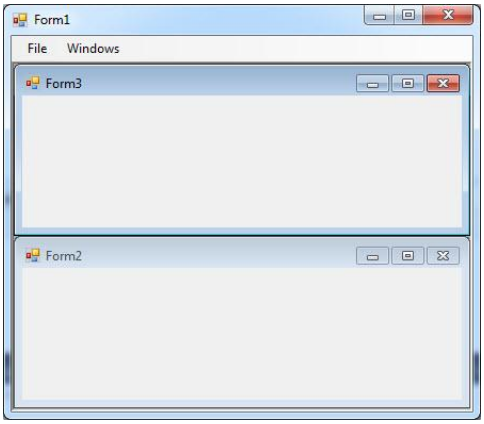
```
private void cascadeToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.Cascade);
}

private void tileHorizontalToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.TileHorizontal);
}

private void tileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.TileVertical);
```

```
}

private void arrangeIconToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.ArrangeIcons);
}
```

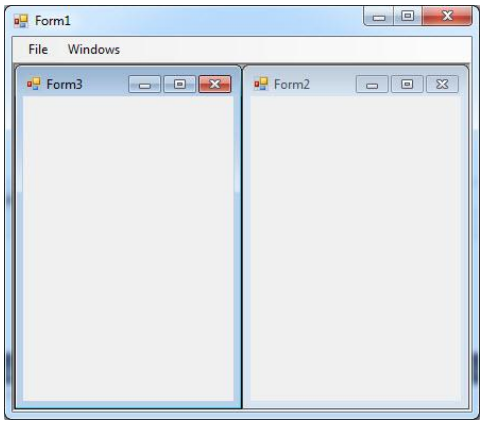
اجرای برنامه و کلیک بر روی زیر منوهای بالا به صورت زیر است:



Cascade

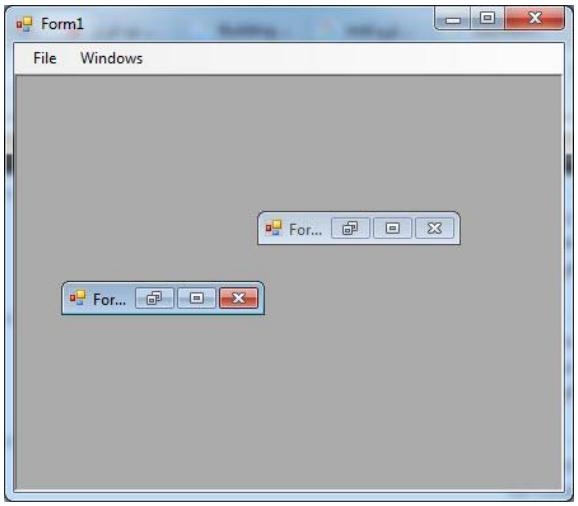


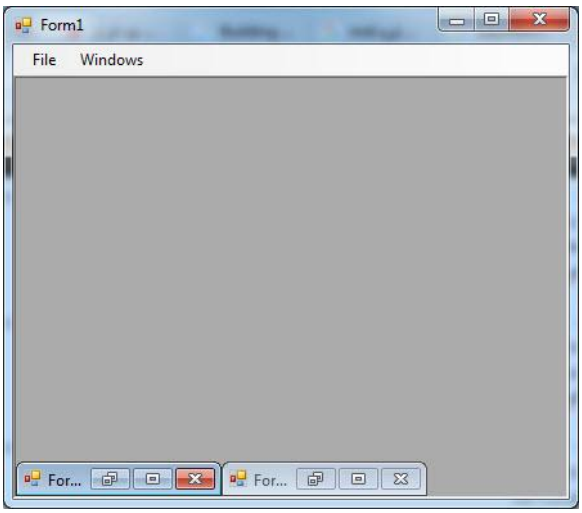
Horizontal



Vertical

به این نکته توجه کنید که گزینه Arrange icon فقط در صورتیکه فرم‌های فرزند Minimize باشند، کار می‌کند. به این صورت که اگر فرم‌ها را Minimizeکرده و ترتیب آنها را بر هم بزنید این گزینه آنها را مرتب می‌کند:





فصل سوم

دات نت فریم ورک

کلاس System.DateTime

کلاس System.DateTime کلاسی از دات‌نت است که به شما اجازه استفاده، ذخیره، و دستکاری ساعت و تاریخ را می‌دهد. این کلاس دارای متدهایی برای دستکاری تاریخ مانند اضافه و کم کردن روزها، ماه‌ها، یا سال‌ها و مطابق کردن آنها با تاریخ جاری را می‌دهند. همچنین دارای متدهایی است که تاریخ را به اشکال متفاوتی نشان می‌دهند. کد زیر نحوه استفاده از کلاس DateTime را نشان می‌دهد.

```
using System;

public class Program
{
    public static void Main()
    {
        DateTime newyear = new DateTime(2010, 12, 25);

        Console.WriteLine(newyear.ToString());
    }
}
```

```
12/25/2010 12:00:00 AM
```

از یک سازنده که سال و ماه و روز را قبول می‌کند برای مقدار دهی به شیء DateTime و همچنین از متد ()ToString کلاس DateTime برای نمایش تاریخ و ساعت استفاده می‌کنیم. متد ()ToString در حالت پیش‌فرض شیء DateTime شما را به فرمت زیر نمایش می‌دهد: MM/DD/YYYY HH:MM:SS AM/PM. با متدهای مختلفی که فرمت خروجی را با فرمت‌های دیگری نمایش می‌دهند، در آینده آشنا می‌شوید. کلاس DateTime دارای چندین سازنده است که می‌توانید از آنها استفاده کنید. جدول زیر لیست سازنده‌های متداول این کلاس را نشان می‌دهد.

سازنده
DateTime()
DateTime(int year, int month, int day)
DateTime(int year, int month, int day, int hour, int minute, int second)
DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond)

سازنده پیشفرض، تاریخ را به صورت 01/01/0001 12:00:00 AM نشان می‌دهد. برای به دست آوردن تاریخ و ساعت جاری، می‌توان از خاصیت استاتیک Now کلاس DateTime استفاده کرد. برای به دست آوردن فقط تاریخ جاری می‌توان از خاصیت Today استفاده کرد.

```
DateTime now = DateTime.Now;
DateTime today = DateTime.Today;
```

برای نمایش شیء DateTime به صورت‌های دیگر می‌توان از چندین متد که در زیر نشان داده شده‌اند، استفاده کرد:

متد	نمونه خروجی
ToLongDateString	Saturday, Octoboer 2, 2010
ToLongTimeString	4:14:18 PM

ToShortDateString	10/2/2010
ToShortTimeString	4:14 PM

حتی می‌توان از متد String.Format() برای سفارشی کردن خروجی‌ها استفاده کرد.

Specifier	نوع	مثال	خروجی
dd	Day	{0:dd}	10
ddd	Day name	{0:ddd}	Tue
dddd	Full Day Name	{0:dddd}	Tuesday
f, ff, …	Second Fractions	{0:fff}	932
gg, …	Era	{0:gg}	A.D.
hh	2 digit hour	{0:hh}	10
HH	2 digit hour, 24hr format	{0:HH}	22
mm	Minute 00-59	{0:mm}	38
MM	Month 01-12	{0:MM}	12
MMM	Month abbreviation	{0:MMM}	Dec
MMMM	Full month name	{0:MMMM}	December
ss	Seconds 00-59	{0:ss}	46
tt	AM or PM	{0:tt}	PM
yy	Year, 2 digits	{0:yy}	02
yyyy	Year	{0:yyyy}	2002
zz	Timezone offset, 2 digits	{0:zz}	-05
zzz	Full timezone offset	{0:zzz}	-05:00
:	Seperator	{0:hh:mm:ss}	10:43:20
/	Seperator	{0:dd/MM/yyyy}	10/2/2010

به عنوان مثال:

```
Console.WriteLine("The current date is {0:dd/MM/yyy}", DateTime.Now);
```

```
The current date is 10/2/2010
```

کلاس DateTime متدهای زیادی برای کار کردن با ساعت و تاریخ در اختیار کاربر قرار می‌دهد. برخی از این متدها در زیر نشان داده شده است:

متد	توضیح
AddDays	به وسیله این متدها می‌توان مقادیری را به شیء DateTime اضافه و یا مقادیری را از آن کم کرد.
AddHours	
AddMilliseconds	
AddMinutes	
AddMonths	
AddSeconds	
AddTicks	
AddYears	
Equals	نشان می‌دهد که آیا شیء DateTime با آرگومان DateTime ارسال شده به متد برابر است یا نه
ToUniversalTime	مقدار نمونه DateTime را به ساعت جهانی تبدیل می‌کند.
DaysInMonth	Static. تعداد روزهای یک ماه را بر می‌کرداند.
IsLeapYear	Static. نشان می‌دهد که آیا سال مشخص شده یک سال کبیسه است یا نه

در جدول زیر لیست برخی از خواص (proerty) کلاس DateTime نشان داده شده است:

خاصیت	توضیحات
Date	تاریخ را بر می‌گرداند.
Day	نشان می‌دهد که چندمین روز ماه جاری است.
DayOfWeek	نشان می‌دهد که چه روزی از هفته است (مثلاً شنبه یا ...).
DayOfYear	نشان می‌دهد که چندمین روز سال جاری است (مثلاً ۱۲۵ مین روز).
Hour	ساعت را نشان می‌دهد.
Millisecond	میلی ثانیه را نشان می‌دهد.
Minute	دقیقه را نشان می‌دهد.
Month	نشان می‌دهد که چندمین ماه سال جاری است
Second	ثانیه را نشان می‌دهد.

TimeOfDay	نشان می‌دهد که چه ساعتی از روز است.
Year	برای نشان دادن سال به کار می‌رود.
Now	Static. تاریخ و ساعت جاری را نشان می‌دهد.
Today	Static. فقط تاریخ جاری را نشان می‌دهد.
UtcNow	Static. تارخ و ساعت جاری محلی را نشان می‌دهد.

در زیر نحوه استفاده از متدها و خواص کلاس DateTime آمده است:

```
using System;

public class Program
{
    public static void Main()
    {
        DateTime myDate = DateTime.Now;

        Console.WriteLine("Year: " + myDate.Year);
        Console.WriteLine("Month: " + myDate.Month);
        Console.WriteLine("Day: " + myDate.Day);
        Console.WriteLine("Today is {0}.", myDate.DayOfWeek.ToString());

        //Assign newDate with the current date added by 3 days
        DateTime newDate = myDate.AddDays(3);
        Console.WriteLine("The date 3 days from now is {0}.",
                  newDate.ToShortDateString());

        //Assign newdate with the current date subtracted by 3 days
        newDate = myDate.AddDays(-3);
        Console.WriteLine("The date 3 days ago is {0}.", newDate.ToShortDateString());
    }
}
```

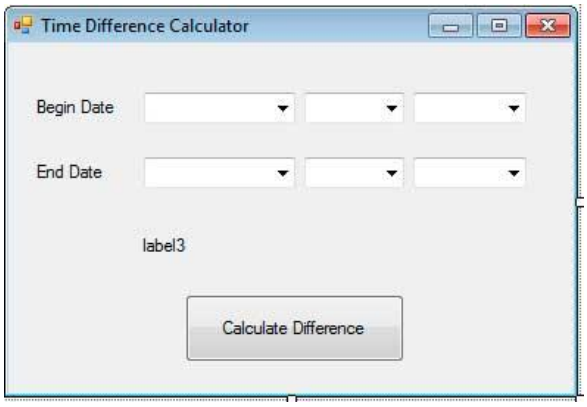
```
Year: 2010
Month: 10
Day: 3
Today is Sunday.
The date 3 days from now is 10/6/2010.
The date 3 days ago is 9/30/2010.
```

همانطور که مشاهده می‌کنید از خواص کلاس DateTime برای نمایش جداگانه اجزاء تاریخ استفاده کرده‌ایم. در مثال بالا نشان داده شده است که چطور به وسیله کم یا زیاد کردن روزها می‌توان مقدار شیء DateTime را اصلاح کرد. مثلاً برای کم کردن ۳ روز از شیء، از یک عدد منفی استفاده کرده‌ایم. همچنین متد ToShortDateString() برای نمایش شکل کوتاه تاریخ به کار می‌رود.

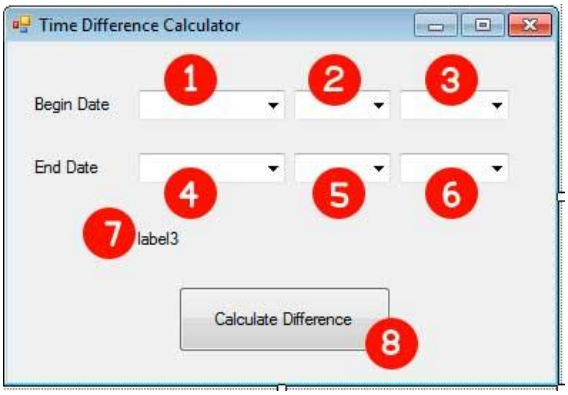
محاسبه اختلاف دو تاریخ

در این بخش به شما نحوه ایجاد یک ماشین حساب که اختلاف بین دو تاریخ را محاسبه می‌کند آموزش داده می‌شود. برنامه از متدهای System.DateTime و System.TimeSpan استفاده می‌کند این آموزش همچنین مهارت شما را در استفاده از رویدادها و کنترل

ComboBox افزایش می‌دهد. یک برنامه ویندوزی ایجاد کنید و نام آن را TimeDifferenceCalculator بگذارید. سپس مانند شکل زیر کنترل‌های لازم را بر روی آن قرار دهید و مرتب کنید.



به صورت اختیاری می‌توانید خاصیت Text فرم را تغییر داده و همچنین خاصیت FormBorderStyle را به FixedSingle تغییر دهید تا اندازه فرم ثابت بماند. خاصیت Name کنترل‌های شماره گذاری شده را مطابق جدول زیر تغییر دهید.



شماره	نام
۱	comboBeginDateMonth
۲	comboBeginDateDay
۳	comboBeginDateYear
۴	comboEndDateMonth
۵	comboEndDateDay
۶	comboEndDateYear
۷	labelResult

۸	buttonCalculate

خاصیت Text کنترل labelResult را پاک کنید تا نتیجه محاسبه در آن قرار بگیرد. بر روی فرم (نه کنترل‌ها) دو بار کلیک کنید تا یک کنترل کننده رویداد برای خاصیت Load فرم ایجاد شود. سپس کد زیر را به آن اضافه کنید.

```
private void Form1_Load(object sender, EventArgs e)
{
    //تعریف ماه‌ها
    string[] months =
    {
        "January", "February", "March", "April",
        "May", "June", "July", "August",
        "September", "October", "November", "December"
    };

    //پر کردن کمبوباکس‌ها با ماه‌ها
    for (int i = 0; i < 12; i++)
    {
        comboBeginDateMonth.Items.Add(months[i]);
        comboEndDateMonth.Items.Add(months[i]);
    }

    //پیشفرض قرار دادن ماه ژانویه
    comboBeginDateMonth.SelectedIndex = 0;
    comboEndDateMonth.SelectedIndex = 0;

    //پر کردن کمبوباکس‌ها با سال‌ها
    for (int i = 1; i <= DateTime.Now.Year; i++)
    {
        comboBeginDateYear.Items.Add(String.Format("{0:0000}", i));
        comboEndDateYear.Items.Add(String.Format("{0:0000}", i));
    }

    //پیش فرض قرار دادن سال ۲۰۰۰
    comboBeginDateYear.SelectedIndex = 1999;
    comboEndDateYear.SelectedIndex = 1999;

   //پر کردن کمبوباکس‌ها با روزها
    FillDays(comboBeginDateDay, "Begin");
    FillDays(comboEndDateDay, "End");

    //اضافه کردن کنترل کننده رویداد
    comboBeginDateMonth.SelectedIndexChanged += new EventHandler(RecalculateDays);
    comboBeginDateYear.SelectedIndexChanged += new EventHandler(RecalculateDays);
    comboEndDateMonth.SelectedIndexChanged += new EventHandler(RecalculateDays);
    comboEndDateYear.SelectedIndexChanged += new EventHandler(RecalculateDays);
}
```

اولین چیزی که ما ایجاد می‌کنیم آرایه ای رشته‌ای است که ۱۲ ماه سال را در خود جای می‌دهد. سپس با استفاده از حلقه for ماه‌ها را به دو کمبوباکس (combo boxes) برای نشان دادن آنها (ماه‌ها) اضافه می‌کنیم. به وسیله متد ()Add خاصیت Item کلاس ComboBox این کار را انجام می‌دهیم. سپس خاصیت SelectedIndex دو کمبوباکس را به عدد صفر تغییر می‌دهیم که نشان دهنده اولین آیتم یعنی ماه ژانویه است.

نکته مهم این است که چون اندیس‌ها در کمبو باکس‌ها از صفر شروع می‌شود در نتیجه شمارش نیز از صفر آغاز می‌شود. سپس کمبو باکس‌هایی که قرار است سال‌ها را نمایش دهند را از عدد ۱ تا سال جاری پر می‌کنیم. برای دسترسی به سال جاری از خاصیت Year خاصیت Now کلاس DateTime استفاده می‌کنیم. خاصیت Now نشان دهنده ساعت و تاریخ جاری است.

همه سال‌ها را برای نمایش به کمبوباکس‌ها اضافه می‌کنیم. برای قالب بندی سال‌ها از ۴ صفر استفاده می‌کنیم بنابراین همه سال‌ها ۴ رقمی نشان داده می‌شوند. با اختصاص مقدار ۱۹۹۹ به خاصیت SelectedIndex دو کمبوباکس مسئول نمایش سال، سال ۲۰۰۰ را سال مبنا قرار می‌دهیم. به خاطر داشته باشید که SelectedIndex از صفر شروع می‌شود در نتیجه برای نمایش سال ۲۰۰۰ باید اندیس آن ۱۹۹۹ باشد.

به این نکته توجه کنید که می‌توانیم به قسمت طراحی فرم رفته و بر روی کمبوباکس مورد نظر کلیک کنیم سپس خاصیت Items را پیدا کنیم. با کلیک بر روی دکمه مقابل این خاصیت که دارای سه نقطه است، صفحه‌ای باز می‌شود که بوسیله آن می‌توانیم هر آیتمی به کمبوباکس اضافه کنیم.

اما این کار قطعاً برای شما خسته کننده است چون قرار است که ۲۰۰۰ یا تعداد بیشتری مقدار را به آن اضافه کنیم. برای راحتی کار می‌توانیم از طریق کدنویسی این کار را انجام دهیم.

متد FillDays() که قبلاً ایجاد کرده‌ایم کمبوباکس‌ها را برای نمایش روز پر می‌کند ولی تعداد روزها در کمبو باکس‌ها برابر است مثلاً کمبوباکس‌های مسئول نشان دادن روزها همگی دارای ۳۰ روز می‌باشند، در اینجا به مشکل بر می‌خوریم چون بعضی از ماه‌ها ۳۱ روزه هستند و سال کبیسه هم وجود دارد.

در ۴ خط آخر کد یک کنترل کننده رویداد به رویداد SelectedIndexChanged هر کمبوباکس اضافه کرده‌ایم. این رویداد زمانی رخ می‌دهد که یک اندیس انتخاب شود و یا یک آیتم تغییر کند. یک کنترل کننده رویداد ایجاد کرده‌ایم که بین ۴ کنترل تقسیم می‌شود.

لازم است که این کنترل کننده رویداد را به کمبوباکس هایی که مسئول نگهداری ماه‌ها و سال‌ها هستند اضافه کنیم، بنابراین زمانی که کاربر یک سال یا ماه را تغییر می‌دهد، آیتم‌های داخل کمبو باکس‌های مسئول نگهداری روز به طور خودکار به روز (آپدیت) می‌شوند. کنترل کننده‌های رویداد را در آخر کدها قرار می‌دهیم، چون ما خاصیت SelectedIndex کمبوباکس‌ها را تغییر می‌دهیم که این کار باعث آزاد شدن رویداد و اجرای کنترل کننده‌های رویدادی می‌شود که باعث خالی شدن کمبوباکس‌ها می‌شوند. این کار باعث به وجود آمدن یک خطا می‌شود. اجازه دهید که یک متد FillDays() ایجاد کنیم. این متد باعث پر شدن روزهای یک کمبوباکس مشخص می‌شود. این متد یک کمبوباکس را به عنوان پارامتر آرگومان قبول می‌کند و دومین پارامتر آن یک رشته‌ای است که نشان می‌دهد کدام یک از کمبوباکس‌های comboBeginDateDay یا comboEndDataDay به روز شده‌اند.

```
private void FillDays(ComboBox comboBox, string tag)
{
    int selectedMonth;
    int selectedYear;

    if (tag == "Begin")
    {
        selectedMonth = comboBeginDateMonth.SelectedIndex + 1;
        selectedYear = comboBeginDateYear.SelectedIndex + 1;
    }
```

```
    else
    {
        selectedMonth = comboEndDateMonth.SelectedIndex + 1;
        selectedYear = comboEndDateYear.SelectedIndex + 1;
    }


    int maxDay = DateTime.DaysInMonth(selectedYear, selectedMonth);

    comboBox.Items.Clear();

    for (int i = 1; i <= maxDay; i++)
    {
        comboBox.Items.Add(i);
    }

    comboBox.SelectedIndex = 0;
}
```

ابتدا با استفاده از پارامتر tag مشخص می‌کنیم کدام یک از کمبوباکس‌های month و year مقادیر را دریافت می‌کنند. سپس خاصیت SelectedIndex آنها را با عدد ۱ جمع کرده و برابر متغیرهایی قرار می‌دهیم که به وسیله متد DaysInMonth() مورد استفاده قرار می‌گیرند. سپس از متد استاتیک DaysInMonth() کلاس DateTime برای چک کردن روزهای ماه‌های سال‌های خاص (کبیسه) استفاده می‌کنیم. برای آپدیت کردن کمبوباکس ابتدا آیتم‌ها را حذف می‌کنیم. تعداد روزها را با خواندن روزها با استفاده از یک حلقه for بروز می‌کنیم. سر انجام مقدار خاصیت SelectedIndex ازکامبو باکس را برابر ۰ قرار می‌دهیم تا مقدار پیش‌فرض آن برابر ۱ شود. سازنده Form1 را پیدا کرده و کد پر رنگ شده زیر را به آن اضافه کنید.

```
public Form1()
{
    InitializeComponent();

    comboBeginDateMonth.Tag = "Begin";
    comboBeginDateYear.Tag = "Begin";
    comboEndDateMonth.Tag = "End";
    comboEndDateYear.Tag = "End";
}
```

سپس آن مقادیر را به خصوصیت Tag نسبت می‌دهیم تا کامبوباکس درست را برای بروزرسانی به متد RecalculateDays() ارسال کنیم.

```
private void RecalculateDays(object sender, EventArgs e)
{
    if ((sender as ComboBox).Tag.ToString() == "Begin")
        FillDays(comboBeginDateDay, "Begin");
    else
        FillDays(comboEndDateDay, "End");
}
```

به این نکته توجه نمایید که این متد به عنوان کنترل کننده رویداد برای رویداد SelectedIndexChanged انتخاب شده است، نماینده این رویداد متدی را می‌پذیرد که ۲ پارامتر از نوع object و EventArgs داشته باشد.

متد شامل یک دستور if است که کنترل فعال کننده متد را به نوع کامبو باکس تبدیل می‌کند تا از خصوصیت Tag آن استفاده کند. ما نتیجه را با رشته "Begin" مقایسه می‌کنیم بنابراین برنامه تشخیص می‌دهد که کمبوباکس به روز شده است. به قسمت طراحی برمی گردیم و بر روی buttonCalculate دو بار کلیک می‌کنیم تا یک کنترل کننده رویداد برای رویداد کلیک ایجاد شود. کد زیر را اضافه می‌کنیم.

```
private void buttonCalculate_Click(object sender, EventArgs e)
{
    DateTime beginDate = new DateTime(
        comboBeginDateYear.SelectedIndex + 1,
        comboBeginDateMonth.SelectedIndex + 1,
        comboBeginDateDay.SelectedIndex + 1);

    DateTime endDate = new DateTime(
        comboEndDateYear.SelectedIndex + 1,
        comboEndDateMonth.SelectedIndex + 1,
        comboEndDateDay.SelectedIndex + 1);

    TimeSpan difference = endDate.Subtract(beginDate);

    labelResult.Text = String.Format("{0} Days", difference.Days);
}
```

بر روی دکمه کلیک کرده تا تاریخ‌های شروع و پایان به درستی در کامبوباکس‌ها نمایش داده می‌شوند. سپس تفاوت را با استفاده از متد Subtract() محاسبه می‌کنیم. این متد یک شیء از TimeSpan را برگشت می‌دهد. در آخر هم تفاوت دو تاریخ را در کنترل labelResult نمایش می‌دهیم.

برنامه قابلیت بهبود بیشتری را هم دارا می‌باشد. به عنوان مثال شما می‌توانید تعداد بیشتری کمبو باکس جهت یافتن اختلاف ساعت، دقیقه و ثانیه به برنامه اضافه کنید. همچنین می‌توانید تفاوت بین ماه‌ها و سال‌ها را نیز محاسبه کنید. از متد IsLeapYear() هم می‌توان برای تشخیص سال‌های کبیسه استفاده نمود.

کلاس System.Math

از کلاس System.Math برای انجام محاسبات ریاضی استفاده می‌شود. از این کلاس می‌توان برای گرد کردن اعداد، گرفتن جذر یا نتیجه توان یک عدد استفاده کرد. این کلاس یک کلاس استاتیک است و شما نمی‌توانید از آن نمونه ایجاد کرده و از متدهای آن استفاده نمایید. برای روشن شدن مطلب برخی از متدهای این کلاس را توضیح می‌دهیم.

گرد کردن اعداد

می‌توان با استفاده از Math.Ceiling() و Math.Floor() یک عدد با قسمت اعشار را گرد کرد. متد Math.Ceiling() یک عدد از نوع double را گرفته و یک مقدار از نوع double گرد شده را بر می‌گرداند. نتیجه این متد بزرگ‌تر یا مساوی آرگومان دریافت شده است. Math.Floor() یک عدد double را گرد کرده و نتیجه کوچک‌تر یا مساوی آرگومان گرفته شده را بر می‌گرداند. برای روشن شدن مطلب به مثال زیر توجه کنید:

```
double number = 34.567;
double ceil = Math.Ceiling(number);
double floor = Math.Floor(number);

Console.WriteLine("Math.Ceiling({0}) = {1}", number, ceil);
Console.WriteLine("Math.Floor({0}) = {1}", number, floor);
```

```
Math.Ceiling(34.567) = 35
Math.Floor(34.567) = 34
```

اگر بخواهید یک عدد را به عددی با قسمت اعشاری مشخص گرد کنید، می‌توانید از متد Math.Round() استفاده کنید:

```
//Round 3.31674 into 2 decimal places
double number = 3.31674

Console.WriteLine(Math.Round(number, 2));
```

```
3.32
```

به توان رساندن یک عدد

برای به توان رساندن یک عدد از متد Math.Pow() استفاده می‌شود. این متد دو آرگومان از نوع double قبول کرده که اولین آرگومان، پایه و دومی توان می‌باشد. مقدار برگشتی از این متد double است. به کد زیر توجه کنید:

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("2^{0} = {1}", i, Math.Pow(2, i));
}
```

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
```

گرفتن ریشه یک عدد

برای محاسبه ریشه یک عدد از متد Math.Sqrt() استفاده می‌شود. این متد یک عدد به عنوان آرگومان قبول می‌کند که همان عددی است که می‌خواهیم ریشه آن را محاسبه کنیم. مقدار برگشتی از این متد هم double می‌باشد.

```
Console.WriteLine(Math.Sqrt(25));
```

```
5
```

یافتن بزرگ‌ترین و کوچک‌ترین عدد

کلاس System.Math دارای متدهای Math.Min() و Math.Max() برای یافتن بزرگ‌ترین و کوچک‌ترین عدد از بین چندین عدد می‌باشد. هر دو متد، دو آرگومان از نوع عددی قبول می‌کنند. در حالت پیش‌فرض دو عدد را می‌توانید با هم مقایسه نمایید.

```
Console.WriteLine(Math.Min(1,2));
Console.WriteLine(Math.Max(1,2));
```

```
1
2
```

برای جلوگیری از محدودیت این متدها، می‌توان به صورت تو در تو از آنها استفاده کرد:

```
//Get the maximum and minimum of 3 numbers
int max = Math.Max(Math.Max(1, 2), 3);
int min = Math.Min(Math.Min(1, 2), 3);

Console.WriteLine("Max = {0}", max);
Console.WriteLine("Min = {0}", min);
```

```
Max = 3
Min = 1
```

همچنین می‌توانید یک تابع تعریف کنید که کوچک‌ترین و بزرگ‌ترین هر تعداد عدد را به شما معرفی کند:

```
static int GetMax(int[] numbers)
{
    int maximum = numbers[0];

    for (int i = 1; i < numbers.Length; i++)
    {
        maximum = Math.Max(maximum, numbers[i]);
    }

    return maximum;
}

static int GetMin(int[] numbers)
{
    int minimum = numbers[0];

    for (int i = 1; i < numbers.Length; i++)
    {
        minimum = Math.Min(minimum, numbers[i]);
    }

    return minimum;
}

static void Main()
{
    int[] numbers = { 32, 17, 45, 10, 5 };
    int max = GetMax(numbers);
    int min = GetMin(numbers);
    Console.WriteLine(max);
    Console.WriteLine(min);
}
```

```
45
5
```

توابع `GetMax()` و `GetMin()` آرایه ای از اعداد صحیح قبول می‌کنند (هر تعداد عدد را قبول می‌کنند). در داخل توابع ما فرض را بر این گذاشته‌ایم که اولین مقدار بزرگ‌ترین و آخرین مقدار کوچک‌ترین عدد است. سپس یک حلقه `for` ایجاد کرده که با اندیس ۱ شروع می‌شود. با استفاده از متدهای `Math.Min()` و `Math.Max()` تعیین می‌کنیم که آیا عنصر جاری حلقه از مقدار جاری اعداد بزرگ‌تر است یا کوچک‌تر. که در این صورت مقدار جاری را جایگزین متغیرهای بزرگ یا کوچک می‌کنیم. سپس مقادیر نتیجه را برگشت می‌دهیم.

Math.PI

از ثابت `PI` که مقدار ۳٫۱۴۱۵۹۲۶۵۳۵۸۹۷۹۳۲۳۸۴۶ را در خود ذخیره دارد، زمانی که بخواهید محیط یا مساحت یک دایره را پیدا کنید، استفاده می‌شود. برای یافتن محیط یک دایره به صورت زیر عمل می‌شود که در آن `radius` شعاع می‌باشد:

```
double area = Math.PI * Math.Pow(radius, 2);
```

کلاس System.Math دارای متدهای بیشتری است که ما به توضیح همین چند متد بسنده کردیم.

ایجاد عدد تصادفی

با استفاده از کلاس Random می‌توان اعداد تصادفی تولید کرد. این کلاس دارای متدهایی برای تولید عدد تصادفی می‌باشد. شانس آمدن هر عددی وجود دارد. می‌توان با استفاده از متد Next() یک عدد تصادفی مثبت مانند مثال زیر تولید کرد:

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine(generator.Next());
}
```

```
673131583
2075827769
530790962
853400196
1181061071
1657679493
1459501829
452543008
1814178911
1933670708
```

خروجی شما مطمئناً متفاوت است، چون همه اعداد تصادفی می‌باشند. یکی دیگر از نسخه‌های متد Next() به شما اجازه می‌دهد که تعدادی عدد تصادفی را که از یک عدد خاص بزرگ‌تر نباشند، تولید کنید. این نسخه از متد همچنین باعث می‌شود که تعداد اعداد تولید شده از مقدار ذکر شده در پرانتز بیشتر نشود. به عنوان مثال برنامه زیر اعدادی را تولید می‌کند که، نه مقدار آنها از ۱۱ بیشتر است و نه تعداد آنها.

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine(generator.Next(11));
}
```

```
2
3
4
4
10
4
1
1
0
8
```

متد Next() آرگومانی را به عنوان بزرگ‌ترین عدد می‌پذیرد. بدین معنی که این متد یک مقدار تصادفی که کوچک‌تر از (نه برابر) آرگومانی که می‌پذیرد، را برگشت می‌دهد. مثلاً در مثال بالا عدد ۱۱ را به این متد ارسال کرده‌ایم در نتیجه، متد هم مقادیر ممکن بین ۰ و ۱۰ را تولید می‌کند.

همچنین می‌توان محدوده اعدادی که متد Next() برمی گرداند را، مشخص کرد. به عنوان مثال اگر بخواهید اعداد ۱ تا ۶ روی طاس تولید شود، می‌توان کد بالا را به صورت زیر تصحیح کرد:

```
Console.WriteLine(generator.Next(1, 7)); //Returns random value from 1 to 6
```

اولین آرگومان نشان دهنده کوچک‌ترین و دومین آرگومان نشان دهنده بزرگ‌ترین عددی است که توسط این متد برگشت داده می‌شود. می‌توان از یک مقدار هم برای تولید یک توالی تکراری از اعداد تصادفی استفاده کرد. بدین معنی که با هر بار اجرای برنامه، توالی و نوع اعداد تولید شده، مانند سری قبلی باشد که برنامه اجرا شده است. این مقدار، هر عددی می‌تواند باشد. برای اختصاص این مقدار باید آن را در داخل پرانتز سازنده کلاس Random قرار دهید. مانند عدد صفر در مثال زیر:

```
Random generator = new Random(0);
//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
      Console.WriteLine(generator.Next(11));
}
```

```
7
8
8
6
2
6
9
4
10
3
```

با هر بار اجرای برنامه بالا، همین اعداد را مشاهده خواهید نمود. حال اجازه بدهید که یک مثال ساده را توضیح دهیم. این برنامه پیغام‌های متفاوتی در هر بار اجرای آن به شما نشان می‌دهد.

```
using System;

namespace RandomMessage
{
    class Program
    {
        static void Main()
        {
            Random generator = new Random();

            int messageNumber = generator.Next(1, 4);

            switch (messageNumber)
            {
                case 1:
                    Console.WriteLine("Hello to you my friend.");
                    break;
                case 2:
                    Console.WriteLine("Good day to you sir/mam.");
                    break;
                case 3:
                    Console.WriteLine("Have a happy day.");
                    break;
            }

            Console.ReadKey();
```

```
        }
    }
}
```

برنامه مقادیر ۱ تا ۳ را تولید می‌کند. سپس با استفاده از دستور switch برای نشان دادن یک پیغام برای هر مقدار ممکن استفاده می‌کنیم. بعد از چندین بار اجرای برنامه متوجه می‌شوید که پیغام خوش آمد گویی تغییر می‌کند.

رشته‌ها و عبارات با قاعده (منظم)

رشته‌ها، معمول‌ترین انواع داده‌ای هستند که تقریباً در همه زبان‌های برنامه‌نویسی یافت می‌شوند. یک رشته گروهی از کاراکترها مانند حروف، اعداد یا نشانه‌ها می‌باشد. رشته به وسیله کلمه کلیدی string تعریف می‌شود. اهمیت این نوع داده‌ای زمانی مشخص می‌شود که شما بخواهید اطلاعاتی از قبیل نام، آدرس، جنسیت یا ایمیل خودتان را ذخیره کنید. با استفاده از رشته‌ها می‌توان فوراً به کاربر اطلاعاتی ارائه و یا در مورد یک خطا در برنامه هشدار داد. همچنین با استفاده از regular expression یا عبارات با قاعده در سی‌شارپ، می‌توان کارهایی شبیه به رشته‌ها انجام داد. از عبارات با قاعده در اعتبارسنجی اطلاعات استفاده می‌شود. در درس‌های آینده در مورد خصوصیات مختلف رشته‌ها و متدهایی که به وسیله آنها شما می‌توانید این خصوصیات را دستکاری کنید، توضیح داده خواهد شد.

کلاس System.String

رشته‌ها در سی‌شارپ با کلاس String سر و کار دارند. قبل از استفاده از رشته‌ها ابتدا باید فضای نام System را در ابتدای برنامه وارد کنید. برای ایجاد یک رشته چندین راه وجود دارد.

```
String str1;
str1 = "An example of a string.";
```

به این نکته توجه کنید که رشته‌ها را باید در داخل دابل کوتیشن ("") قرار دهید. دابل کوتیشن به کامپایلر نشان می‌دهد که داده‌ای را که شما می‌خواهید در متغیر ذخیره کنید، یک رشته است. این به شما اجازه می‌دهد که به جای استفاده از کلاس System.String از کلمه کلیدی string برای ایجاد یک رشته استفاده کنید.

```
string myString;
myString = "An example of a string.";
```

برای تعریف و مقدار دهی به یک رشته می‌توان به صورت زیر عمل کرد:

```
string myString = "This is another string";
```

همچنین می‌توان از کلمه کلیدی new و سازنده System.String برای تخصیص یک مقدار به رشته استفاده کرد:

```
String myString = new String("This is a string.");
string myString = new string("This is a string");
```

در دات‌نت رشته یک نوع ارجاعی است اما بسیار شبیه به انواع مقداری رفتار می‌کند. به تکه کد زیر توجه کنید:

```
Button btn1 = new Button();
btn1.Text = "Button 1 ";
Button btn2 = btn1; // Assign address of btn1 to btn2

btn2.Text += " and Button 2";
Console.WriteLine(btn1.Text);
Console.WriteLine(btn2.Text);
```

```
Button 1 and Button 2
Button 1 and Button 2
```

ابتدا یک دکمه ایجاد کرده و نام آن را btn1 می‌گذاریم و خاصیت Text آنرا برابر یک رشته قرار می‌دهیم. سپس ارجاع btn1 را برابر دکمه ای دیگر به نام btn2 قرار دهید. اکنون هر دو به یک دکمه یکسان وابسته می‌شوند. حال خاصیت Text، btn2 را به وسیله الحاق آن با رشته دیگر تغییر می‌دهیم. به این نکته توجه کنید btn2.Text شامل متن btn1 است چون هر دو به یک دکمه وابسته‌اند. بعد از تغییر، محتویات خاصیت Text هر دو دکمه را در خروجی چاپ می‌کنیم. نتیجه یکسان است. هنگامی که بخواهیم متن btn2 را اصلاح کنیم، متن btn2 هم تغییر می‌کند (در خروجی دستور WriteLine() مشاهده می‌کنید). اگر رشته‌ها واقعاً جزء انواع مرجع باشند، پس باید دو خط کد زیر رفتار یکسانی از خود نشان بدهند:

```
string str1 = "String 1";
string str2 = str1;
```

حال str1 و str2 باید به یک شیء اشاره کنند. اگر مقدار str2 را تغییر دهیم باید مقدار str1 نیز تغییر کند.

```
str2 += " and String 2";
```

اجازه بدهید مقادیر رشته‌ها را چاپ کنیم:

```
Console.WriteLine(str1);
Console.WriteLine(str2);
```

```
String 1
String 1 and String 2.
```

همانطور که می‌بینید مقادیر متفاوت هستند. مقدار str1 در str2 کپی شد. آنها به یک نمونه مشابه اشاره نمی‌کنند. بنابراین اگر در str2 تغییری ایجاد کنید در str1 تأثیر نمی‌گذارد. یک رشته نمی‌تواند نوع مقداری باشد چون طول کاراکترهای تشکیل دهنده آن غیر قابل پیش بینی است. یک رشته مجموعه‌ای از کاراکترها می‌باشد. کد زیر نحوه جداسازی کاراکترهای یک رشته را به وسیله حلقه foreach نشان می‌دهد.

```
string str1 = "This is a string";
foreach (char c in str1)
{
Console.WriteLine(c);
}
```

```
T
h
i
s

i
s
```

```
a

s
t
r
i
n
g
```

در حقیقت، نسخه دیگر سازنده System.String یک آرایه از نوع کاراکتر قبول می‌کند.

```
char[] charArray = { 'H', 'e', 'l', 'l', 'o' };
String myString = new String(charArray);
```

در درس آینده نحوه مقایسه و مرتب سازی رشته‌ها را نشان خواهیم داد.

مقایسه رشته‌ها

می‌توان رشته‌ها را به روش‌های مختلف با هم مقایسه کرد. به عنوان مثال با استفاده از عملگر == می‌توان تست کرد که آیا دو رشته با هم برابرند یا نه. با وجودیکه رشته‌ها از نوع مرجع هستند، این عملگر مقدار رشته‌ها را با هم مقایسه می‌کند نه آنها را.

```
string str1 = "Hello";
string str2 = "Hello";
string str3 = "Goodbye";

Console.WriteLine("str1 == str2 : {0}", str1 == str2);
Console.WriteLine("str1 == str3 : {0}", str1 == str3);
```

```
str1 == str2 : True
str1 == str3 : False
```

همچنین از متد استاتیک String.Compare() هم برای مقایسه دو رشته استفاده می‌شود. به کد زیر توجه کنید:

```
if (String.Compare(str1, str2) == 0)
{
   Console.WriteLine("str1 is equal to str2");
}
```

متد String.Compare() دو رشته را قبول می‌کند و اگر با هم برابر باشند مقدار ۰، اگر مقدار اولین رشته از دومین رشته بیشتر باشد مقدار ۱ و اگر مقدار اولین رشته از دومین رشته کوچک‌تر باشد مقدار ۱- را بر می‌گرداند. بنابراین برای تشخیص تساوی دو رشته با استفاده از متد Compare() باید مانند کد بالا تست کنید که آیا مقدار برگشتی برابر صفر است یا نه.

چطور تشخیص بدهیم که یک رشته از رشته دیگر کوچک‌تر یا بزرگ‌تر است؟ هر کاراکتر در رشته، به یونیکد معادل خود تبدیل می‌شود. اولین کاراکتر اولین رشته با اولین کاراکتر دومین رشته مقایسه می‌شود. اگر برابر بودند سپس دومین کاراکترها و به همین ترتیب بقیه کاراکترها با هم مقایسه می‌شوند.

سربارگذاری دیگر این متد یک آرگومان سومی دیگر از نوع بولی قبول می‌کند و به وسیله آن تشخیص می‌دهد که آیا در مقایسه دو رشته بزرگی و کوچکی حروف در نظر گرفته شده است یا نه (دو رشته از لحاظ بزرگی و کوچکی حروف با هم برابرند یا نه). در حالت پیش فرض متد

String.Compare() دو رشته یکسان را که فقط در بزرگی و کوچکی حروف با هم متفاوت باشند را، دو رشته متفاوت در نظر می‌گیرد. کد زیر نشان می‌دهد که چطور می‌توان دو رشته را بدون در نظر گرفتن بزرگی و کوچکی حروف با هم مقایسه کرد:

```
string str1 = "Microsoft";
string str2 = "microsoft";

if (String.Compare(str1, str2, true)==0)
{
   Console.WriteLine("The strings are equal.");
}
```

```
The strings are equal.
```

در زیر چندین سربارگذاری از متد String.Compare() بیان شده است:

متد	توضیح
Compare(String, String)	دو رشته را با هم مقایسه می‌کند.
Compare(String, String, Boolean)	دو رشته را بدون در نظر گرفتن بزرگی و کوچکی حروفشان مقایسه می‌کند.
Compare(String, Int32, String, Int32, Int32)	زیر رشته‌های دو رشته رشته را با هم مقایسه می‌کند
Compare(String, Int32, String, Int32, Int32, Boolean)	زیر رشته‌های دو رشته را بدون در نظر گرفتن بزرگی و کوچکی حروفشان مقایسه می‌کند.

یکی دیگر از روش‌های مقایسه دو رشته استفاده از متد CompareTo() است. به نحوه استفاده از این متد توجه کنید:

```
string str1 = "Hello";
string str2 = "Hello";

if (str1.CompareTo(str2) == 0)
{
   Console.WriteLine("The strings are equal.");
}
```

```
The strings are equal.
```

متد CompareTo() یک مقدار صحیح را بر می‌گرداند. مانند متد Compare() این متد نیز اگر رشته‌ها با هم برابر باشند مقدار ۰، اگر مقدار اولین رشته از دومین رشته بیشتر باشد مقدار ۱ و اگر مقدار اولین رشته از دومین رشته کوچک‌تر باشد، مقدار ۱- را بر می‌گرداند.

الحاق رشته‌ها

چندین راه برای الحاق رشته‌ها به هم وجود دارد. الحاق به معنای چسباندن چندین رشته به هم و تبدیل آنها به یک رشته است. در سی‌شارپ یکی از راه‌های ساده الحاق رشته‌ها، استفاده از عملگر + است:

```
string str1 = "Happy ";
string str2 = "New Year";
string result = str1 + str2;

Console.WriteLine(result);
```

```
Happy New Year
```

مشاهده می‌کنید که استفاده از این عملگر چطور باعث ترکیب دو عملوند رشته‌ای شد. به این نکته توجه کنید که استفاده از عملگر + باعث ترکیب یک رشته با یک نوع داده‌ای دیگر مانند int می‌شود که در این حالت نوع int به صورت خودکار به نوع رشته‌ای تبدیل می‌شود.

```
Console.WriteLine("Number of Guests: " + 100);
```

```
Number of Guests: 100
```

راه دیگر برای الحاق دو رشته استفاده از متد استاتیک String.Concat() می‌باشد. شما می‌توانید تعداد بی نهایت رشته یا شیء به عنوان آرگومان به این متد ارسال کنید. در زیر نحوه استفاده از این متد نشان داده شده است:

```
string result = String.Concat("We have ", 100, " guests ", " this evening.");

Console.WriteLine(result);
```

```
We have 100 guests this
```

از متد استاتیک String.Join() نیز یکی دیگر از روش‌های الحاق رشته‌ها است. از این رشته برای ترکیب رشته‌ها و ذخیره آنها در یک آرایه از نوع رشته استفاده می‌شود.

```
string[] words = { "Welcome", "to", "my", "site." };
string result = String.Join(" ", words);

Console.WriteLine(result);
```

```
Welcome to my site
```

متد String.Join() دو آرگومان قبول می‌کند، اولین آرگومان، رشته‌ای است که در داخل هر یک از رشته‌هایی که قرار است با هم ترکیب شوند، قرار می‌گیرد. که در مثال بالا یک رشته که در اصل یک فضای خالی است (" ") را بین کلمات قرار می‌دهیم. دومین آرگومان خود آرایه رشته‌ای است. سربارگذاری دیگر این متد به شما اجازه می‌دهد به جای آرایه رشته‌ای لیست رشته‌ها را ارسال کنید.

```
string result = String.Join(" ", "Welcome", "to", "my", "site.");

Console.WriteLine(result);
```

```
Welcome to my site
```

همه رشته‌ها در دات‌نت تغییر ناپذیر هستند. این بدین معنی است که متغیر رشته‌ای یکبار مقدار دهی می‌شود و مقدار آن تغییر نمی‌کند. وقتی مقدار یک رشته را در سی‌شارپ اصلاح می‌کنید یک نسخه جدید از رشته ایجاد و نسخه قبلی دور انداخته می‌شود. بنابراین همه متدهایی که با رشته‌ها سر و کار دارند یک نسخه از رشته اصلاح شده را بر می‌گردانند و نسخه اصلی دست نخورده باقی می‌ماند. کلاس StringBuilder به شما اجازه ترکیب رشته‌های بزرگ را می‌دهد که در درس آینده در مورد آن توضیح خواهیم داد.

جا دادن یک رشته در داخل رشته دیگر

دات نت به شما اجازه می‌دهد که با استفاده از متد Insert() یک رشته را در داخل رشته دیگر قرار دهید. به عنوان مثال اگر رشته‌ای مانند Hello World! داشته باشید می‌توانید با استفاده از متد ذکر شده کلمه Happy را در وسط آن قرار دهید و رشته جدیدی مانند Hello Happy World! ایجاد کنید. کد زیر نحوه استفاده از متد Insert() را نشان می‌دهد.

```
string str1 = "Hello World!";
string str2 = "Happy ";
string result = str1.Insert(6, str2);

Console.WriteLine(result);
```

```
Hello Happy World!
```

متد Insert() دو آرگومان قبول می‌کند، اولین آرگومان مکانی که رشته جدید می‌خواهد در آن قرار گیرد و دومین آرگومان، خود رشته جدید را نشان می‌دهد. اندیس اولین کاراکتر یک رشته با صفر و اندیس آخرین کاراکتر یک رشته با یک واحد کمتر از طول رشته نمایش داده می‌شود. در رشته Hello World! می‌خواهیم رشته جدید را درست بعد از اولین فضای خالی یعنی جایی که حرف W قرار دارد، جای دهیم. به کادر زیر توجه کنید:

```
H  e  l  l  o     W  o  r  l  d   !
0  1  2  3  4  5  6  7  8  9  10  11
```

اگر توجه کرده باشید رشته Happy در این مکان (بعد از اولین فضای خالی) قرار گرفته است. و هر چیز بعد از آن به جلو رانده و رشته Hello Happy World! ایجاد می‌شود.

اضافه کردن کاراکتر به سمت چپ یا راست یک رشته

با استفاده از متدهای PadLeft() و PadRight() می‌توان کاراکتر یا فضاهای خالی را به سمت چپ یا راست رشته‌ها اضافه کرد. به عنوان مثال اگر بخواهید یک سری فضا را به سمت چپ رشته جهت ترازبندی آن اضافه کنید، می‌توانید از متد PadLeft() استفاده کنید:

```
string str1 = "Example";
str1 = str1.PadLeft(10);

Console.WriteLine(str1);
```

```
   Example
```

متدهای PadLeft() و PadRight() آرگومانی را قبول می‌کنند که تعداد کاراکترها یا فضاهای خالی وارد شده را تشخیص می‌دهد. همانطور که در مثال بالا مشاهده کردید عدد ۱۰ ارسال شده باعث ایجاد ۳ فضای خالی در سمت چپ رشته می‌شود. برای به دست آوردن مقدار این فضاها یا کاراکترهایی که در سمت چپ یا راست یک رشته قرار می‌گیرند، می‌توان از فرمول (تعداد کاراکتر یا فضای خالی – طول رشته) استفاده کرد.

مثلاً در مثال بالا طول رشته Example برابر ۷ می‌باشد و زمانی که ما عدد ۱۰ را به عنوان آرگومان ارسال می‌کنیم با استفاده از فرمول ۳= ۷ - ۱۰ می‌توان فهمید که چند فضای خالی در سمت چپ رشته قرار می‌گیرد (۳ فضای خالی). اگر مقدار آرگومانی که به متد PadLeft() ارسال می‌شود، از طول رشته کمتر باشد هیچ تاثیری بر رشته نهایی ندارد. عمل PadRight نیز شبیه به PadLeft است.

```
string str1 = "Example";
str1 = str1.PadRight(10);

Console.WriteLine(str1 + "|");
```

```
Example   |
```

همانطور که در مثال بالا مشاهده می‌کنید ما با استفاده از متد PadRight() کاراکتر "|" را به سمت راست رشته اضافه کرده‌ایم. یکی دیگر از نسخه‌های این دو متد آرگومان دومی را هم قبول می‌کند که کاراکتر دلخواهی است که به سمت راست یا چپ رشته اضافه می‌شود. به مثال زیر توجه کنید:

```
string str1 = "Example";
str1 = str1.PadLeft(10, '*');

Console.WriteLine(str1);
```

```
***Example
```

استفاده از متدهای بالا زمانی مفید است که خروجی داده‌ها به صورت جدول باشد. به مثال زیر توجه کنید:

```
string[,] data = { { "Name", "Gender", "Age" },
                   { "John", "Male", "21" },
                   { "Mark", "Male", "18" },
                   { "Jean", "Female", "18" } };

for (int i = 0; i < data.GetLength(0); i++)
{
    for (int j = 0; j < data.GetLength(1); j++)
    {
        Console.Write(data[i, j].PadRight(10));
    }

    Console.WriteLine();
}
```

```
Name      Gender    Age
John      Male      21
Mark      Male      18
Jean      Female    18
```

در مثال بالا از یک آرایه چند بعدی برای ذخیره داده‌هایمان استفاده کرده‌ایم. اولین ردیف عناوین و ردیف‌های دیگر داده‌ها هستند.

حذف زائده‌ها از رشته‌ها

برخی اوقات کاربر به طور ناخواسته با تایپ فضاهای خالی غیر ضروری باعث ایجاد خطا می‌شود. وقتی رشته‌ای از کاربر دریافت می‌شود، مخصوصاً اگر از طریق یک text box یا کنترلی شبیه به آن این کار انجام شود، فضاهای خالی ابتدا و انتهای رشته به وسیله این کنترل‌ها حذف می‌شوند.

دات‌نت با استفاده از متدهای Trim()، TrimStart() و TrimEnd() این کار را انجام می‌دهد. TrimStart() فضاهای خالی ابتدا، TrimEnd() فضاهای خالی انتها و متد Trim() فضاهای خالی هر دو طرف رشته را حذف می‌کند. به عنوان مثال:

```
string str1 = "   Example   ";
str1 = str1.Trim();

Console.WriteLine(str1);
```

```
Example
```

متد Trim() دارای یک سربارگذاری است که آرایه ای از کاراکترهایی را که می‌خواهید از رشته حذف شوند را، قبول می‌کند. به مثال زیر توجه کنید:

```
string str1 = "&&&&Hello***";
str1 = str1.Trim("&*".ToCharArray());

Console.WriteLine(str1);
```

```
Hello
```

همانطور که مشاهده می‌کنید رشته بالا دارای کاراکترهای غیر ضروری در ابتدا و انتهای خود است. این کاراکترهای اضافی را به متد Trim() ارسال کرده و آنها را حذف می‌کنیم. به این نکته توجه کنید که ما از متد ToCharArray() استفاده کرده‌ایم. متد Trim() یک آرایه کاراکتری قبول کرده و هنگامی که ما یک رشته را به آن ارسال می‌کنیم متد ToCharArray() آن را به کاراکتر تبدیل می‌کند.

جداکردن رشته‌ها

اگر بخواهید یک رشته را به چند رشته تکه تکه کنید می‌توانید از متد split() استفاده نمایید. اجازه دهید نگاهی به سربارگذاری‌های مختلف این متد بیندازیم. متد Split() آرایه ای از رشته‌ها را بر می‌گرداند که هر عنصر از این آرایه شامل یک زیر رشته است. اولین سربارگذاری این متد آرایه ای از کاراکترها را قبول می‌کند و بر اساس آنها تشخیص می‌دهد که رشته باید در چه جایی به قسمت‌های مختلف تقسیم شود.

```
string message = "The quick brown fox jumps over the lazy dog.";
string[] substrings = message.Split(' ');

foreach (string s in substrings)
{
    Console.WriteLine(s);
}
```

```
The
quick
brown
fox
jumps
over
the
lazy
dog.
```

در مثال بالا از کاراکتر فاصله (' ') برای جدا کردن کلمات در رشته بالا استفاده کرده‌ایم، چون دو کلمه متوالی به وسیله فاصله از هم جدا می‌شوند. کلمات در آرایه substrings() ذخیره می‌شوند. سپس با استفاده از دستور foreach آنها را در خطوط جداگانه چاپ می‌کنیم. می‌توان تعداد زیر رشته‌های برگشتی را به وسیله سربارگذاری دیگر متد Split() محدود کرد.

```
string message = "The quick brown fox jumps over the lazy dog.";
string[] substrings = message.Split(" ".ToCharArray(), 3);

foreach (string s in substrings)
{
    Console.WriteLine(s);
}
```

```
The
quick
brown fox
```

همانطور که در مثال بالا مشاهده می‌کنید دومین آرگومان برای تشخیص تعداد زیر رشته‌ها به کار می‌رود. خروجی نشان می‌دهد که دو کلمه اول از رشته جدا شده‌اند و مابقی رشته در عنصر آخر آرایه ذخیره می‌شود.

سربارگذاری دیگر این متد استفاده از نوع شمارشی StringSplitOptions است. این نوع شمارشی دارای دو مقدار None و RemoveEmptyEntries است. با استفاده از StringSplitOptions.None می‌توانید یک زیر رشته خالی در آرایه ذخیره کنید. و با استفاده از StringSplitOptions.RemoveEmptyEntries می‌توان تمام زیر رشته‌های خالی را حذف کرد. به تکه کد زیر توجه کنید:

```
string message = "string1#string2##string4#string5";
string[] results = message.Split("#".ToCharArray(), StringSplitOptions.None);

foreach (string s in results)
{
   Console.WriteLine(s);
}
```

```
string1
string2

string4
string5
```

در سربارگذاری متد Split() که در مثال بالا مشاهده کردید، متد یک آرایه از نوع کاراکتر به عنوان اولین آرگومان قبول کرده و به دستور ما از کاراکتر "#" برای تعیین حد زیر رشته‌ها استفاده می‌کند. به همین دلیل است که از متد ToCharArray() استفاده کرده‌ایم. دومین آرگومان مقدار StringSplitOption است. در این قسمت ما مقدار None را ارسال می‌کنیم و چون رشته دارای زیر رشته خالی است ارسال این مقدار چندان تاثیری در نمایش نهایی ندارد (می‌توانید یکبار قسمت StringSplitOptions.None را حذف کرده و برنامه را اجرا کنید). حال از مقدار RemoveEmptyEntries استفاده می‌کنیم.

```
string message = "string1#string2##string4#string5";
string[] results = message.Split("#".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

foreach (string s in results)
{
    Console.WriteLine(s);
}
```

```
string1
string2
string4
string5
```

همانطور که مشاهده می‌کنید نتیجه استفاده از این مقدار حذف سومین زیررشته خالی است. از این روش اغلب برای ذخیره داده در فایل متنی استفاده می‌شود.

جستجو کردن در رشته‌ها

جستجوی رشته‌ها به وسیله متدهای دات‌نت بسیار راحت است. اجازه بدهید که نگاهی به متدهای مختلفی که محل وقوع یک رشته خاص را پیدا می‌کنند بیندازیم. متدهای `IndexOf()` و `LastIndexOf()` محل یک رشته خاص را در رشته دیگر نشان می‌دهند. اگر رشته مورد نظر پیدا نشود، متدهای فوق مقدار ۱- را بر می‌گردانند. به مثالی در مورد متد `IndexOf()` توجه کنید:

```
string str = "The quick brown fox jumps over the lazy dog.";
int index = str.IndexOf("quick");

Console.WriteLine(str);
Console.WriteLine("quick was found at position " + index);
```

```
The quick brown fox jumps over the lazy dog.
quick was found at position 4
```

`IndexOf()` یک رشته را که شما به دنبال آن هستید را دریافت می‌کند. در مثال بالا متد مقدار ۴ را بر می‌گرداند چون اندیس کلمه `quick` عدد ۴ است. به یاد داشته باشید که اندیس یا مکان از صفر شروع شده و تا یک واحد کمتر از طول رشته ادامه می‌یابد، بنابراین کاراکتر پنجم از یک رشته، دارای اندیس ۴ است. متد `LastIndexOf()` کمی متفاوت است:

```
string str = "A very very very good day.";
int index = str.LastIndexOf("very");

Console.WriteLine(str);
Console.WriteLine("Last occurrence of very was found at position " + index);
```

```
A very very very good day.
Last occurrence of very was found at position 12
```

`LastIndexOf()` بسیار شبیه `IndexOf()` است با این تفاوت که اندیس آخرین محل وقوع رشته را بر می‌گرداند. در کد بالا آخرین محل وقوع کلمه "very" اندیس ۱۲ است. اگر یک رشته خاص در هنگام جستجو پیدا نشود این دو متد مقدار ۱- را بر می‌گردانند.

```
string str1 = "This is a sample string.";

Console.WriteLine(str1);

if (str1.IndexOf("Whatever") == -1)
{
   Console.WriteLine("\"Whatever\" was not found in the string.");
}
```

```
This is a sample string.
"Whatever" was not found in the string.
```

یکی دیگر از سربارگذاری‌های این دو متد این است که یک آرگومان دومی را قبول می‌کنند محل شروع جستجو را تعیین می‌کند. به تکه کد زیر توجه کنید:

```
string str1 = "This is a sample string.";

Console.WriteLine(str1);

if (str1.IndexOf("This", 5) == -1)
{
   Console.WriteLine("\"This\" was not found in the string.");
}
```

همانطور که مشاهده می‌کنید با وجودیکه کلمه this در جمله بالا وجود دارد ولی چون ما محل شروع جستجو در رشته را از اندیس ۵ انتخاب کرده‌ایم، کلمه پیدا نمی‌شود. اگر بخواهیم تمام محل‌های وقوع یک کلمه را پیدا کنیم، می‌توان به صورت زیر عمل کنیم:

```
int position = 0;
string str1 = "This is a long long long string...";
do
{
   position = str1.IndexOf("long", position);

   if (position != -1)
      Console.WriteLine(position);

    position++;
} while (position > 0);
```

```
10
15
20
```

متد Contains() هم می‌تواند چک کند که آیا یک رشته در داخل رشته دیگر وجود دارد یا نه.

```
string str1 = "This is a sample string.";

Console.WriteLine(str1);

if (str1.Contains("sample"))
{
   Console.WriteLine("\"sample\" exists in the string.");
}
```

```
This is a sample string.
"sample" exist in the string.
```

این متد اگر رشته مورد نظر وجود داشته باشد مقدار true و اگر وجود نداشته باشد مقدار false را بر می‌گرداند. هر دو متد StartsWith() و EndsWith() برای یافتن یک رشته در ابتدا و انتهای یک رشته خاص به کار می‌روند.

```
string str1 = "Apple";

Console.WriteLine(str1);

if (str1.StartsWith("A"))
{
    Console.WriteLine("The word starts with A.");
}
if (str1.EndsWith("e"))
{
    Console.WriteLine("The word ends with e.");
}
```

```
Apple
```

```
The word starts with A.
The word ends with e.
```

استخراج، حذف و جایگزین کردن رشته‌ها

برای استخراج قسمتی از یک رشته می‌توان از متد Substring() استفاده کرد. این متد دو آرگومان قبول می‌کند که یکی اندیس شروع و دیگری طولی از رشته را که می‌خواهیم استخراج کنیم را نشان می‌دهد. به مثال زیر توجه کنید.

```
string str1 = "This is a sample string.";
//Extract sample
string str2 = str1.Substring(10, 6);

Console.WriteLine("str1 = {0}", str1);
Console.WriteLine("str2 = {0}", str2);
```

```
str1 = This is a sample string.
str2 = sample
```

استخراج را از اندیس ۱۰ شروع کرده‌ایم (آرگومان اول که عدد ۱۰ است). همانطور که مشاهده می‌کنید کلمه "sample" از اندیس ۱۰ شروع شده است (کاراکتر یازدهم). آرگومان دوم نشان می‌دهد که ما چند کاراکتر را می‌خواهیم استخراج کنیم. ازآنجاییکه عدد ۶ را برای این آرگومان در نظر گرفته‌ایم، ۶ کاراکتر از رشته مورد نظر استخراج می‌شود. اگر نخواهید که مکان قرار گرفتن کلمه "sample" را به صورت دستی شمارش کنید، می‌توانید با استفاده از متد IndexOf() این کار را انجام دهید.

```
string str2 = str1.Substring(str1.IndexOf("sample"), 6);
```

یکی دیگر ازسربارگذاری‌های متد Substirng() فقط یک آرگومان که برای تعیین کردن اندیس شروع استخراج به کار می‌رود را قبول می‌کند. در نتیجه استخراج از این اندیس شروع شده و تا پایان رشته ادامه می‌یابد.

حذف رشته‌ها

برای حذف رشته‌ها می‌توان از متد Remove() استفاده کرد. پارامترهای ان شبیه متد Substring() می‌باشد.

```
string str1 = "This is a sample string.";
Console.WriteLine(str1);
Console.WriteLine("Removing \"sample\"...");
str1 = str1.Remove(10, 7);
Console.WriteLine(str1);
```

```
This is a sample string.
Removing "sample "...
This is a string.
```

یکی از سربارگذاری‌های متد Remove() یک آرگومان که اندیس شروع حذف را نشان قبول می‌کند. حذف رشته از این اندیس شروع و تا پایان رشته ادامه می‌یابد.

جایگزین کردن رشته‌ها

با استفاده از متد Replace() می‌توان یک رشته خاص را با یک رشته دیگر عوض کرد. به عنوان مثال در کد زیر می‌توان کلمه "dog" را با کلمه "cat" عوض کرد.

```
string str1 = "That dog is a lovely dog.";
Console.WriteLine(str1);
Console.WriteLine("Replacing all dogs with cats...");
str1 = str1.Replace("dog", "cat");
Console.WriteLine(str1);
```

```
That dog is a lovely dog.
Replacing all dogs with cats...
That cat is a lovely cat.
```

متد Replace() دو آرگومان قبول می‌کند. اولین آرگومان رشته‌ای است که می‌خواهیم آن را با یک رشته جدید جایگزین کنیم (رشته قدیم) و دیگری رشته جدید است. متد Replace() تمام کلمات "dog" واقع در رشته را پیدا کرده و کلمه "cat" را جایگزین آنها می‌کند. از این متد می‌توان برای حذف همه کلمات مثلاً "dog" در مثال زیر استفاده کرد.

```
string str1 = "That dog is a lovely dog.";
Console.WriteLine(str1);
Console.WriteLine("Removing all dogs...");
str1 = str1.Replace("dog", String.Empty);
Console.WriteLine(str1);
```

```
That dog is a lovely dog.
Removing the dogs...
That  is a lovely .
```

در کد بالا از String.Empty به عنوان یک رشته جایگزین استفاده شده است. String.Empty معادل "" و به معنای رشته خالی می‌باشد. بنابراین تمام محل‌های وقوع کلمه "dog" با رشته خالی جایگزین شده و از رشته واقعی حذف می‌شوند.

تغییر بزرگی و کوچکی حروف یک رشته

می‌توان بزرگی و کوچکی حروف یک رشته را تغییر داد. به عنوان مثال یک رشته که متشکل از حروف کوچک است را می‌توان به حروف بزرگ تبدیل کرد. با استفاده از متدهای ToUpper() و ToLower() می‌توان حروف رشته را بزرگ یا کوچک کرد.

```
string lowercase = "abc";
string uppercase = "ABC";

Console.WriteLine("lowercase.ToUpper() = " + lowercase.ToUpper());
Console.WriteLine("uppercase.ToLower() = " + uppercase.ToLower());
```

```
lowercase.ToUpper() = ABC
uppercase.ToLower() = abc
```

به این نکته توجه کنید که اگر بخواهیم یک رشته به عنوان مثال یک جمله که از حروف بزرگ و کوچک تشکیل شده است، مثلاً حرف اول هر کلمه بزرگ و بقیه حروف کوچک نوشته شده باشند را با استفاده از متد ToUpper() تغییر دهیم فقط حروف کوچک آن تغییر کرده و بزرگ می‌شوند و حروفی که از قبل بزرگ بوده‌اند تغییر نمی‌کنند. این نکته در مورد متد ToLower() نیز صدق می‌کند.

حال بیایید متدی ایجاد کنیم که حرف اول کلمات هر رشته‌ای که به آن ارسال می‌شود را به صورت بزرگ و بقیه را به صورت کوچک تبدیل کند. در این برنامه از متدهای دستکاری رشته که تا به حال یاد گرفته‌ایم، استفاده شده است.

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static string ToTitleCase(string str)
        {
            string[] words = str.Split(' ');

            for (int i = 0; i < words.Length; i++)
            {
                string firstLetter = words[i].Substring(0, 1);
                string rest = words[i].Substring(1);
                string result = firstLetter.ToUpper() + rest.ToLower();
                words[i] = result;
            }

            return String.Join(" ", words);
        }

        static void Main()
        {
            string input;

            Console.WriteLine("Enter a string: ");
            input = Console.ReadLine();

            Console.WriteLine("Converting to Title Case...");
            input = ToTitleCase(input);

            Console.WriteLine(input);
        }
    }

}
```

```
Enter a string:
tHe quICK bROwN fOx
Converting to Title Case...
The Quick Brown Fox
```

در مثال بالا یک متد به نام ToTitleCase() ایجاد کرده‌ایم که یک رشته را به عنوان آرگومان قبول می‌کند.

```
string[] words = str.Split(' ');
```

ابتدا با استفاده از متد Split() رشته را به کلمات تشکیل دهنده‌اش تقسیم می‌کنیم، بنابراین می‌توان هر رشته را به صورت جداگانه دستکاری کرد. سپس با استفاده از یک حلقه for در میان کلمات گردش می‌کنیم.

```
string firstLetter = words[i].Substring(0, 1);
```

```
string rest = words[i].Substring(1);
```

سپس اولین حرف هر کلمه را با استفاده از متد Substring() استخراج و در یک متغیر برای استفاده‌های بعدی ذخیره می‌کنیم. سپس باقیمانده حروف کلمات را استخراج می‌کنیم.

```
string result = firstLetter.ToUpper() + rest.ToLower();
```

حال رشته‌ها را با هم ترکیب می‌کنیم البته از متد ToUpper() برای بزرگ کردن حروف اول و از متد ToLower() برای بزرگ کردن بقیه حروف کلمات استفاده می‌کنیم. بعد از این کار عناصر آرایه را با کلمات اصلاح شده جایگزین می‌کنیم.

```
return String.Join(" ", words);
```

حال کلمات اصلاح شده را ترکیب کرده و در بین آنها فضای خالی قرار می‌دهیم. سپس آنها را به متد فراخوان برگشت می‌دهیم.

قالب بندی رشته‌ها

دات نت جهت قالب بندی رشته و نحوه نمایش آنها در خروجی راه‌های متعددی در اختیار شما قرار می‌دهد. شما می‌توانید با استفاده از String.Format() رشته‌ها را به روش‌های مختلفی قالب بندی کنید.

```
string str1 = "This";
string str2 = "That";
string str3 = String.Format("{0} and {1}", "This", "That");

Console.WriteLine(str3);
```

```
This and That
```

String.Format() متدی است که یک رشته شامل قالب بندی خاص {۰} و {۱} را قبول می‌کند به دنبال این رشته، رشته‌ای که قرار است قالب بندی شود می‌آید. کامپایلر به اعداد رشته‌ای که دارای قالب بندی خاص است (“{0} and {1}”) نگاه می‌کند و آرگومان معادل را جایگزین این اعداد می‌کند. مثلاً در مثال بالا رشته This جایگزین عدد ۰ و رشته That جایگزین عدد ۱ می‌شود. به این نکته توجه کنید که String.Format() کاملاً شبیه متد Console.WriteLine() است.

در حقیقت Console.WriteLine() با همان تکنیکی که در بالا توضیح داده شد به کار می‌رود. از آنجاییکه علائم { و } برای نشان دادن شروع و پایان رشته قالب بندی به کار می‌روند، پس چگونه این علائم را می‌توان در خروجی نمایش داد؟

```
Console.WriteLine("{{{0}}}", 7);
```

```
{7}
```

برای نمایش هر کدام از این علائم کافیست آنها را در بین دو آکولاد مانند بالا قرار دهید. به کد بالا توجه کنید، مشاهده می‌کنید که عدد صفر نشان دهنده اولین آرگومان بعد از رشته با قالب بندی خاص یعنی عدد ۷ است. در نتیجه عدد ۷ همراه با آکولاد در خروجی چاپ می‌شود. به این نکته

توجه کنید که عدد 0 در رشته با قالب بندی خاص نشان دهنده اولین آرگومان، عدد ۱ نشان دهنده دومین آرگومان و ... بعد از این رشته خاص می‌باشد. به مثال زیر توجه کنید:

```
Console.WriteLine("{0} , {1} , {2}","One",5,"seven");
```

```
One , 5 , seven
```

در مثال بالا رشته با قالب خاص "{2} , {1} , {0}" می‌باشد. عدد ۰ در مثال بالا عدد ۰ با رشته "One" و عدد ۱ با عدد ۵ و عدد ۲ با رشته "seven" جایگزین می‌شود. یک آرگومان را بیشتر از یک بار می‌توان در خروجی چاپ کرد. به مثال زیر توجه کنید:

```
Console.WriteLine("{0} {0} {0} {1} {1}", "hello", "world");
```

```
hello hello hello world world
```

قالب بندی اعداد

می‌توان اعداد را نیز با فرمت‌های خاصی قالب بندی کرد:

```
Console.WriteLine("{0:C}", 500);
```

```
$500.00
```

همانطور که می‌بینید می‌توان عدد صحیح ۵۰۰ را با استفاده از C به حالت پولی در آورد. برای این کار می‌توان از فرمت "{0:C}" استفاده کرد. برای اعشاری کردن عدد ۵۰۰ هم می‌توان از فرمت "{0:F3}" به صورت زیر استفاده کرد:

```
Console.WriteLine("{0:F3}", 500);// 500.000
```

همانطور که می‌بینید فرمت "{0:F3}" سه رقم اعشار به عدد ۵۰۰ اضافه می‌کند. برای اعمال دو رقم اعشار کافیست عدد ۳ را به ۲ تغییر دهید. در جدول زیر لیستی از فرمت‌های مشخص (format specifiers) برای قالب بندی اعداد آمده است:

Specifier	Format
C	مقدار پولی خاص یک محل
D	انواع صحیح(integer)
E	نماد علمی
F	نقطه اعشار ثابت
G	اعداد عمومی
N	نقطه اعشار ثابت با جدا کننده کاما
P	اعداد دارای درصد

X	هگزا دسیمال

تأثیر دقت یک قالب خاص بستگی به تنظیمات منطقه‌ای دارد. به عنوان مثال قالب پولی C، به طور خودکار قالب پولی منطقه انتخاب شده را نشان می‌دهد. برای اکثر کاربران این اطلاعات به طور پیش‌فرض مربوط به منطقه و زبان آنها می‌باشد. در برنامه زیر چندین قالب عددی نشان داده شده است:

```
double v = 17688.65849;
double v2 = 0.15;
int x = 21;

Console.WriteLine("{0:F2}", v);
Console.WriteLine("{0:N5}", v);
Console.WriteLine("{0:e}", v);
Console.WriteLine("{0:r}", v);
Console.WriteLine("{0:p}", v2);
Console.WriteLine("{0:X}", x);
Console.WriteLine("{0:D12}", x);
Console.WriteLine("{0:C}", 189.99);
```

```
17688.66
17,688.65849
1.768866e+004
17688.65849
15.00 %
15
000000000021
$189.99
```

به تأثیر اعشار در قالب‌های مختلف توجه کنید. به این نکته هم توجه کنید که با استفاده از متد ToString() هم می‌توان قالب بندی عددی اعمال کرد.

```
int x = 500;
Console.WriteLine(x.ToString("C"));
```

```
$500.00
```

فاصله خالی دادن با قالب‌ها

با استفاده از قالب‌های مشخصی می‌توان فاصله‌هایی به چپ و راست رشته‌ها اضافه کرد. به مثال زیر توجه کنید:

```
Console.WriteLine("{0,10}", "hello");
Console.WriteLine("{0,-10}{1}", "hello", "world");
```

```
     hello
hello     world
```

همانطور که مشاهده می‌کنید در مثال بالا از قالب {x:y} استفاده شده است که در آن x آرگومان عددی و به معنای رشته مورد نظر ما و y طول فاصله است. مثلاً در خط دوم کد بالا {0,-10}{1} یعنی بعد از رشته Hello که عدد ۰ نماینده آنست و قبل از رشته World که عدد ۱ نماینده آن است ۱۰ فضای خالی قرار بده. طول مثبت فاصله را به سمت چپ و طول منفی فاصله را به سمت راست رشته اضافه می‌کند. مقدار فاصله‌ها به طول رشته‌ها بستگی دارد. به عنوان مثال اگر طول یک رشته ۵ باشد و طول فاصله‌ها ۱۰، آنگاه فضای خالی که به سمت راست یا چپ رشته اعمال

می‌شود، ۵ فاصله است. یعنی طول رشته منهای طول فاصله. پس در نتیجه اگر طول فاصله کمتر از طول رشته باشد هیچ فضای خالی به سمت چپ یا راست اعمال نمی‌شود. می‌توانید قالب بندی عددی را با قالب بندی فاصله ترکیب کنید:

```
Console.WriteLine("{0,10:C}", 500);
```

```
    $500.00
```

به نکته‌ای در مورد مثال بالا توجه کنید که ما ۱۰ فضای خالی برای قالب بندی تعیین کرده‌ایم و در نتیجه چون طول عدد ۵۰۰، سه است، باید ۷ فاصله در سمت چپ آن اضافه شود ولی چون علامت $ هم یک فضا را اشغال کرده است و همچنین دو عدد اعشار هم به عدد اضافه می‌کند به جای ۷ فاصله با احتساب ممیز، ۳ فاصله خالی در سمت چپ عدد ۵۰۰ قرار می‌گیرد.

قالب بندی سفارشی اعداد

سی‌شارپ به شما اجازه قالب بندی اعداد به صورت سفارشی را نیز می‌دهد. برای این کار کاراکترهای خاصی در اختیار شما می‌گذارد که در جدول زیر لیست آنها را مشاهده می‌کنید:

کاراکتر	معنی
#	عدد
.	اعشار
,	جدا کننده سه رقمی
%	درصد
0	اضافه کردن صفر
;	اعداد مثبت، منفی و مقادیر صفر را با قالب‌های خاصی نمایش می‌دهد.
E0 E+0 E-0 e0 e+0 e-0	نماد علمی

کاراکتر نقطه (.)، مکان اعشار را مشخص می‌کند. علامت # می‌تواند در برگیرنده عددی باشد که قرار است در سمت چپ یا راست علامت اعشار قرار بگیرد. اگر این علامت در سمت راست علامت اعشار قرار بگیرد، دقت اعشار را مشخص می‌کند و ممکن است در صورت لزوم عدد را گرد کند و اگر در سمت چپ اعشار باشد، نشان دهنده قسمت صحیح عدد می‌باشد. اگر طول عدد از تعداد علامت‌های # که در سمت چپ علامت اعشار قرار دارند بیشتر باشد، کل عدد نشان داده می‌شود. به عنوان مثال اگر عدد مورد نظر شما ۱۲۳۴۵ باشد و شما از سه علامت ### قبل از اعشار استفاده کرده باشید همه عدد نشان داده می‌شود. اگر مقدار اصلی دارای اعشار باشد و شما در قالب بندی قسمت اعشار را قالب بندی نکنید، عدد اصلی گرد می‌شود. به عنوان مثال اگر مقدار اصلی ۱۲۳.۴۵ باشد و شما از ### استفاده کنید عدد ۱۲۳ نمایش داده می‌شود. برای قرار دادن صفر قبل از علامت اعشار (قبل از عدد) و یا بعد از قسمت اعشار باید از کاراکتر نقطه به صورت زیر استفاده کنید:

```
Console.WriteLine("{0:00##.#00}", 21.3); // 0021.300
```

برای اعداد بزرگ هم می‌توانید از علامت کاما استفاده کنید :

```
Console.WriteLine("{0:#,###.#}", 3421.3); // 3,421.3
```

لازم نیست که مکان کاما را برای همه قسمت‌ها مشخص کنید چون خودش به صورت خودکار سه رقم سه رقم جدا می‌کند مثلاً :

```
Console.WriteLine("{0:#,###.#}", 8763421.3); // 8,763,421.3
```

کاما معنای دیگری هم دارد. وقتی که قبل از علامت اعشار قرار می‌گیرد به عنوان یک مقیاس عمل می‌کند. هر کاما مقدار را بر ۱۰۰۰ تقسیم می‌کند :

```
Console.WriteLine("{0:#,###,.#}", 8763421.3); // 8,763.4
```

همانطور که در خروجی مشاهده می‌کنید مقیاس در اینجا به معنی هزار است. ولی در مثال زیر همانطور که مشاهده می‌کنید در متد WriteLine()برای نمایش خروجی به صورت دقیق استفاده شده است:

```
Console.WriteLine("Fuel efficiency is {0:##.# mpg}", 21.3); // 21.3 mpg
```

می‌توان از کاراکترهای کنترلی \t یا \n در صورت لزوم استفاده کرد. کاراکتر E و e برای نمایش اعداد به صورت نماد علمی به کار می‌روند. حداقل یک صفر و در صورت لزوم تعداد بیشتری صفر بعد از E و e می‌آیند. صفرها تعداد ارقام اعشار را نشان می‌دهند. استفاده از حرف E و یا e ممکن است باعث چاپ این حروف در خروجی شوند. در نتیجه برای تعریف توان مثبت و منفی می‌توان از علامت - و + بعد از این دو کاراکتر به صورت E+، e+ و یا E-، e- استفاده کرد. علامت ";" هم شما را قادر می‌سازد که اعداد مثبت، منفی و مقادیر صفر را با قالب‌های خاصی جدا سازید. شکل کلی سفارشی سازی با این کاراکتر به صورت زیر است:

```
Console.WriteLine("{0:#.##;(#.##);0.00}", num);
```

در مثال بالا اگر num مثبت باشد مقدار با دو رقم اعشار، در غیر اینصورت اگر منفی باشد مقدار با دو رقم اعشار در داخل پرانتز و اگر صفر باشد به صورت 0.00نمایش داده می‌شود. مثال زیر کاربرد کاراکترهای ذکر شده را نشان می‌دهد:

```
double num = 64354.2345;

Console.WriteLine("{0:#.##}", num);
Console.WriteLine("{0:#,###.##}", num);
Console.WriteLine("{0:#.###e+00}", num);
Console.WriteLine("{0:#0,}", num);
Console.WriteLine("{0:#.#;(#.##);0.00}", num);
num = -num;
Console.WriteLine("{0:#.#;(#.##);0.00}", num);
num = 0.0;
Console.WriteLine("{0:#.#;(#.##);0.00}", num);
num = 0.95;
Console.WriteLine("{0:#%}", num);
```

```
64354.23
64,354.23
6.435e+04
64
```

```
64354.2
(64354.23)
0.00
95%
```

کلاس StringBuilder

کلاس String می‌تواند با استفاده از عملگر + دو رشته را به هم متصل کند. اما این عملگر برای الحاق دو رشته مختلف کارا نیست. چون شیء رشته در دات‌نت تغییر ناپذیر است. یعنی وقتی که یک متغیر از نوع رشته را تعریف می‌کنیم، مقدار آن تغییر نمی‌کند. هنگامی که یک رشته را به یک رشته موجود می‌چسبانید، مقدار قبلی حذف و یک شیء جدید که شامل دو رشته به هم چسبیده است به وجود می‌آید:

```
string Animal = "Dog";
Animal = "Cat";
Console.WriteLine(Animal);
```

```
Cat
```

با هر بار انجام این فرایند اشیاء موقتی ایجاد و اشیاء قدیمی از بین می‌روند و این ایجاد و حذف شدن‌ها هم زمان هستند و هم حافظه را اشغال می‌کنند. به مثال زیر توجه کنید، مثلاً اگر بخواهید همه اعداد ۰ تا ۹۹۹۹ را به هم بچسبانید:

```
int counter = 9999;
string s = string.Empty;

for (int i = 0; i <= counter; i++)
{
   s += i.ToString();
}
Console.WriteLine(s);
```

ممکن است با نگاه کردن به کد متوجه هیچ مشکلی نشوید. اما اگر از شیء Stopwatch استفاده کنید، متوجه می‌شوید که اجرای این برنامه چقدر زمان می‌برد. کد بالا را به صورت زیر اصلاح می‌کنیم:

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int counter = 9999;
            System.Diagnostics.Stopwatch sw =
            new System.Diagnostics.Stopwatch();
            sw.Start();

            string s = string.Empty;

            for (int i = 0; i <= counter; i++)
            {
                s += i.ToString();
            }

            sw.Stop();
            Console.WriteLine(s);
            Console.WriteLine("Took {0} ms", sw.ElapsedMilliseconds);
```

```
        }
    }
}
```

با استفاده از متد Start() کلاس Stopwatch زمان شروع و اجرای حلقه استفاده می‌کنیم. بعد از پایان حلقه متد Stop() از کلاس Stopwatch را فراخوانی می‌کنیم. به طور میانگین اجرای این برنامه 374ms زمان می‌برد (البته این زمان ممکن است برای شما متفاوت باشد). حال اجازه دهید که با استفاده از کلاس StringBuilder این اعداد را به هم وصل کنیم. این کار را با استفاده از متد Append() این کلاس انجام می‌دهیم.

```
using System;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int counter = 9999;
            System.Diagnostics.Stopwatch sw =
            new System.Diagnostics.Stopwatch();
            sw.Start();

            StringBuilder sb = new StringBuilder();

            for (int i = 0; i <= counter; i++)
            {
                sb.Append(i.ToString());
            }

            sw.Stop();
            Console.WriteLine(sb.ToString());
            Console.WriteLine("Took {0} ms", sw.ElapsedMilliseconds);
        }
    }
}
```

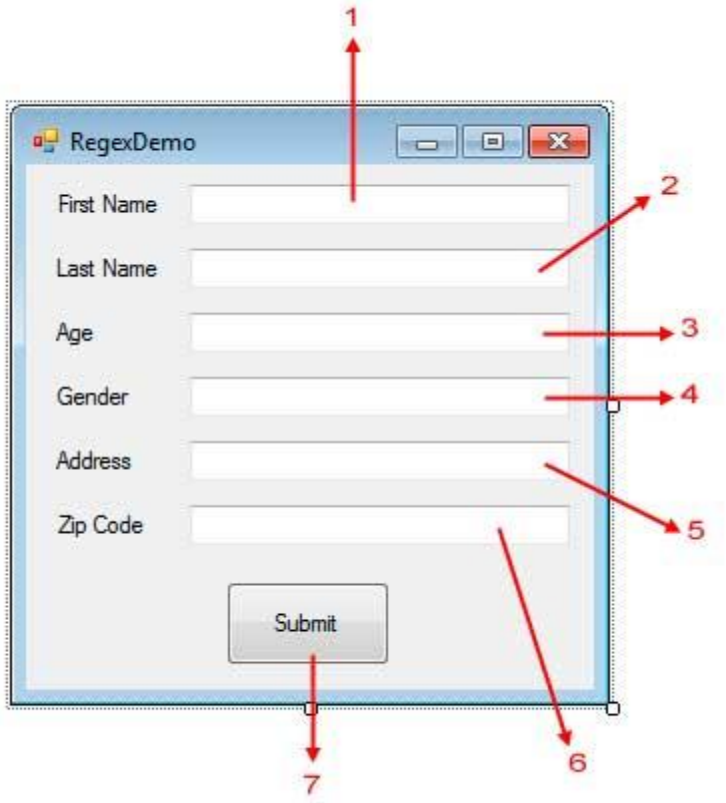
مشاهده می‌کنید که اجرای برنامه حدود 3ms طول می‌کشد. همانطور که مشاهده کردید کارایی برنامه به طور چشمگیری بهبود یافت. کارایی برنامه با افزایش متغیر حلقه بیشتر می‌شود. StringBuilder برای کار و اتصال رشته‌های زیاد به هم بسیار قدرتمند است. این کلاس دارای متدهای مختلفی است که در زیر به برخی از آنها اشاره شده است.

متد	توضیح
Append	رشته معادل شیء که به عنوان آرگومان به متد ارسال می‌شود در انتهای نمونه قرار می‌دهد.
AppendFormat	رشته قالب بندی شده‌ای به انتهای نمونه اضافه می‌کند.
AppendLine	یک پایان دهنده خط یا یک رشته همراه با پایان دهنده خط پیش‌فرض به انتهای نمونه اضافه می‌کند.
CopyTo	کاراکترهایی از قسمت خاصی از نمونه به یک آرایه کاراکتری مشخص انتقال می‌دهد.
Insert	یک رشته را در مکان خاصی از نمونه قرار می‌دهد.

Remove	محدوده‌ای از کاراکترهای یک نمونه را حذف می‌کند.
Replace	یک کاراکتر یا رشته را جایگزین تمام کاراکتر یا رشته‌های خاصی از نمونه می‌کند.
ToString	مقدار StringBuilder را به نوع String تبدیل می‌کند.

اعتبار سنجی با استفاده از عبارات باقاعده

استفاده از عبارات با قاعده یکی از راه‌های عالی برای اعتبار سنجی داده‌هایی است که توسط کاربر وارد فیلد یک فرم می‌شوند. فرض کنید که یک کاربر در فیلد مربوط به سن، نام و در فیلد مربوط به آدرس، جنسیت خود را وارد کند، در این حالت شما می‌توانید با استفاده از عبارات باقاعده از بروز خطا جلوگیری کنید. با استفاده از برنامه زیر که یک برنامه ویندوزی است، نحوه استفاده از عبارات با قاعده و اعتبار سنجی فیلدهای یک فرم نشان داده شده است. یک برنامه ویندوزی ایجاد کنید و نام آن را RegexValidation بگذارید.



خاصیت مقدار	نام	نوع	Label
	firstNameTextBox	Label	1
	lastNameTextBox	Label	2
	ageTextBox	Label	3
	genderTextBox	Label	4

5	Label	addressTextBox		
6	Label	zipCodeTextBox		
7	Button	submitButton	Text	Submit

با وجودیکه می‌توان از کنترل radio buttons برای مشخص نمودن جنسیت کاربر استفاده شود ولی در اینجا برای آموزش استفاده از عبارات باقاعده و اعتبار سنجی، از کنترل textBox استفاده می‌کنیم. بر روی submitButton دو بار کلیک کنید تا یک کنترل کننده رویداد برای رویداد کلیک آن ایجاد شود. مطمئن شوید که در قسمت تعاریف فضاهای نامی، فضای نام System.Text برای استفاده ازکلاس StringBuilder و System.Text.RegularExpressions برای استفاده از کلاس Regex وارد شده باشند.

```
using System.Text;
using System.Text.RegularExpressions;
```

حال باید اعضای StringBuilder و Regex را در کلاس برنامه مان تعریف کنیم.

```
private StringBuilder errors;
private Regex validator;
```

کد زیر را در قسمت کنترل کننده رویداد کلیک مربوط به کنترل submitButton بنویسید.

```
private void submitButton_Click(object sender, EventArgs e)
{
   if (AreFieldsValid())
   {
      //Do task here
   }
}
```

با کلیک بر روی این کنترل متد AreFieldsValid() فراخوانی می‌شود. این متد را برای چک کردن هر فیلد جهت یافتن خطا ایجاد می‌کنیم. این متد هنگامی که خطایی وجود نداشته باشد، مقدار true و اگر خطایی وجود داشته باشد، مقدار false را بر می‌گرداند. کد این متد به صورت زیر است:

```
1   private bool AreFieldsValid()
2   {
3          errors = new StringBuilder();
4
5          validator = new Regex(@"^([A-Z][a-z]+)(\s[A-Z][a-z]+)*$");
6
7          if (!validator.Match(firstNameTextBox.Text).Success)
8                 errors.AppendLine("First name is not in the proper format.");
9
10         if (!validator.Match(lastNameTextBox.Text).Success)
11                errors.AppendLine("Last name is not in the proper format.");
12
13         validator = new Regex(@"^\d{1,2}$");
14
15         if (!validator.IsMatch(ageTextBox.Text))
16                errors.AppendLine("Invalid Age.");
17
18         validator = new Regex(@"^([M|m]ale|[F|f]emale)$");
19
20         if (!validator.IsMatch(genderTextBox.Text))
```

```
21                  errors.AppendLine("Invalid Gender.");
22
23          validator = new Regex(@"^[0-9]+(\s[a-zA-Z]+)+$");
24
25          if (!validator.IsMatch(addressTextBox.Text))
26                  errors.AppendLine("Address is not in the proper format.");
27
28          validator = new Regex(@"^\d{4}$");
29
30          if (!validator.IsMatch(zipCodeTextBox.Text))
31                  errors.AppendLine("Invalid zip code");
32
33          if (errors.ToString() == String.Empty)
34          {
35                  return true;
36          }
37          else
38          {
39                  MessageBox.Show(errors.ToString(), "Validation Failed",
40                          MessageBoxButtons.OK, MessageBoxIcon.Error);
41
42                  return false;
43          }
44  }
```

در خط ۳ یک نمونه از کلاس StringBuilder ایجاد شده است. سازنده پیش‌فرض این کلاس یک رشته خالی را ایجاد می‌کند. در خط ۶ یک نمونه از عبارات با قاعده برای نام و نام خانوادگی ایجاد شده است. از آنجاییکه دو فیلد نام و نام خانوادی دارای نمونه یکسانی هستند پس می‌توان به راحتی از عبارت باقاعده مشابهی استفاده کرد. در خطوط ۷ و ۱۱ اعتبار نام و نام خانوادگی که توسط کاربر وارد شده است توسط متد Match() از کلاس Regex چک می‌شود.

متد Match() یک شیء از نوع کلاس Match را بر می‌گرداند. این کلاس یک خاصیت به نام success دارد که تعیین می‌کند که رشته موجود با الگو مطابقت دارد یا خیر. در صورتیکه رشته با الگو مطابقت نداشته باشد با استفاده از متد AppendLine() کلاس StringBuilder یک پیغام به متغیر errors اضافه می‌کنیم. این پیغام بعد از پایان اعتبار سنجی به کاربر نمایش داده می‌شود. در خط ۱۳ شیء Regex دیگری با الگوی جدید برای اعتبار سنجی سن ایجاد می‌کنیم. در خط ۱۵، از متد IsMatch() کلاس Regex به جای متد Match استفاده کرده‌ایم، هدف هر ۲ متد یکسان است. اگر رشته با الگو مطابقت داشته باشد مقدار true و در غیر این صورت مقدار false بر گردانده می‌شود. در حالت کلی برای هر فیلد یک شیء Regex با الگوی متفاوت می‌سازیم. اگر فیلد با الگو مطابقت نداشته باشد یک پیغام خطا به متغیر errors اضافه می‌شود. بعد از اینکه تمامی فیلدها را اعتبار سنجی کردیم، در خط ۳۳ بررسی می‌کنیم که آیا متغیر errors خالی است یا نه. اگر مقدار errors، empty باشد نمایانگر این است که تمامی فیلدها با الگوهایشان مطابقت دارند، در غیر این صورت ممکن است پیغام خطایی به شکل زیر به کاربر نمایش داده شود.

با روش اعتبار سنجی با استفاده از کلاس regular expression آشنا شدید. تکنیک‌های بیشتری برای اعتبار سنجی وجود دارد، از جمله استفاده از کنترل ErrorProvider.

File System

بیشتر اوقات لازم است که داده‌هایتان را در یک فایل دائمی ذخیره کنید. همچنین لازم است که به فایل‌ها یا پوشه‌های کامپیوتر دسترسی داشته باشید به طوری که بتوانید فایل‌ها را در آنها قرار داده و ذخیره کنید، مکانی را که می‌خواهید فایل در آن ذخیره شود را نشان داده، فایلی که در یک پوشه ذخیره شده را مشاهده کرده و به طور خلاصه جزییاتی در مورد یک پوشه خاص را به دست آورید. شما حتی می‌توانید چک کنید که اندازه فایل چقدر است و آیا فایل از نوع فقط خواندنی است یا نه؟ اگر چه استفاده از پایگاه داده امروزه بیشتر ترجیح داده می‌شود، اما گاهی اوقات برای برنامه‌های کوچک استفاده از یک فایل متنی کاراتر است. همچنین برخی از برنامه‌های کاربردی هنوز وجود دارند که داده‌هایشان را در یک فایل متنی ذخیره می‌کنند و بنابراین لازم است که اطلاعات در این فایل نوشته و از آن خوانده شود. دات‌نت دارای کلاس‌هایی که شما با استفاده از آنها می‌توانید بخوانید، بنویسید و صفات فایل‌ها و دایرکتوری‌ها را چک کنید. در درس بعد نحوه نوشتن یک داده باینری و فشرده سازی فایل‌ها که باعث کاهش حجم آنها می‌شود را، خواهید آموخت.

آدرس‌های مطلق و نسبی

فایل‌ها و دایرکتوری (پوشه) ها را با استفاده از مسیر (آدرس) شان می‌توان به صورت منحصر بفردی شناسایی کرد. مسیر (Path) نام مخصوص جایی است که فایل یا پوشه در آن قرار دارد. با استفاده از مسیر می‌توان از آدرس دهی مطلق یا منطقی استفاده کرد. آدرس مطلق نام کامل فایل یا پوشه است. آدرس مطلق با پوشه ریشه (Root) شروع می‌شود و به عنوان والد تمام زیر پوشه‌های مسیرهای موجود در خود هست و پوشه‌های دیگر والد پوشه‌های موجود در خود هستند تا به فایل مورد نظر برسند. به عنوان مثال آدرس " C:\Program Files\Tutorials\Sample.txt"، یک آدرس مطلق است. یا در آدرس دهی اینترنتی آدرس http://visualcsharptutorials/tutorials/sample.html، یک آدرس مطلق هست. در آدرس دهی نسبی، نیاز نیست تا پوشه ریشه مشخص شود. به راحتی می‌توان نام پوشه یا فایل دیگری را برای جستجو در مسیری که هستید استفاده کنید. برای مثال اگر در پوشه "C:\"

و به دنبال "Sample.txt" به عنوان آدرس نسبی هستید، در واقع آدرس"C:\Sample.txt" را فراخوانی کرده‌اید. یا مثلاً اگر فایل " Program Files/Sample.txt" فراخوانی کنید، برنامه به صورت خودکار مسیر این فایل را در پوشه " C:\" فراخوانی می‌کند (چون برنامه در C:\قرار گرفته است). در بیشتر مواقع، برای آدرس دهی از آدرس دهی کلمه به کلمه استفاده می‌شود. در این آدرس دهی مجبور به استفاده از ۲ بک اسلش هستیم که مقداری از خوانایی آدرس را کم می‌کند. برای رفع این مشکل می‌توان قبل از شروع آدرس دهی از یک "@" استفاده کرد. مثلاً آدرس "C:\Program Files\Tutorials\Sample.txt"را به دو صورت زیر می‌توان بیان نمود:

```
@"C:Program\FilesTutorials\Sample.txt"
```

```
"C:\\Program Files\\Tutorials\\Sample.txt"
```

در درس آینده در مورد کلاس‌های مختلفی از فضای نام System.IO که از آنها برای حذف و اضافه و ویرایش فایل‌ها و پوشه‌ها استفاده می‌شود، توضیح می‌دهیم.

فضای نام System.IO

فضای نام System.IO دارای کلاس‌های مختلفی است که از آنها برای انجام عملیاتی مانند ایجاد و حذف فایل‌ها، خواندن از فایل و نوشتن در آن و همچنین باز و بسته کردن فایل‌ها استفاده می‌شود. قبل از توضیح این کلاس‌ها ابتدا بهتر است با دو کلمه file و stream آشنا شوید.

یک File مجموعه ای از داده‌های ذخیره شده در دیسک با یک نام خاص، پسوند و یک مسیر می‌باشد. وقتی فایل را با C# برای خواندن و نوشتن باز می‌کنید، فایل تبدیل به Stream می‌شود. Stream در لغت به معنای جریان است و در مجموع به یک توالی یا سری از بایت‌ها که بین دو مسیر مبداء و مقصد در حال حرکت هستند، گفته می‌شود. برای کار با کلاس‌های فضای نام System.IO ابتدا باید این فضای نام را در قسمت تعریف فضاهای نامی به صورت زیر وارد کنید:

```
using System.IO
```

در جدول زیر لیست کلاس‌های معمول این فضای نام آمده است:

کلاس	توضیح
BinaryReader	برای خواندن داده از یک جریان باینری به کار می‌رود.
BinaryWriter	برای نوشتن در یک جریان باینری به کار می‌رود.
BufferedStream	یک محل ذخیره سازی موقت برای جریان از بایت‌ها می‌باشد.
Directory	برای ایجاد، حذف، انتقال و انجام اعمال مختلف بر روی پوشه‌ها به کار می‌رود.
DirectoryInfo	برای کار با پوشه‌ها به کار می‌رود.

DriveInfo	برای دریافت اطلاعاتی در مورد درایوها به کار می‌رود.
File	برای دستکاری فایل‌ها به کار می‌رود.
FileInfo	برای دستکاری فایل‌ها به کار می‌رود.
FileStream	برای خواندن از و نوشتن در فایل به کار می‌رود.
MemoryStream	برای دسترسی تصادفی به جریآنهای فایل ذخیره شده در Ram مورد استفاده قرار می‌گیرد.
Path	برای ایجاد مسیر فایل‌ها و پوشه‌ها به کار می‌رود.
StreamReader	برای خواندن کاراکتر از یک جریان بایتی به کار می‌رود.
StreamWriter	برای نوشتن کاراکترها در یک جریان به کار می‌رود.
StringReader	برای خواندن از یک استرینگ بافر استفاده می‌شود.
StringWriter	برای نوشتن روی یک استرینگ بافر استفاده می‌شود.

حال که با کلاس‌های این فضای نام آشنا شدید در درس‌های آینده در مورد کار با آنها توضیح می‌دهیم.

کلاس System.IO.File

کلاس System.IO.File یک کلاس استاتیک است، که دارای متدهایی برای دستکاری فایل‌های موجود در یک دایرکتوری می‌باشد. این متدها به شما اجازه کپی، حذف، بازکردن، انتقال و ایجاد یک فایل را می‌دهند. همانطور که می‌بینید ابتدا لازم است برای استفاده از این کلاس در قسمت فضای‌های نامی، فضای نام System.IO را وارد کنیم. برخی از متدهای پر کاربرد کلاس File در جول زیر آمده است:

متد	کاربرد
Copy	کپی کردن یک فایل در یک مقصد خاص
Create	ایجاد یک فایل در یک مسیر خاص
Delete	حذف فایل
Open	بازگشت یک شیء از FileStream که به وسیله آن شما می‌توانید یک فایل را بنویسید یا بخوانید.
Move	انتقال یک فایل به یک مقصد خاص.
Exists	چک کردن اینکه آیا یک فایل در یک مسیر خاص وجود دارد یا نه؟

ایجاد یک فایل

با استفاده از متد Create() می‌توان یک فایل جدید ایجاد کرد. این متد یک آرگومان از نوع رشته قبول کرده که این رشته مسیری است که فایل در آن ذخیره شده است:

```
File.Create(@"C:\Files\sample.txt");
```

نکته اینجاست که اگر فایلی از قبل به این نام وجود داشته باشد، حذف شده و فایل جدید جایگزین می‌شود. با استفاده از متد Exists() می‌توان تست کرد که آیا مسیری که مورد نظر شماست وجود دارد یا نه؟ این متد در صورت پیدا کردن مسیر فایل یا دایرکتوری، مقدار true و در غیر اینصورت مقدار false را بر می‌گرداند.

```
if (File.Exists(@"C:\Files\sample.txt"))
   Console.WriteLine("Path already exist.");
else
   Console.WriteLine("Path does not exist.");
```

حذف یک فایل

برای حذف یک فایل می‌توان با استفاده از متد Delete() و مسیری که فایل در آن قرار دارد این کار را انجام داد.

```
File.Delete(@"C:\Files\sample.txt");
```

کپی کردن یک فایل

برای کپی یک فایل می‌توان از متد Copy() که دارای دو پارامتر است، استفاده کرد. اولین پارامتر، مسیر کنونی فایل و دومین پارامتر، مقصدی است که فایل باید در آن کپی شود.

```
File.Copy(@"C:\Source\sample.txt", @"C:\Dest\sample.txt");
```

یکی دیگر از سربارگذاری‌های متد Copy() دارای یک پارمتر سومی و از نوع بولی است، و برای تشخیص این است که آیا فایل مورد نظر با فایل هم نام خود در مقصد جایگزین شود یا نه؟

```
File.Copy(@"C:\Source\sample.txt", @"C:\Dest\sample.txt", true);
```

انتقال یک فایل

برای انتقال یک فایل، از یک مسیر به مسیر دیگر می‌توانید از متد Move() که دو آرگومان مبداء و مقصد فایل را قبول می‌کند، استفاده کنید.

```
File.Move(@"C:\Source\sample.txt", @"C:\Dest\sample.txt");
```

باز کردن یک فایل

متد Open() برای باز کردن یک فایل به کار می‌رود. کلمه باز کردن به معنای اجرای فایل نیست. بلکه با فراخوانی این متد، یک شیء از FileStream را برگشت داده می‌شود که با استفاده از آن می‌توانیم یک فایل را در کلاس‌های دیگر بخوانیم و بنویسیم.

```
FileStream fs = File.Open(@"C:\Files\sample.txt", FileMode.Open);
```

نحوه ایجاد یک FileStream را در درس‌های آینده توضیح خواهیم داد. اولین سربارگذاری متد ()Open مسیر فایلی که قرار است باز شود و یک مقدار از نوع شمارشی System.IO.FileMode را، قبول می‌کند. استفاده از FileMode.Open بدین معناست که ما می‌خواهیم فایل را باز کنیم. در مورد این نوع شمارشی در درس‌های آینده توضیح خواهیم داد.

سربارگذاری دیگر این متد آرگومان سومی که یک مقدار از نوع شمارشی System.FileAccess است، را قبول می‌کند. این پارامتر اعلام می‌کند که آیا شما می‌خواهید عملیات خواندن، نوشتن یا هر دو را بر روی فایل انجام دهید. در مورد نوشتن و خواندن یک فایل در درس آینده توضیح خواهیم داد.

کلاس System.IO.FileInfo

این کلاس کاملاً شبیه کلاس File است و هر دوی آنها دارای یک هدف برای دستکاری فایل‌ها می‌باشند. تفاوت آنها این است که کلاس fileInfo دارای متد استاتیک نیست. بنابراین لازم است که ابتدا از آن نمونه ایجاد کنید. شیء ایجاد شده از FileInfo نشان دهنده یک فایل در دیسک است.

```
FileInfo file = new FileInfo(@"C:\Files\sample.txt");
```

سازنده مسیر فایل را قبول می‌کند. از آنجاییکه مسیر فایل را در داخل سازنده قرار داده‌ایم، فراخوانی متدهای ()Create، ()Delete، ()Move و ()Copy راحت تر است، چون هنگام فراخوانی مجبور نیستیم مسیر مذکور را در هر کدام از متدها دوباره نویسی کنیم. به عنوان مثال در کد زیر تفاوت بین کلاس File و FileInfo وقتی که متد ()Create فراخوانی می‌شود، نشان داده شده است:

```
FileInfo file = new FileInfo(@"C:\Files\sample.txt");
file.Create();
File.Create(@"C:\Files\sample.txt");
```

و در زیر تفاوت بین انتقال و کپی یک فایل نشان داده شده است:

```
file.CopyTo(@"C:\Dest\sample.txt");
file.MoveTo(@"C:\Dest\sample.txt");
File.Copy(@"C:\Source\sample.txt", @"C:\Dest\sample.txt");
File.Move(@"C:\Source\sample.txt", @"C:\Dest\sample.txt");
```

به این نکته توجه کنید که درشی FileInfo از ()CopyTo و ()MoveTo به جای ()Copy و ()Move استفاده کرده‌ایم و فقط مقصد را نشان داده‌ایم چون از قبل و هنگام ایجاد شیء، مبدأ را مشخص کرده‌ایم. برخی از متدهای کلاس FileInfo در جدول زیر ذکر شده‌اند.

متد	توضیح
Create	یک فایل جدید ایجاد می‌کند.
Delete	فایل وابسته به شیء FileInfo را حذف می‌کند.

CopyTo	فایل را در مسیر مقصد کپی می‌کند. یکی از سربارگذاری‌های این متد یک مقدار بولی دریافت می‌کند و نشان می‌دهد که آیا فایل موجود در مقصد را جایگزین کند.
MoveTo	فایل را به مقصد مشخصی انتقال می‌دهد.
Open	یک FileStream ایجاد می‌کند که می‌تواند در خواندن و نوشتن فایل مورد استفاده قرار گیرد.

برخی از خواص کلاس FileInfo در جدول زیر آمده است:

خاصیت	توضیح
CreationTime	زمان ایجاد فایل را نشان می‌دهد. نتیجه را به عنوان یک شیء DateTime بر می‌گرداند.
Directory	یک DirectoryInfo را بر می‌گرداند که شامل اطلاعات پوشه‌ای است که فایل در آن قرار دارد.
Exists	می‌گوید که آیا مسیر تعیین شده در سازنده در سیستم فایل موجود است یا نه؟
Extention	پسوند یک فایل را به صورت رشته بر می‌گرداند. (".txt")
IsReadOnly	می‌گوید که آیا فایل فقط خواندنی است یا نه؟
LastAccessTime	آخرین زمان دسترسی به فایل را نشان می‌دهد. مقدار را به عنوان یک شیء DateTime بر می‌گرداند.
LastWriteTime	آخرین زمانی که محتویات فایل دستکاری شده است را نشان می‌دهد. مقدار را به عنوان یک شیء DateTime بر می‌گرداند.
Length	سایز فایل را بر حسب بایت بر می‌گرداند.
Name	نام فایل را بر می‌گرداند(sample.txt) .
FullName	نام و مسیر کامل فایل را بر می‌گرداند ((C:\Files\sample.txt.

شیء FileInfo دارای خواص و متدهای بیشتری می‌باشد که به مرور در درس‌های آینده در مورد آنها بحث می‌کنیم.

کلاس System.IO.Directory

کلاس System.IO.Directory دارای متدهایی برای ایجاد، حذف، انتقال و انجام اعمال مختلف بر روی پوشه‌ها است. متدهای این کلاس استاتیک هستند و در نتیجه برای استفاده از آنها نیازی به ایجاد نمونه از کلاس نیست. در جدول زیر متدهای مفید کلاس Directory آمده است.

متد	توضیح
CreateDirectory	ایجاد یک پوشه جدید

Delete	حذف یک پوشه موجود
Exists	مشخص می‌کند که آیا یک پوشه در یک مسیر مشخص وجود دارد یا نه.
GetCurrentDirectory	پوشه فعلی را بر می‌گرداند.
GetDirectories	نام پوشه‌هایی را که در یک پوشه مشخص وجود دارند را بر می‌گرداند.
GetFiles	نام فایل‌هایی را که در یک پوشه مشخص وجود دارند را بر می‌گرداند.
GetSystemFileEntries	یک آرایه رشته‌ای که شامل نام فایل‌ها و پوشه‌های موجود در یک پوشه مشخص است را بر می‌گرداند.
GetParent	پوشه والد یک مسیر را بر می‌گرداند.
Move	پوشه را به مکان جدیدی منتقل می‌کند.

ایجاد یک پوشه

برای ایجاد یک پوشه از متد CreateDirectory() استفاده می‌شود. به تکه کد زیر توجه کنید:

```
Directory.CreateDirectory(@"C:\Directory\Subdirectory");
```

این متد یک آرگومان از نوع رشته که همان مسیر ایجاد پوشه است را دریافت می‌کند. متد CreateDirectory() همچنین همه پوشه‌ها و زیر پوشه‌ها را با استفاده از مسیری که به آن داده می‌شود ایجاد می‌کند.

حذف یک پوشه

اگر قصد حذف کردن یک پوشه را داشته باشید می‌توانید از متد Delete() استفاده کنید. این متد نیز یک آرگومان از نوع رشته دریافت می‌کند که همان مسیری است که پوشه دران قرار داد.

```
Directory.Delete(@"C:\Directory\Subdirectory");
```

در حالت پیش‌فرض متد Delete() اگر یک پوشه خالی نباشد یک استثناء (exception) را تولید می‌کند. اگر بخواهید همه فایل‌ها و زیر پوشه‌های یک پوشه را حذف کنید، باید از یکی دیگر از سربارگذاری‌های متد Delete() که دارای پارامتر دومی از نوع بولی است، استفاده کنید. اگر مقدار آن را true قرار دهید، هر چیزی که داخل پوشه مشخص شده است، حذف می‌شود.

```
Directory.Delete(@"C:\Directory\Subdirectory", true);
```

تست وجود یک پوشه

مانند کلاس File، کلاس Directory هم دارای متد Exists است که به کاربر اعلام می‌کند که آیا در یک مسیر مشخص پوشه مورد نظر وجود دارد یا نه؟

```
if (Directory.Exists(@"C:\Directory\Subdirectory"))
   Console.WriteLine("The directory exists.");
```

به دست آوردن لیست زیر پوشه‌ها و فایل‌ها

متد GetDirectories() همه زیر پوشه‌های یک مسیر خاص را در اختیار ما قرار می‌دهد. برای روشن شدن مطلب اجازه بدهید یک پوشه در درایو C ایجاد کنیم (C:\Directory) و پوشه‌هایی را در داخل این پوشه با نام‌های Subdirectory1، Subdirectory2، Subdirectory3 قرار دهیم. برای به دست آوردن لیست زیر پوشه‌ها به صورت زیر عمل می‌کنیم:

```
string[] directories = Directory.GetDirectories(@"C:\");

foreach (string directory in directories)
{
    Console.WriteLine(directory);
}
```

```
C:\Directory\Subdirectory1
C:\Directory\Subdirectory2
C:\Directory\Subdirectory3
```

اگر بخواهید فقط نام زیر پوشه‌ها را نشان دهید، می‌توانید به صورت زیر عمل کنید:

```
Console.WriteLine(directory.Substring(directory.LastIndexOf(@"\") + 1));
```

برای این کار از متدهای کلاس Path هم استفاده کنید که در درس‌های آینده در مورد آن توضیح خواهیم داد. متدهای GetFiles() و GetSystemFileEntries() بسیار شبیه به متد GetDirectory() هستند. GetFiles() یک آرایه رشته‌ای که شامل لیست فایل‌های داخل پوشه است را، بر می‌گرداند.GetSystemFileEntries() متشکل از لیست فایل‌ها و پوشه‌ها است.

انتقال یک پوشه

برای انتقال یک پوشه به یک مکان دیگر، می‌توان از متد Move() که دو آرگومان قبول می‌کند، استفاده نمود. اولین آرگومان نام پوشه مورد نظر و دومین آرگومان، مکانی است که پوشه باید به آنجا منتقل شود.

```
Directory.Move(@"C:\Directory", @"C:\AnotherDirectory\Directory");
```

کلاس System.IO.DirectoryInfo

از این کلاس برای کار با یک پوشه استفاده می‌شود. امکانات این کلاس شبیه به کلاس Directory است. برای استفاده از متدها و خواص کلاس DirectoryInfo لازم است که از آن نمونه ایجاد کنید.

```
DirectoryInfo directory = new DirectoryInfo(@"C:\Directory");
```

سازنده کلاس DirectoryInfo آرگومانی از نوع رشته که نشان دهنده مسیر پوشه است را، قبول می‌کند. این کلاس دارای متدهای متفاوتی برای ایجاد، حذف، انتقال و... پوشه‌ها می‌باشد. برخی از خواص مفید این کلاس در جدول زیر آمده است:

خاصیت	توضیح
Attributes	صفات یک پوشه را بر می‌گرداند.
CreationTime	زمان و تاریخی که یک پوشه ایجاد شده است را بر می‌گرداند.
Exists	اگر پوشه‌ای در سیستم فایل موجود باشد مقدار true را بر می‌گرداند.
LastAccessTime	تاریخ و زمان آخرین دسترسی به پوشه را بر می‌گرداند.
LastWriteTime	تاریخ و زمان آخرین دستکاری پوشه را نشان می‌دهد.
Name	نام پوشه را بر می‌گرداند.
FullName	مسیر کامل فایل یا پوشه را بر می‌گرداند.
Parent	پوشه والد یک زیر پوشه را بر می‌گرداند.
Root	ریشه یک مسیر را بر می‌گرداند.

خاصیت Parent نام پوشه والد یک شیء از DirectoryInfo را بر می‌گرداند. به عنوان مثال Directory پوشه والد C:\Directory\Subdirectory می‌باشد. خاصیت Root ریشه پوشه DirectoryInfo را برمی گرداند. مثلاً C:\ ریشه C:\Directory می‌باشد. برخی از متدهای این کلاس در زیر ذکر شده‌اند:

خاصیت	توضیح
Create	یک پوشه ایجاد می‌کند.
CreateSubdirectory	یک یا چند زیر پوشه در یک مسیر مشخص ایجاد می‌کند.
Delete	یک پوشه را حذف می‌کند.
EnumerateDirectories	یک لیست از اشیاء DirectoryInfo را که زیر پوشه‌های شیء DirectoryInfo هستند بر می‌گرداند.
EnumerateFiles	یک لیست اشیاء FileInfo که فایل‌های واقع در شیء DirectoryInfo هستند را بر می‌گرداند.
GetDirectories	شبیه EnumerateDirectories است ولی نتیجه را به عنوان یک آرایه بر می‌گرداند.
GetFiles	شبیه EnumerateFiles است ولی نتیجه را به عنوان یک آرایه بر می‌گرداند.
MoveTo	یک پوشه را به پوشه دیگر انتقال می‌دهد.

ایجاد زیر پوشه

از متد CreateSubdirectory() برای ایجاد زیر پوشه‌ها استفاده می‌شود. به مثال زیر توجه کنید:

```
DirectoryInfo directory = new DirectoryInfo(@"C:\Directory");
directory.Create(); //Creates C:\Directory
directory.CreateSubdirectory("Subdirectory"); //Creates C:\Directory\Subdirectory
```

مسیری که شما به عنوان آرگومان به متد CreateSubdirectory() ارسال می‌کنید یک مسیر نسبی است. بدین معنی که لازم نیست مسیر کامل زیر پوشه را بنویسید.

شمارش فایل‌ها و پوشه

می‌توان از متدهای EnumerateDirectories()، EnumerateFiles()، GetDirectories() یا GetFiles() برای به دست آوردن لیستی از پوشه‌ها و فایل‌ها استفاده نمود. تفاوت بین EnumereateDirectories() و EnumerateFiles() در برگشت نتیجه به عنوان یک System.Collections.Generic.IEnumerable<T> است به طوریکه T یا DirectoryInfo است یا FileInfo. متدهای GetDirectories() و GetFiles() نتیجه را به عنوان یک آرایه ساده DirectoryInfo یا FileInfo برگشت می‌دهند. کد زیر لیست فایل‌های موجود در یک پوشه را نشان می‌دهد:

```
DirectoryInfo directory = new DirectoryInfo(@"C:\");
IEnumerable<FileInfo> files = directory.EnumerateFiles();

foreach (FileInfo file in files)
{
     Console.WriteLine(file.Name);
}
```

خروجی نمایش داده نمی‌شود چون نتیجه بستگی به فایل‌هایی دارد که در مسیر C:\ directory وجود دارند. می‌توانید از یکی از سربارگذاری‌های چهار متد استفاده کنید که یک آرگومان به عنوان رشته قبول می‌کند. این رشته می‌تواند یک الگو و یا یک پسوند باشد. به عنوان مثال شما می‌توانید لیستی از همه فایل‌های با پسوند.txt را جستجو کنید:

```
IEnumerable<FileInfo> textFiles = directory.EnumerateFiles("*.txt");
```

کاراکتر * یک کاراکتر عمومی است که نشان دهنده نام فایل است. textFiles شامل لیستی از اشیاء FileInfo است که دارای پسوند.txt می‌باشند.

کلاس System.IO.Path

کلاس System.IO.Path کلاس مفیدی است که دارای متدهایی برای ایجاد مسیر فایل‌ها و پوشه‌ها می‌باشد. با وجودیکه می‌توان به سادگی از متدها و الحاق رشته‌ها برای نشان دادن مسیر استفاده کنید، اما استفاده از کلاس Path کار با مسیرها را برای شما آسان می‌کند. در جدول زیر برخی از متدهای کلاس Path ذکر شده‌اند.

متدها	توضیح
ChangeExtention	پسوند یک مسیر خاص را تغییرمی دهد.
Combine	آرایه ای از مسیرها را با هم ترکیب می‌کند.
GetDirectoryName	اطلاعات مربوط به پوشه را از رشته یک مسیر معین بر می‌گرداند.
GetExtension	پسوند را از رشته مسیر استخراج می‌کند.
GetFileName	نام فایل و پسوند را از رشته مسیر استخراج می‌کند.
GetFullPath	مسیر مطلق را بر می‌گرداند.
HasExtension	تعیین می‌کند که آیا یک مسیر پسوند دارد یا نه؟

ChangeExtension()

از متد ChangeExtension() برای تغییر پسوند یک مسیر مشخص شده استفاده می‌شود. فرض کنید مسیر C:\path.txt را در اختیار دارید. پسوند این مسیرtxt. است. کد زیر نشان می‌دهد که چطور پسوند یک مسیر را تغییر دهیم:

```
string path = Path.ChangeExtension(@"C:\path.txt", "bmp");
 Console.WriteLine(path);
```

```
C:\path.bmp
```

متد ChangeExtension() دو آرگومان از نوع رشته را قبول می‌کند و یک مسیر اصلاح شده از نوع رشته را بر می‌گرداند. اولین آرگومان، مسیر پسوندی است که می‌خواهیم تغییر دهیم و دومین آرگومان پسوندی است که می‌خواهیم جایگزین پسوند فعلی کنیم.

Combine()

از این متد برای ترکیب چندین مسیر استفاده می‌شود. به عنوان مثال، فرض کنید دو مسیر C:\Documents و sample.txt را در اختیار دارید و می‌خواهید این دو مسیر را با استفاده از متد Combine() ترکیب کنید، نتیجه ترکیب، C:\Documents\sample.txt خواهد شد. توجه کنید که بک اسلش (\) به طور خودکار در بین دو مسیر قرار می‌گیرد.

```
string path1 = @"C:\Document";
string path2 = @"sample.txt";

string newPath = Path.Combine(path1, path2);

Console.WriteLine(newPath);
```

```
C:\Document\sample.txt
```

به جای استفاده از الحاق رشته‌ها می‌توان از متد Combine() که ساده تر و کاراتر است، استفاده نمایید.

GetDirectoryName()

مسیر پوشه مادر یک فایل را بر می‌گرداند. به مثال زیر توجه نمایید:

```
string path1 = @"C:\Parent\sample.txt";
string path2 = @"C:\Parent\Child";

string parent1 = Path.GetDirectoryName(path1);
string parent2 = Path.GetDirectoryName(path2);

Console.WriteLine(parent1);
Console.WriteLine(parent2);
```

```
C:\Parent
C:\Parent
```

همانطور که در مثال بالا مشاهده می‌کنید با وجودی که فایل sample.text و پوشه Child در داخل پوشه Parent قرار دارند، متد GetDiretoryName() مسیر پوشه مادر (Parent) را بر می‌گرداند.

GetExtension()

این متد پسوند فایل که در یک مسیر مشخص وجود دارد را بر می‌گرداند:

```
string path1 = @"C:\Parent\sample.txt";
Console.WriteLine(Path.GetExtension(path));
```

```
.txt
```

GetFileName()

این متد نام کامل فایل (شامل پسوند) را بر می‌گرداند:

```
string path1 = @"C:\Parent\sample.txt";
Console.WriteLine(Path.GetFileName(path));
```

```
sample.txt
```

GetFullPath()

این متد مسیر کامل فایل را بر می‌گرداند، به مثال زیر توجه نمایید:

```
string path = "sample.txt";
Console.WriteLine(Path.GetFullPath(path));
```

```
C:\Users\VS\Documents\Visual Studio 2010\Projects\SimpleProgram\SimpleProgram\bin\Release\sample.txt
```

در مثال بالا، فایل sample.text در مسیر که برنامه در آن قرار دارد جستجو می‌شود، بدیهی است که اگر مسیر فایل اجرایی برنامه تغییر کند، خروجی متد بالا نیز تغییر خواهد کرد.

HasExtension()

اگر مسیر فایل شامل پسوند باشد خروجی این متد true و در غیر این صورت false می‌باشد، از این متد زمانی استفاده می‌شود که قصد داشته باشید تعیین کنید که آیا یک مسیر خاص به یک پوشه یا به یک فایل ختم می‌شود؟

کلاس FileStream

کلاس FileStream کلاسی از فضای نام System.IO است که از آن برای خواندن از، نوشتن در و بستن فایل استفاده می‌شود. این کلاس از کلاس انتزاعی Stream مشتق می‌شود. برای ایجاد و یا باز کردن یک فایل لازم است که یک شیء از این کلاس ایجاد کنید. دستور استفاده از کلاس FileStream به صورت زیر است:

```
FileStream object = new FileStream( file, FileMode, FileAccess, FileShare );
```

در جدول زیر پارامترهایی که سازنده این کلاس دریافت می‌کند، ذکر شده است:

پارامتر	توضیح
FileMode	این نوع شمارشی دارای مقادیر زیر برای ایجاد و باز کردن فایل‌ها می‌باشد: • Append اگر فایل موجود باشد آن را باز کرده و نشانگر ماوس را به انتهای آن می‌برد و اگر وجود نداشته باشد فایل را ایجاد می‌کند. • Create یک فایل جدید ایجاد می‌کند. • CreateNew یک فایل جدید ایجاد می‌کند. • Open یک فایل موجود را باز می‌کند. • OpenOrCreate یک فایل موجود را باز می‌کند و اگر وجود نداشته باشد آن را ایجاد می‌کند. • Truncate فایل موجود را باز می‌کند و سایز آن را به صفر بایت کاهش می‌دهد.
FileAccess	این نوع شمارشی دارای مقادیر Read, ReadWrite و Write برای خواندن و نوشتن فایل می‌باشد.
FileShare	این نوع شمارشی دارای مقادیر زیر می‌باشد: • None اشتراک گذاری فایل موجود را از بین می‌برد. • Read اجازه باز کردن فایل را برای خواندن می‌دهد. • ReadWrite اجازه‌ی باز کردن فایل را برای خواندن و نوشتن می‌دهد. • Write اجازه باز کردن فایل را برای نوشتن می‌دهد.

بهتر است که هنگام استفاده از این کلاس از دستور using استفاده کنید. استفاده از این دستور باعث فراخوانی متد Dispose() شده و با پاک کردن شیء ایجاد شده از کلاس FileStream باعث دسترسی دیگران به فایل می‌شود. اگر شیء ایجاد شده ازFileStream را پاک نکنید ارتباط با فایل قطع نمی‌شود و فایل تا زمانی که زباله روب (garbage collector) شیء را پاک نکند آزاد نمی‌شود.

```
using (FileStream Object = new FileStream())
{
    // Read from file or Write to file
}
```

روش دیگر برای از بین بردن شیء FileStream استفاده از دستور try و فراخوانی متد Close() به صورت زیر است:

```
FileStream Object = new FileStream();
try
{
    // Read from file or Write to file
}
finally
{
    Object.Close();
}
```

به کد زیر توجه کنید:

```
using System;
using System.IO;

namespace FileStreamDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (FileStream fileStream = new FileStream(@"C:\File.txt", FileMode.CreateNew))
            {
                Console.WriteLine("The File Create Successfully ");
            }
        }
    }
}
```

```
The File Create Successfully
```

همانطور که در کد بالا مشاهده می‌کنید ما یک شیء از کلاس FileStream در داخل دستور using ایجاد کرده‌ایم. سپس در داخل سازنده این کلاس گفته‌ایم که یک فایل به نام File.txt در داخل درایو C ایجاد کند. برای ایجاد فایل مقدار نوع شمارشی FileMode را برابر CreateNew قرار داده‌ایم. حال اگر درایو C را باز کنید مشاهده می‌کنید که فایلی با نامFile.txt ایجاد شده است. کد بالا را به صورت زیر هم می‌توان نوشت:

```
using System;
using System.IO;

namespace FileStreamDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream fileStream = new FileStream(@"C:\File.txt", FileMode.CreateNew);
            try
            {
                Console.WriteLine("The File Create Successfully ");
            }
            finally
```

```
            {
                fileStream.Close();
            }
        }
    }
}
```

```
The File Create Successfully
```

نوشتن در یک فایل متنی

نوشتن در یک فایل متنی راهی برای ذخیره دائمی داده‌ها می‌باشد. با وجودیکه بیشتر از دیتابیس‌ها و فایل‌های XML برای ذخیره اطلاعات استفاده می‌شود، اما فایل‌های متنی همچنان در برنامه‌های قدیمی مورد استفاده قرار می‌گیرند. البته اگر برنامه شما کوچک و داده‌های آن کم هستند بهتر است از فایل متنی برای ذخیره اطلاعاتتان استفاده کنید. قبل از نوشتن اطلاعات، به یک استریم نیاز داریم خصوصاً یک استریم فایل. یک استریم می‌تواند به یک فایل در روی سیستم شما یا سوکت شبکه یا... اشاره کند. کلاس FileStream بیانگر استریمی است که به یک فایل اشاره می‌کند. شیء از نوع FileStream به عنوان آرگومان، در سازنده کلاس StreamWriter استفاده می‌شود. کلاس StreamWriter مسئول نوشتن در استریم است، که در این مثال همان فایل است. ابتدا توجه کنید که کلاس‌های FileStream و StreamWriter در فضای نامی System.IO قرار دارند. نمونه برنامه زیر، اطلاعات نام، نام خانوادگی، سن را از کاربر دریافت می‌کند.

```
using System;
using System.IO;
using System.Text;

namespace WritingToFile
{
    class Program
    {
        static void Main()
        {
            try
            {
                FileStream fs = new FileStream("sample.txt", FileMode.Create);
                StreamWriter writer = new StreamWriter(fs);
                StringBuilder output = new StringBuilder();
                int repeat = 0;

                do
                {
                    Console.WriteLine("Please enter first name: ");
                    output.Append(Console.ReadLine() + "#");
                    Console.WriteLine("Please enter last name: ");
                    output.Append(Console.ReadLine() + "#");
                    Console.WriteLine("Please enter age: ");
                    output.Append(Console.ReadLine());
                    writer.WriteLine(output.ToString());
                    output.Clear();
                    Console.WriteLine("Repeat? 1-Yes, 0-No : ");
                    repeat = Convert.ToInt32(Console.ReadLine());
                } while (repeat != 0);

                writer.Close();
            }
            catch (IOException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
```

```
    }
}
```

```
Please enter first name: John
Please enter last name: Smith
Please enter age: 21
Repeat? 1-Yes, 0-No: 1
Please enter first name: Mike
Please enter last name: Roberts
Please enter age: 31
Repeat? 1-Yes, 0-No: 1
Please enter first name: Garry
Please enter last name: Mathews
Please enter age: 27
Repeat? 1-Yes, 0-No: 0
```

خروجی نشان می‌دهد که در مثال بالا مشخصات سه شخص از کاربر دریافت و آنها را در یک فایل متنی که در شیء FileStream مشخص شده می‌نویسد. در خط ۱۳ یک شیء از نوع FileStream ایجاد شده است. فایل در مسیری که برنامه اجرا می‌شود، ایجاد خواهد شد. سازنده کلاس FileStream همانطور که می‌بینید ۲ آرگومان دریافت می‌کند، یکی مسیر فایل و دیگری مقدار یک نوع شمارشی. نوع شمارشی تعیین می‌کند که سیستم عامل چگونه فایل را باز کند. مقادیر مجاز برای این نوع شمارشی را در پایین می‌بینید.

مقدار	توضیح
Append	اشاره گر را به انتهای فایل می‌برد و عمل نوشتن در آنجا آغاز می‌شود. اگر فایل وجود نداشته باشد آنرا ایجاد می‌کند.
Create	فایل را ایجاد می‌کند، اگر فایل وجود داشته باشد اطلاعات آن را حذف می‌کند.
CreateNew	یک فایل جدید ایجاد می‌کند. اگر فایل وجود داشته باشد یک استثناء از نوع System.IO.IOException رخ می‌دهد.
Open	یک فایل موجود را باز می‌کند. یک مقدار از نوع شمارشی System.IO.FileAccess رفتار این مقدار را مشخص می‌کند. اگر فایلی برای باز کردن وجود نداشته باشد یک استثناء از نوع System.IO.FileNotFoundException رخ می‌دهد.
OpenOrCreate	مشابه Open عمل می‌کند با این تفاوت اگر فایلی وجود نداشته باشد آن را ایجاد می‌کند.

ارسال FileMode.Create باعث می‌شود که یک فایل جدید ایجاد شود، اگر فایل موجود باشد اطلاعات آن دوباره نویسی می‌شود. اگر بخواهیم اطلاعاتی را به انتهای یک فایل موجود اضافه کنیم از FileMode.Append استفاده می‌کنیم. یک نوع شمارشی دیگر، System.IO.FileAccess است که عملیاتی را که قصد دارید بر روی فایل انجام دهید مشخص می‌کند. مقادیر آن در زیر نشان داده شده است:

مقدار	توضیح
Read	تنها مجاز به خواندن از فایل هستید.
Write	فقط اجازه نوشتن در فایل را دارید.

ReadWrite	مجاز به نوشتن و خواندن از فایل هستید.

برای استفاده از این نوع شمارشی، باید از سازنده دیگری از کلاس FileStream استفاده کنید که آرگومان دیگری از نوع شمارشی FileAccess می‌پذیرد.

```
FileStream fs = new FileStream("sample.txt", FileMode.Create, FileAccess.ReadWrite);
```

بعد از اینکه با موفقیت شیء FileStream را ایجاد کردیم باید آنرا به سازنده کلاس StreamWriter ارسال کنیم.

```
StreamWriter writer = new StreamWriter(fs);
```

کلاس StreamWriter متدهایی برای نوشتن اطلاعات در فایل دارد. در زیر آنها را مشاهده می‌کنید:

مقدار	توضیح
Write	داده‌ها را در فایل می‌نویسد
WriteLine	داده‌ها را در فایل می‌نویسد، سپس یک کاراکتر خط جدید در انتهای آن قرار می‌دهد.
Close	استریم را می‌بندد.

در خطوط ۲۵-۲۰ از کلاس StringBuilder برای تولید خروجی استفاده کرده‌ایم. به این نکته توجه کنید که در انتهای نام و نام خانوادگی یک کاراکتر # قرار داده‌ایم. به این دلیل هنگام خواندن اطلاعات از فایل تفکیک اطلاعات به آسانی انجام شود. شما می‌توانید به جای نوشتن این کاراکتر از هر کاراکتر خاص دیگر همچون $ استفاده کنید. بعد از نوشتن اطلاعات سرانجام باید استریم را ببندیم که این کار را در خط ۳۲ و با استفاده از متد ()Close انجام داده‌ایم تا منابع را از حافظه آزاد کند. همه‌ی خطوط را در یک بلوک try/catch قرار داده‌ایم زیرا این عملیات ممکن است یک استثنای System.IO.IOException تولید کند. به عنوان مثال، اگر فایل موجود نباشد این استثناء رخ می‌دهد. خطوط ۳۷-۳۴ این استثناء را گرفته و پردازش می‌کند. اجازه دهید به محتوای فایل نگاهی بیندازیم:

```
John#Smith#21
Mike#Roberts#31
Garry#Matthews#27
```

هر خط نمایانگر اطلاعات یک شخص است، اطلاعات مختلف توسط یک جداکننده از یکدیگر جدا شده است. در درس بعدی اطلاعات هر شخص را بازگردانی می‌کنیم.

خواندن از یک فایل متنی

عمل خواندن از یک فایل متنی بسیار شبیه به نوشتن در آن است. ابتدا باید یک FileStream که به فایل مورد نظر اشاره دارد ایجاد کنید. ولی به جای ارسال آن به کلاس StreamWriter باید آن را به کلاس StreamReader ارسال کنیم. برنامه زیر عمل خواندن از فایل را انجام می‌دهد. برنامه از فایل متنی که در درس قبلی ایجاد کرده‌ایم اطلاعات را می‌خواند.

```
using System;
using System.IO;
using System.Text;
using System.Collections.Generic;

namespace ReadingFromFile
{
    class Person
    {
        public string FirstName { get; set; }
        public string LastName  { get; set; }
        public int    Age       { get; set; }
    }

    class Program
    {
        static void Main()
        {
            List<Person> persons = new List<Person>();

            try
            {
                FileStream fs =
                   new FileStream("sample.txt", FileMode.Open, FileAccess.Read);
                StreamReader reader = new StreamReader(fs);

                while (!reader.EndOfStream)
                {
                    string line = reader.ReadLine();
                    string[] fields = line.Split('#');
                    Person newPerson = new Person();
                    newPerson.FirstName = fields[0];
                    newPerson.LastName  = fields[1];
                    newPerson.Age       = fields[2];
                    persons.Add(newPerson);
                }
            }
            catch (IOException ex)
            {
                Console.WriteLine(ex.Message);
            }

            //Show each person's details
            foreach (Person p in persons)
            {
                Console.WriteLine("Name: {0} {1}\nAge: {2}",
                    p.FirstName, p.LastName, p.Age);
            }
        }
    }
}
```

```
Name: John Smith
Age: 21
Name: Mike Roberts
Age: 31
Name: Garry Mathews
Age: 27
```

یک کلاس به نام Person (در خطوط ۱۳-۸) ایجاد کرده‌ایم. این کلاس اطلاعات مختلف هر شخص را که از فایل می‌خوانیم در خود نگهداری می‌کند. همچنین لیستی از اشخاص (در خط ۱۹) ایجاد تا اطلاعات همه‌ی آنها را در یک مکان جمع آوری کنیم. برای باز کردن و خواندن از فایل متنی (خط ۲۴) یک شیء از کلاس FileStream را ساخته و مسیر فایل، مقدار Open از نوع شمارشی FileMode و مقدار Read از نوع شمارشی FileAccess را به عنوان آرگومان به سازنده آن ارسال می‌کنیم. یک شیء StreamReader در خط ۲۵ ساخته و استریم ساخته شده را به عنوان

آرگومان به سازنده آن ارسال می‌کنیم. شیء StreamReader ساخته شده مسئول خواندن اطلاعات از فایل متنی است. در خط ۲۷ یک حلقه نوع while ساخته و با استفاده از خاصیت EndOfStream کلاس StreamReader بررسی می‌کنیم که اشاره گر به انتهای فایل نرسیده باشد.

در داخل حلقه از متد ()ReadLine کلاس StreamReader استفاده کرده‌ایم. این متد از مکان قرار گیری اشاره گر تا انتهای خط را می‌خواند. سپس اشاره گر را به ابتدای خط بعدی انتقال می‌دهد. تمامی خط خوانده شده را در متغیر از نوع رشته قرار می‌دهیم (خط ۲۹). سپس با استفاده از متد ()Split در خط ۳۰ و ارسال کاراکتر # به آن اطلاعات مختلف هر شخص را جدا و در یک آرایه‌ی رشته‌ای قرار می‌دهیم. یادآوری می‌شود که در درس قبلی از این کاراکتر (#) به عنوان جداکننده اطلاعات مختلف استفاده شده است. سپس یک شیء Person جدید ساخته و مقادیر موجود در عناصر آرایه را به خصوصیات شیء ایجاد شده اختصاص می‌دهیم (۳۴-۳۲)، و در انتها آن شیء را به لیست اشخاص اضافه می‌کنیم. هنگامی که کار خواندن از فایل به پایان رسید، با استفاده از یک حلقه اطلاعات تمامی اشخاص نمایش داده می‌شود.

فشرده کردن و از حالت فشرده در آوردن یک فایل متنی

بهترین ایده برای فایل‌هایی که حجم آنها زیاد شده است این است که آنها را فشرده کنیم. فشرده سازی پروسه کاهش حجم یک فایل است. این فرایند شامل یافتن ورودی‌های زائد و تکرار شده در یک فایل و تبدیل آنها به یک ورودی ساده است (یعنی اگر در یک فایل مثلاً ۱۰۰ بار کلمه Hello به کار رفته باشد آنها را به یک کلمه Hello در نظر می‌گیرد). کاهش حجم فایل‌ها در وب بیشتر کاربرد دارد. می‌توان با کاهش حجم صفحات وب سرعت بارگذاری یک وب سایت را بالا برد.

از این فرایند همچنین می‌توان برای گرفتن پشتیبان یا بایگانی فایل‌ها و نیز به اشتراک گذاشتن یک فایل در اینترنت (حجم فایل را کم کرده تا سریع‌تر آپلود و دانلود شود) استفاده نمود. از نرم افزارهای مختلفی مانند WinRar و WinZip می‌توان برای فشرده سازی استفاده نمود. اگر با فایل‌هایی که دارای پسوند .zip و .rar هستند مواجه شدید، بدانید که این فایل‌ها فشرده شده‌اند و حجم آنها در مقایسه با حالت عادی‌شان (غیر فشرده‌شان) کمتر است.

فشرده سازی فایل‌ها با استفاده از کلاس‌های GzipStream و DeflateStream

کتابخانه کلاس دات‌نت دارای کلاس‌ها و متدهایی جهت فشرده سازی و باز کردن فایل‌ها از حالت فشرده می‌باشد. این کلاس‌ها از الگوریتم GZIP یا Deflate جهت فشرده سازی فایل‌ها استفاده می‌کنند. در این درس می‌خواهیم یک فایل متنی را به راحتی فشرده کنیم.

شما می‌توانید برای فشرده کردن فایل‌ها و باز کردن آنها از حالت فشرده از کلاس GzipStream یا DeflateStream استفاده کنید. کارکرد این دو کلاس تقریباً شبیه هم است. این کلاس‌ها به یک شیء از FileStream که به یک فایل اشاره می‌کند، نیاز دارند. اجازه بدهید که نحوه استفاده به موقع از این کلاس‌ها را توضیح دهیم. در زیر دو مجموعه کد مربوط به دو کلاس ذکر شده است. یک برنامه کنسول با نام TextFileCompression ایجاد کنید. دو فضای نام لازم برای برنامه مان را وارد نمایید.

```
using System.IO;
using System.IO.Compression;
```

ابتدا حجم یک فایل را در حالت فشرده و غیر فشرده مقایسه می‌کنیم. کد زیر را برای ایجاد یک فایل متنی بدون هیچ فشردگی می‌نویسیم.

```
static void Main(string[] args)
{
    StringBuilder data = new StringBuilder();

    for (int i = 0; i < 1000; i++)
    {
        data.AppendLine("The quick brown fox jumps over the lazy dog.");
    }

    Console.WriteLine("A compressed file was created!");

    try
    {
        FileStream stream = new FileStream(@"C:\compressedFile.txt",
        FileMode.Create, FileAccess.Write);

        StreamWriter writer = new StreamWriter(stream);

        writer.Write(data.ToString());
        writer.Close();
    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

خطوط ۸-۳، ۱۰۰۰ خط ساختگی را در داخل فایل متنی می‌نویسد. خطوط ۱۲ تا ۲۵ یک فایل متنی را همانطور که در درس‌های قبلی یاد گرفتیم ایجاد می‌کنند. توجه کنید که ما تا الان هیچ عملی جهت فشرده کردن فایل انجام نداده‌ایم. همانطور که در خط ۱۴ مشاهده می‌کنید یک فایل متنی در C:\compressedFile.txt ایجاد می‌شود. اگر به مسیر بالا رفته و حجم فایل مورد نظر را چک کنید، مشاهده می‌کنید که حجم آن در حدود ۴۵ کیلو بایت است. حال اجازه بدهید که از کلاس‌های فضای نام System.IO.Compression جهت فشرده سازی استفاده کنیم. در مثال زیر از GzipStream و در مثال بعد از DeflateStream جهت فشرده سازی فایل مذکور استفاده می‌کنیم.

```
static void Main(string[] args)
{
    StringBuilder data = new StringBuilder();

    for (int i = 0; i < 1000; i++)
    {
        data.AppendLine("The quick brown fox jumps over the lazy dog.");
    }

    Console.WriteLine("A compressed file was created!");

    try
    {
        FileStream stream = new FileStream(@"C:\compressedFile.txt",
        FileMode.Create, FileAccess.Write);

        GZipStream gzip = new GZipStream(stream, CompressionMode.Compress);

        StreamWriter writer = new StreamWriter(gzip);

        writer.Write(data.ToString());
        writer.Close();
    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
```

```
28  }
```

این مثال تقریباً شبیه به مثال صفحه قبل است و تفاوت آنها در خطوط ۱۷ و ۱۹ می‌باشد. در خط ۱۷ شیء GzipStream از الگوریتم GZIP جهت فشرده سازی استفاده می‌کند. در داخل سازنده (خط ۱۷)، شیء ایجاد شده از FileStream یعنی straem را قرار می‌دهیم. دومین پارامتر سازنده، یک مقدار از نوع شمارشی CompressionMode است که شامل دو مقدار Compress و Decompress می‌باشد. برای ایجاد فایل‌های فشرده از مقدار CompressionMode.Compress استفاده می‌کنیم. در خط ۱۹ به جای ارسال FileStream به StreamWriter، شیء ایجاد شده GzipStream را ارسال می‌کنیم. وقتی که متد ()Write در خط ۲۱ را فراخوانی می‌کنیم، داده‌های نوشته شده به طور خودکار با استفاده از الگوریتم GZIP فشرده می‌شوند.

به یاد دارید که حجم فایل در حالت غیر فشرده در حدود 45kb بود. حال اجازه بدهید که نگاهی به سایز جدید فایل در حالت فشرده بیندازیم. مشاهده می‌کنید که حجم آن در این حالت 660 byte یا در حدود 1kb می‌باشد که 44KB کمتر از فایل غیر فشرده‌ای است که قبلاً ایجاد کردیم. مقدار کاهش یک فایل به مقدار زوائد و کل سایز فایل بستگی دارد. از آنجاییکه فایل اصلی شامل یک جمله است که ۱۰۰۰ بار تکرار شده است کار فشرده سازی فایل راحت تر می‌شود. اگر فایل متنی فشرده شده را باز کنید نمی‌توانید آن را بخوانید چون محتویات آن تغییر کرده است. همانطور که در مثال زیر مشاهده خواهید کرد کلاس DeflateStream بسیار شبیه به کلاس GzipStream می‌باشد.

```
static void Main(string[] args)
{
    StringBuilder data = new StringBuilder();

    for (int i = 0; i < 1000; i++)
    {
        data.AppendLine("The quick brown fox jumps over the lazy dog.");
    }

    Console.WriteLine("A compressed file was created!");

    try
    {
        FileStream stream = new FileStream(@"C:\compressedFile.txt",
        FileMode.Create, FileAccess.Write);

        DeflateStream deflate = new DeflateStream(stream, CompressionMode.Compress);

        StreamWriter writer = new StreamWriter(deflate);

        writer.Write(data.ToString());
        writer.Close();
    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

سایز فایل فشرده ایجاد شده توسط الگوریتم بالا در حدود 640bytes است که کمی کمتر از حالت فشرده آن به وسیله الگوریتم Gzip می‌باشد.

باز کردن فایل از حالت فشرده

برای خواندن فایل‌های فشرده باید آنها را باز کرد. برای این کار نیز می‌توان از کلاس‌های GzipStream و DeflateStream استفاده کرد. ابتدا نحوه استفاده از کلاس GzipStream برای باز کردن فایل‌ها از حالت فشرده را توضیح می‌دهیم. به این نکته توجه کنید که از یک کلاس صحیح برای باز کردن یک فایل فشرده استفاده کنید. مثلاً اگر یک فایل با استفاده از Deflate فشرده شده است باید از DeflateStream برای باز کردن آن از حالت فشرده استفاده کنید تا با خطا مواجه نشوید.

```
1   static void Main(string[] args)
2   {
3       try
4       {
5           FileStream stream = new FileStream(@"C:\compressedFile.txt",
6           FileMode.Open, FileAccess.Read);
7
8           GZipStream gzip = new GZipStream(stream, CompressionMode.Decompress);
9
10          StreamReader reader = new StreamReader(gzip);
11          string contents = reader.ReadToEnd();
12          reader.Close();
13
14          Console.WriteLine(contents);
15
16      }
17      catch (IOException ex)
18      {
19          Console.WriteLine(ex.Message);
20      }
21
22  }
```

فایلی که دارید از حالت فشرده خارج می‌کنید، اگر با کلاس GzipStream فشرده شده باشد و بخواهید آن را با استفاده از کلاس DeflateStream باز کنید با خطا مواجه می‌شوید. در خطوط ۶-۵ یک شیء FileStream ایجاد شده است که به یک فایل فشرده اشاره می‌کند. برای خواندن محتویات فایل از مقادیر FileMode.Open و FileAccess.Read در داخل سازنده استفاده می‌کنیم. در خط ۸ یک شیء از GzipStream ایجاد می‌کنیم.

تنها تفاوت اینجاست. همانطور که می‌بینید در آرگومان دوم می‌بینید تعیین مقدار compression mode است. در این خط (خط ۸) مقدار CompressionMode.Decompress را قرار می‌دهیم. از این مقدار (decompress) برای باز کردن فایل فشرده استفاده می‌شود. شیء GzipStream ایجاد شده از قبل را به شیء StreamReader در خط ۱۰ ارسال می‌کنیم تا محتویات فایل را بخواند. در خط ۱۱ از متد ReadToEnd() استفاده شده است که محتویات فایل فشرده را باز کرده و آن را به عنوان یک رشته بر می‌گرداند. در خط ۱۴ نیز محتویات فایل، که از حالت فشرده خارج شده‌اند، نمایش داده می‌شوند. اگر فایل را با استفاده از کلاس DeflateStream فشرده کرده‌اید، باید از همین کلاس هم برای باز کردن آن استفاده نمایید. در مثال زیر به شما نشان می‌دهیم که چگونه یک فایل را با استفاده از کلاس DeflateStream از حالت فشرده خارج کنیم. برای این کار، دقیقاً مانند استفاده از کلاس GzipStream عمل می‌کنیم.

```
static void Main(string[] args)
{
    try
    {
```

```
        FileStream stream = new FileStream(@"C:\compressedFile.txt",
                                FileMode.Open, FileAccess.Read);

        DeflateStream deflate = new DeflateStream(stream, CompressionMode.Decompress);

        StreamReader reader = new StreamReader(deflate);
        string contents = reader.ReadToEnd();
        reader.Close();

        Console.WriteLine(contents);

    }
    catch (IOException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

در این درس به شما نحوه فشرده کردن یک فایل ساده آموزش داده شد. شما می‌توانید از این کلاس‌ها برای فشرده سازی انواع مختلف فایل استفاده کنید.

زبان نشانه گذاری توسعه پذیر(XML)

زبان نشانه گذاری توسعه پذیر (XML) به شما اجازه می‌دهد که داده‌ها را در یک متن و قالب ساخت یافته ذخیره کنید. این زبان به طور گسترده، به عنوان یک دیتابیس جایگزین و برای ذخیره اطلاعات مربوط به پیکربندی نرم افزارها به کار می‌رود. XML از لحاظ دستوری شبیه به HTML بوده و اگر با HTML آشنایی داشته باشید یادگیری این زبان برایتان راحت تر است. در زیر یک سند XML را مشاهده می‌کنید:

```
<Persons>
  <Person>
    <Name>John Smith</Name>
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person>
    <Name>Mike Folly</Name>
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person>
    <Name>Lisa Carter</Name>
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
</Persons>
```

سند XML ترکیبی از عناصر XML می‌باشد. یک عنصر XML شامل یک تگ آغازی، یک تگ پایانی و داده‌ای است که در بین این دو تگ قرار می‌گیرد.

```
<open>data</close>
```

می‌توان بر اساس داده‌ای که یک عنصر XML در خود نگهداری می‌کند یک نام برای عنصر انتخاب کرد. به این نکته توجه کنید که عناصر به حروف بزرگ و کوچک حساس‌اند، بنابراین دو کلمه person و Person با هم متفاوت‌اند. XML فضاهای خالی را نادیده می‌گیرد، بنابراین به جای نوشتن

یک فایل در یک خط می‌توانید آن را در چند خط بنویسید تا خوانایی آن بالاتر رود. بین عناصر XML ممکن است رابطه پدر- فرزندی وجود داشته باشد

```
<parent>
   <child1>data</child1>
   <child2>
      <grandchild1>data</grandchild1>
   </child2>
</parent>
```

سند XML بالا دارای اطلاعاتی برای سه شخص می‌باشد. هر سند XML باید دارای یک عنصر ریشه (root) باشد. در مثال اول این درس، عنصر Persons، عنصر ریشه (پدر) و دیگر عناصر داخل آن در حکم فرزندان آن می‌باشند. جزییات هر شخص در داخل عنصر Person قرار دارند. عناصر فرزند عنصر Person عبارت‌اند از Name، Age و Gender. صفات XML، روشی دیگر برای اضافه کردن داده به یک عنصر می‌باشند.

```
<Person name="John Smith">some data</Person>
```

عنصر بالا یک خاصیت به نام name دارد که مقدار آن John Smith می‌باشد. مقادیر باید در داخل کوتیشن (' ') یا دابل کوتیشن (" ") قرار بگیرند. در زیر روش اضافه کردن صفات نشان داده شده است.

```
<element att1="value1" att2="value2" ... attN="valueN">data</element>
```

همانطور که مشاهده می‌کنید، می‌توان به یک عنصر چندین صفت اضافه کرد.

```
<Person name="John Smith" age="30" gender="Male">some data</Person>
```

اجازه دهید که به عناصر مثال ابتدای درس صفاتی اضافه کنیم.

```
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folly">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
</Persons>
```

عنصر Name هر شخص (person) را حذف و صفت معادل آن (name) را برای هر عنصر می‌نویسیم. اسناد XML می‌توانند دارای یک تعریف XML باشند. تعریف XML شامل اطلاعاتی درباره سند XML مانند نسخه (همیشه نسخه ۱,۰ پیشنهاد می‌شود) و نوع رمزگذاری (encode) متن آن می‌باشد.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

این تعریف در بالاترین بخش سند و درست قبل از عنصر اصلی نوشته می‌شود. برای فایل XML می‌توان توضیحات نیز نوشت. نحوه نوشتن توضیحات در XML به صورت زیر است.

```
<!-- This is an XML comment -->
```

می‌توان با استفاده از یک ویرایشگر متن ساده فایل‌های XML تولید کرد اما ویژوال استودیو دارای یک ویرایشگر XML قوی بوده که فایل را به صورت خودکار قالب بندی می‌کند و همچنین می‌توان از خاصیت IntelliSense برای سرعت بخشیدن به کارتان استفاده کنید. هنگام ایجاد یک پروژه جدید می‌توان بر روی دکمه Add New Item واقع در نوار ابزار کلیک کرده و از داخل پنجره ظاهر شده گزینه XML file را انتخاب کنید. ویژوال استودیو به صورت خودکار یک فایل xml همراه با تعریف آن ایجاد می‌کند.

XML Document Object Model

دات نت دارای کلاس‌هایی برای خواندن و دستکاری فایل‌های XML می‌باشد. این کلاس‌ها در فضای نامی System.Xml قرار دارند. دات‌نت از XML Document Object Model استفاده می‌کند، که شامل مجموعه‌ای از کلاس‌ها برای نمایش قسمت‌های مختلف یک سند XML می‌باشد. در زیر نام برخی از این کلاس‌ها ذکر شده است:

کلاس	توضیح
XmlNode	نشان دهنده یک گره در سند XML می‌باشد. هر چیزی که در داخل یک فایل XML می‌بینید مانند خواص عناصر، توضیحات و حتی فضاهای خالی یک گره می‌باشند. این کلاس، کلاس پایه دیگر کلاس‌های XML DOM می‌باشد.
XmlDocument	نشان دهنده یک سند واقعی XML می‌باشد که می‌تواند در بارگذاری یا ذخیره یک فایل XML مورد استفاده قرار گیرد.
XmlElement	یک عنصر XML را نمایش می‌دهد.
XmlAttribute	یک صفت XML را نمایش می‌دهد.
XmlText	متن داخل یک عنصر XML را نمایش می‌دهد.
XmlComment	یک توضیح XML را نمایش می‌دهد.

یک سند XML نماینده یک کلاس XmlDocument می‌باشد که می‌توانید آن را خوانده و دستکاری نمایید. سند XML باید با یک عنصر اصلی شروع شود. در داخل این عنصر اصلی فرزندهای آن قرار دارند. شما باید با مفهوم گره‌های پدر و فرزند عناصر آشنا شوید. یک گره پدر شامل گره‌های فرزند می‌باشد. به عنوان مثال به عنصر زیر که شامل چندین عنصر است توجه کنید:

```
<Product>
   <ProductID>001</ProductID>
   <ProductName>Shampoo</ProductName>
   <Price>10</Price>
</Product>
```

Product در مثال بالا گره پدر محسوب می‌شود. این گره شامل گره‌هایی مانند ProductID، ProductName و Price می‌باشد که فرزندان آن به حساب می‌آیند. اگر به عنوان مثال گره ProductName نیز دارای گره‌هایی در داخل خود باشد، خود ProductName یک گره پدر می‌شود، یعنی

این گره دو نقش دارد: یکی فرزند گره Product و دیگری پدر گره‌هایی که در داخل آن قرار دارند. اجازه دهید به توضیح برخی از خواص و متدهای این کلاس‌ها بپردازیم.

XmlNode

XmlNode نشان دهنده همه قسمت‌های XML بوده و کلاس پایه دیگر کلاس‌های XML DOM می‌باشد. به عنوان مثال کلاس XmlElement از XmlNode مشتق شده و دارای خواص و متدهای آن می‌باشد. در زیر برخی از خواص کلاس پایه XmlNode آمده است:

خصوصیت	توضیح
Attributes	مجموعه‌ای از اشیاء XmlAttribute که نماینده صفات گره هستند.
ChildNodes	لیستی از گره‌هایی که فرزند گره مورد نظر ما هستند را بر می‌گرداند.
FirstChild	اولین گره فرزند، گره جاری را بر می‌گرداند.
HasChildNodes	می‌گوید که آیا گره مورد نظر ما گره فرزند دارد یا نه؟
InnerText	متن همه گره‌های فرزند، گره مورد نظر را به صورت یک رشته ترکیب و نمایش می‌دهد.
InnerXml	رشته XML ی که بین دو تگ باز و بسته گره قرار دارد را بر می‌گرداند.
LastChild	آخرین گره فرزند، گره مورد نظر ما را بر می‌گرداند.
Name	نام گره را بر می‌گرداند.
NextSibling	گره‌ای که بعد از گره جاری آمده است را بر می‌گرداند.
NodeType	نوع گره جاری را نشان می‌دهد.
OuterXml	گره موجود و تگ‌ها و متن‌های داخل آن را بر می‌گرداند.
OwnerDocument	سند XML ی که گره به آن تعلق دارد را بر می‌گرداند.
ParentNode	گره والد یا پدر گره مورد نظر را بر می‌گرداند.
PreviousSibling	گره قبل از گره مورد نظر ما را برمی گرداند.
Value	مقدار گره مورد نظر ما را بر می‌گرداند.

خاصیت Attributes شامل لیستی از خواص یک گره یا عنصر می‌باشد. هر خاصیت به وسیله کلاس XmlAttribute نشان داده می‌شود. به مثال زیر توجه کنید.

```
<Person name="John Smith" age="30"></Person>
```

خاصیت Attributes عنصر بالا شامل دو شیء XmlAttribute که در بردارنده اطلاعاتی برای خواص name و age می‌باشند، می‌باشد. خاصیت ChildNodes مجموعه‌ای از گره‌های فرزند، گره جاری می‌باشند. هر گره فرزند یک شیء XmlNode می‌باشد. می‌توانید با چک کردن مقدار خاصیت HasChildNodes بفهمید که آیا یک گره دارای گره فرزند می‌باشد یا نه. اگر گره دارای یک یا چند زیر گره (گره فرزند باشد) مقدار true و اگر هیچ زیر گره‌ای نداشته باشد مقدار false برگشت داده می‌شود. برای به دست آوردن اولین و آخرین گره فرزند یک گره می‌توان به ترتیب از خواص FirstChild و LastChild و برای به دست آوردن گره قبل و بعد از یک گره می‌توان از خواص PreviousSibling و NextSibling استفاده نمود.

```
<PreviousNode></PreviousNode>
<CurrentNode></CurrentNode>
<NextNode></NextNode>
```

ParentNode، پدر یک گره را نشان می‌دهد. اگر گره هیچ پدری نداشته باشد این خاصیت مقدار تهی را بر می‌گرداند. خاصیت Name، نشان دهنده نام عنصر می‌باشد.

```
<Person>Example</Person>
```

نام عنصر بالا Person است. InnerText متن داخل دو تگ باز و بسته یک کنترل را بر می‌گرداند. اگر با توضیحات سر و کار داشته باشید، این خاصیت متن توضیحات را بر می‌گرداند. اگر یک عنصر دارای چندین گره در داخل خود باشد، خاصیت InnerText متن داخل همه گره‌های فرزند را گرفته، آنها را ترکیب و به صورت یک رشته ساده بر می‌گرداند. به کد XML زیر توجه کنید:

```
<Example>
   <Sample1>Text1</Sample1>
   <Sample2>Text2</Sample2>
   <Sample3>
      <Sample4>Text3</Sample4>
   </Sample3>
</Example>
```

خروجی خاصیت InnerText در مثال بالا، Text1Text2Text3 می‌باشد. به این نکته توجه کنید که سومین فرزند در مثال فوق (Sample3) خود دارای یک فرزند است که خاصیت InnerText متن داخل آن (Text3) را بر می‌گرداند. InnerXml شبیه InnerText است با این تفاوت که تگ‌های XML موجود در متن را هم بر می‌گرداند. OuterXml شبیه به InnerXml با این تفاوت که تگ‌های گره جاری و متن‌های داخل آن را بر می‌گرداند.

خاصیت OwnerDocument سند مرجع گره جاری را بر می‌گرداند. NodeType شامل مقداری از نوع شمارشی System.Xml.XmlNodeType است که به وسیله آن می‌توانید نوع یک گره را بگویید. در جدول زیر لیستی از مقادیر نوع شمارشی XmlNodeType آمده است:

مقدار	توضیح
Element	گره یک عنصر است.
Attribute	گره یک صفت است.

Text	گره یک متن است.
Comment	گره یک توضیح است.
Document	عنصر ریشه یک سند.
Whitespace	یک فضای سفید، مانند فضای خالی، خط جدید یا tab
XmlDeclaration	تعریف سند XML

خاصیت Value مقدار یک گره را مشخص می‌کند و در هر گره متفاوت است. به عنوان مثال خاصیت Value در یک توضیح، متن توضیح را بر می‌گرداند و در یک صفت، مقدار صفت. به این نکته توجه کنید که برای دسترسی به محتوای یک XmlElement باید از InnerText یا InnerXml استفاده کنید نه خاصیت Value.

قبل از معرفی متدهای کلاس XmlNode ابتدا بهتر است با XPath آشنا شوید. XPath زبانی برای یافتن اطلاعات در یک سند XML است. با استفاده از XPath می‌توان محل و موقعیت ساختار سند و یا داده‌های موجود در یک سند XML را مشخص نمود. هدف اولیه XPath، امکان آدرس دهی بخش‌های متفاوت یک سند XML است. بمنظور تأمین خواسته فوق از امکانات و پتانسیل‌های متعددی بمنظور انجام عملیات بر روی رشته‌ها، اعداد و منطق استفاده می‌شود. XPath از یک گرامر فشرده و عدم مبتنی بر XML بهمراه URI و مقادیر خصلت‌های XML استفاده می‌نماید. دلیل انتخاب نام XPath برای تکنولوژی فوق بدین علت است که در حقیقت از یک آدرس بمنظور حرکت در طول یک سند XML با ساختار سلسله مراتبی استفاده می‌گردد. XPath یک سند XML را بعنوان درختی از گره‌ها شبیه سازی می‌نماید. در این راستا، گره‌های متفاوتی نظیر گره‌های Element، گره‌های Attribute و گره‌های Text، وجود دارد. برای هر گره توسط XPath، یک رشته در نظر گرفته می‌شود. برخی از انواع خاص گره‌ها دارای اسامی اختصاصی خود می‌باشند. XPath بطور کامل XML Namespace را پشتیبانی می‌نماید. در جدول زیر متدهای مفید کلاس XmlNode آمده است:

متد	توضیح
AppendChild	یک کره فرزند به گره اضافه می‌کند.
InsertAfter	بعد از گره جاری یک گره ایجاد می‌کند.
InsertBefore	قبل از گره جاری یک گره ایجاد می‌کند.
PrependChild	یک گره را به ابتدای گره‌های فرزند گره مورد نظر ما اضافه می‌کند.
RemoveAll	گره‌های فرزند و خواص گره مورد نظر را حذف می‌کند.
RemoveChild	یک گره فرزند خاص مشخص را حذف می‌کند.
ReplaceChild	یک گره فرزند جدید را جایگزین یک گره فرزند قدیمی تر می‌کند.

SelectNodes	لیستی از گره‌هایی که با عبارت XPath مطابقت دارند را انتخاب می‌کند.
SelectSingleNode	اولین گره‌ای که با عبارت XPath مطابقت دارد انتخاب می‌کند.

کاربرد این متدها را در درس‌های آینده توضیح می‌دهیم.

XmlDocument

از کلاس XmlDocument برای نمایش سند XML و محتویات آن استفاده می‌شود. در زیر برخی از خواص این کلاس ذکر شده است:

خاصیت	توضیح
DocumentElement	عنصر ریشه سند را نشان می‌دهد.
PreserveWhitespace	می‌گوید که آیا فضاهای خالی سند XML حذف یا نگهداری شوند.

در زیر برخی از متدهای کلاس XmlDocument آمده است:

متد	توضیح
CreateAttribute	یک شئ XmlAttribute ایجاد می‌کند.
CreateComment	یک شئ XmlComment ایجاد می‌کند.
CreateElement	یک شئ XmlElement ایجاد می‌کند.
CreateTextNode	یک شئ XmlText با متن مشخص ایجاد می‌کند.
CreateXmlDeclaration	یک XmlDeclaration با مقدار مشخص ایجاد می‌کند.
GetElementById	یک عنصر XML با ID مشخص را بر می‌گرداند.
GetElementsByTagName	مجموعه‌ای از عناصری که با نام مشخصی مطابقت دارند را برگشت می‌دهد.
Load	یک سند XML را از یک فایل بارگذاری می‌کند.
Save	محتوای سند XML را در فایل مشخصی ذخیره می‌کند.

به متدهایی که با کلمه Create شروع شده‌اند، توجه کنید. از این متدها در ساخت اشیاء XML DOM استفاده می‌شود. به عنوان مثال، اگر می‌خواهید یک عنصر (element) برای سند جاری ایجاد کنید، می‌توانید از متد ()CreateElement استفاده نمایید. بیشتر متدهای بالا را در درس‌های بعدی توضیح می‌دهیم.

XmlElement

از کلاس XmlElement برای نمایش یک عنصر ساده XML استفاده می‌شود. در جدول زیر خواص پر کاربرد این کلاس ذکر شده است:

خاصیت	توضیح
Attributes	همه صفات یک عنصر را شامل می‌شود.
HasAttributes	می‌گوید که آیا عنصر شامل صفت است یا خیر؟
Value	متن بین تگ‌های باز و بسته عنصر مورد نظر را بر می‌گرداند. اگر عنصر مورد نظر ما دارای عنصر فرزندی باشد مقدار این خصوصیت خالی خواهد شد.

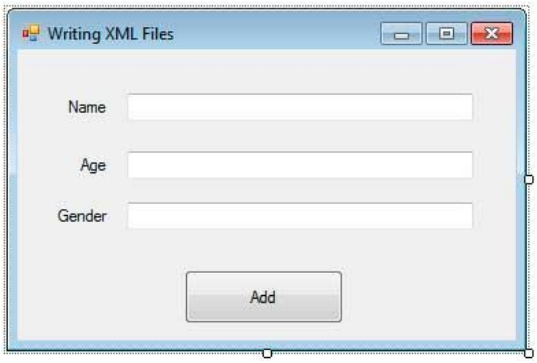
در جدول زیر برخی از متدهای XmlElement آمده است:

متد	توضیح
GetAttribute	مقدار یک صفت با نام مشخص را بر می‌گرداند.
GetAttributeNode	صفتی از یک گره با نام مشخص را بر می‌گرداند.
GetElementsByTagName	تمام عناصری که نام آنها برابر آن چیزی است که در پرانتز آمده از سند xml انتخاب می‌شوند.

کلاس‌های XmlText و XmlComment به ترتیب متن داخل یک عنصر و توضیحات مربوط به آن را نمایش می‌دهند. استفاده از آنها بسیار آسان است و در درس‌های بعدی در مورد آنها توضیح نمی‌دهیم.

نوشتن در یک فایل XML

با استفاده از متدها و کلاس‌های XML Document Object Model می‌توان به راحتی یک سند XML ایجاد و آن را در یک فایل ذخیره کرد. در این درس می‌خواهیم نحوه نوشتن در یک فایل XML را با استفاده از یک برنامه به شما آموزش دهیم. این برنامه به کاربر اجازه ثبت سن، جنس و نامش و ذخیره آنها در یک فایل XML را می‌دهد. یک برنامه ویندوزی جدید به شکل زیر ایجاد کنید:



نام کنترل‌های textBox را به textBoxName، textBoxAge و textBoxGender و نام دکمه را به buttonAdd تغییر دهید. حال نوبت به بخش کدنویسی می‌رسد. با زدن دکمه F7 به محیط کدنویسی رفته و فضای نام System.Xml را به قسمت فضاهای نامی اضافه کنید.

```
using System.Xml;
```

حال یک فیلد XmlDocument و یک رشته برای نمایش مسیر فایل XML اضافه می‌کنیم.

```
private XmlDocument doc;
private const string PATH = @"C:\sample.xml";
```

به محیط طراحی بر می‌گردیم و بر روی دکمه دوبار کلیک می‌کنیم تا کنترل کننده رویداد مربوط به رویداد کلیک ایجاد شود:

```
private void buttonAdd_Click(object sender, EventArgs e)
{
    doc = new XmlDocument();

    if (!System.IO.File.Exists(PATH))
    {
        XmlDeclaration declaration = doc.CreateXmlDeclaration("1.0", "UTF-8", "yes");
        XmlComment comment = doc.CreateComment("This is an XML Generated File");
        XmlElement root = doc.CreateElement("Persons");
        XmlElement person = doc.CreateElement("Person");
        XmlAttribute name = doc.CreateAttribute("name");
        XmlElement age = doc.CreateElement("Age");
        XmlElement gender = doc.CreateElement("Gender");

        name.Value = textBoxName.Text;
        age.InnerText = textBoxAge.Text;
        gender.InnerText = textBoxGender.Text;

        doc.AppendChild(declaration);
        doc.AppendChild(comment);
        doc.AppendChild(root);
        root.AppendChild(person);
        person.Attributes.Append(name);
        person.AppendChild(age);
        person.AppendChild(gender);

        doc.Save(PATH);
    }
    else
    {
        doc.Load(PATH);

        XmlElement root = doc.DocumentElement;

        XmlElement person = doc.CreateElement("Person");
        XmlAttribute name = doc.CreateAttribute("name");
        XmlElement age = doc.CreateElement("Age");
        XmlElement gender = doc.CreateElement("Gender");

        name.Value = textBoxName.Text;
        age.InnerText = textBoxAge.Text;
        gender.InnerText = textBoxGender.Text;

        person.Attributes.Append(name);
        person.AppendChild(age);
        person.AppendChild(gender);

        root.AppendChild(person);

        doc.Save(PATH);
```

```
    }

    MessageBox.Show("Details have been added to the XML File.");

    textBoxName.Text = String.Empty;
    textBoxAge.Text = String.Empty;
    textBoxGender.Text = String.Empty;
}
```

وقتی که دکمه add کلیک شود، یک XmlDocument (سند XML) ایجاد می‌شود و مسیر آن هم همان مسیری است که خودمان به صورت یک ثابت تعریف کرده‌ایم (PATH = @"C:\sample.xml";). حال با استفاده از متد ()Exists تست می‌کنیم که آیا فایل ایجاد شده است یا نه؟ اگر وجود نداشت فایل جدید را ایجاد و اولین رکورد را اضافه می‌کنیم و اگر وجود داشت، رکورد جدید را به آخر آن اضافه می‌کنیم. اجازه دهید در مورد کدها وقتی که فایل برای اولین بار ایجاد می‌شود بحث کنیم:

```
//Create neccessary nodes
XmlDeclaration declaration = doc.CreateXmlDeclaration("1.0", "UTF-8", "yes");
XmlComment comment = doc.CreateComment("This is an XML Generated File");
XmlElement root = doc.CreateElement("Persons");
XmlElement person = doc.CreateElement("Person");
XmlAttribute name = doc.CreateAttribute("name");
XmlElement age = doc.CreateElement("Age");
XmlElement gender = doc.CreateElement("Gender");
```

کد بالا گره‌های لازم برای ایجاد سند XML را ایجاد می‌کند. ابتدا یک تعریف برای سند با استفاده از کلاس XmlDeclaration و متد ()CreateXmlDeclaration از کلاس XmlDocument ایجاد می‌کنیم. همانطور که در خط سوم مثال بالا می‌بینید، این متد سه پارامتر قبول می‌کند. سپس یک توضیح را با استفاده از کلاس XmlComment و متد ()CreateComment و آن را برای استفاده متد ارسال می‌کنیم. من این توضیح را برای نشان دادن چگونگی ایجاد و استفاده از توضیحات اضافه کرده‌ام. سپس با استفاده از متد ()CreateElement و کلاس XmlElement عنصر ریشه را ایجاد می‌کنیم.

از کلاس CreateElement برای ایجاد عنصر ریشه، پدر و عناصر فرزند استفاده می‌شود. این متد یک آرگومان از نوع رشته قبول می‌کند که همان نام عنصر است. سپس یک شخص جدید را نگهداری اطلاعات وارد شده توسط کاربرایجاد می‌کنیم. عنصر Person یک صفت به نام name و دو فرزند سن (Age) و جنسیت (Gender) دارد. متد ()CreateAttribute یک آرگومان از نوع رشته قبول می‌کند که همان نام صفت است. جزییات صفت در یک شئ XmlAttribute ذخیره می‌شود. از آنجاییکه Person، Age و Gender عنصر هستند. در نتیجه ما از متد ()CreateElement و کلاس XmlElement برای آنها استفاده کرده‌ایم.

```
name.Value = textBoxName.Text;
age.InnerText = textBoxAge.Text;
gender.InnerText = textBoxGender.Text;
```

حال نوبت به اضافه کردن مقادیر به گره‌هاست. این مقادیر توسط کاربر و از طریق کنترل‌های textBox اضافه می‌شوند. شئ XmlAttribute از خاصیت Value برای دسترسی و مقداردهی گره‌ها و XmlElement از InnerText یا InnerXml برای اضافه کردن مقادیر به داخل گره‌ها استفاده می‌کند.

```
doc.AppendChild(declaration);
```

```
doc.AppendChild(comment);
doc.AppendChild(root);
root.AppendChild(person);
person.Attributes.Append(name);
person.AppendChild(age);
person.AppendChild(gender);
```

می‌خواهیم قسمت‌های مختلف را برای ایجاد یک سند کامل XML کنار هم قرار دهیم. ابتدا قسمت تعاریف سند را با استفاده از متد AppendChild() که یک XmlNode قبول می‌کند به عنوان آخرین فرزند به سند اضافه می‌کنیم. سپس توضیحات را درست در پایین قسمت تعاریف اضافه می‌کنیم. عنصر ریشه را نیز در پایین قسمت توضیحات قرار می‌دهیم. به این نکته توجه کنید که ترتیب اضافه کردن گره‌ها مهم است. بعد از ایجاد عنصر ریشه گره‌های لازم را به داخل آن اضافه کنیم.

یک عنصر person به عنصر ریشه و سپس با استفاده از متد Append() یک صفت name به خاصیت Attributes آن اضافه می‌کنیم. و در نهایت عناصر age و gender را به عنصر person اضافه می‌کنیم. به این نکته توجه کنید که ترتیب ایجاد عناصر مهم نیست. به عنوان مثال می‌توان ابتدا عنصر person را ایجاد و کامل کرده و بعد آن را به عنصر ریشه اضافه کنید. اما بهتر است که ایجاد قسمت تعریف، توضیحات و عنصر ریشه به ترتیب باشد.

```
doc.Save(PATH);
```

متد Save() سند XML ایجاد شده را در مسیر مشخص شده، ذخیره می‌کند. اجازه دهید به قسمت else کد بالا نگاهی بیندازیم. این قسمت زمانی اجرا می‌شود که فایل XML از قبل وجود داشته باشد. این قسمت فایل XML را بارگذاری کرده و به ما اجازه می‌دهد که شخص جدیدی را به لیست اشخاص موجود در عنصر ریشه اضافه کنیم.

```
//Load the XML File
doc.Load(PATH);

//Get the root element
XmlElement root = doc.DocumentElement;
```

وقتی که فایل XML با استفاده از متد Load() بارگذاری شد، عنصر ریشه با استفاده از خاصیت DocumentElement برگردانده می‌شود. حال که به عنصر ریشه دسترسی داریم می‌توانیم عناصر بیشتری به آن اضافه کنیم. یک شخص جدید را با استفاده از مواردی که توسط کاربر وارد می‌شود ایجاد و آن را به آخر لیست عناصر فرزند اضافه می‌کنیم.

سپس فایل XML را بوسیله متد Save() بروزرسانی می‌کنیم. خطوط آخر برای نمایش یک پیغام موفقیت و پاک کردن فیلدهای متنی برای ورود اطلاعات جدید به کار می‌روند. برنامه را اجرا و حداقل دو فیلد (مثلاً سن و نام) را وارد کنید. فایل XML در مسیری که در متغیر PATH تعیین شده است ایجاد می‌شود. شما می‌توانید مسیر ذخیره فایل را تغییر دهید. فایل XML را باز کرده و محتویات آن را مشاهده کنید.

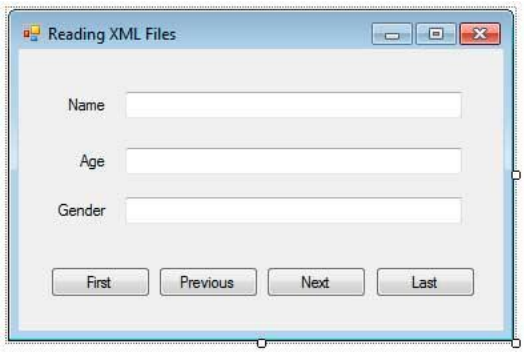
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--This is an XML Generated File-->
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
```

```
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
</Persons>
```

اکنون یک فایل XML را با استفاده از متدها و کلاس‌های XML DOM با موفقیت تولید کرده‌اید. شما می‌توانید با متدهای بیشتری کار و عملکرد آنها را تجربه کنید.

خواندن از فایل XML

خواندن یک فایل XML تکنیکی لازم برای بدست آوردن اطلاعات از آن است. به عنوان مثال می‌توانید اطلاعات مربوط به پیکربندی برنامه را در یک فایل XML ذخیره کرده تا در اجراهای بعدی، برنامه با همان تنظیمات بارگذاری شود. در این درس قصد داریم به شما نحوه خواندن یک فایل XML را آموزش دهیم. یک برنامه ویندوزی مانند شکل زیر ایجاد کنید:



نام چهار دکمه را به buttonFirst، buttonPrevious، buttonNext و buttonLast و نام جعبه‌های متن را به textBoxName، textBoxAge و textBoxGender تغییر دهید. در این آموزش نیاز به فایل XML داریم که دارای رکوردهای باشد که برنامه ما آنها را بخواند. از مسیر File > New > File > XML File یک فایل XML ایجاد می‌کنیم. نام فایل را sample.xml می‌گذاریم. محتویات آن را پاک و کدهای زیر را به جای آنها می‌نویسیم:

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folley">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
  <Person name="Jerry Frost">
    <Age>27</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Adam Wong">
```

```
    <Age>35</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

این فایل شامل پنج گره Person است. برنامه خواندن فایل را از اولین Person شروع می‌کند. کاربر هم با زدن دکمه‌های پیمایش، می‌تواند بقیه رکوردها را مشاهده کند. به محیط کدنویسی رفته و فضای نام زیر را وارد می‌کنیم:

```
using System.Xml;
```

فیلدهای زیر را به کلاس Form1 اضافه می‌کنیم:

```
private XmlDocument doc;
private XmlElement root;
private XmlElement currentPerson;
private const string PATH = @"..\..\sample.xml";
private int current = 0;
private int max;
```

doc برای بارگذاری و دستکاری فایل XML، متغیر root برای به دست آوردن عنصر ریشه و متغیر currentPerson برای نمایش عنصر Person جاری استفاده می‌شود. از مسیر "..\..\sample.xml" به این دلیل استفاده کرده‌ایم چون فایل XML در داخل دو پوشه تو در تو قرار دارد (به ازای هر پوشه دو نقطه (..) قرار داده‌ایم). متغیر current که با صفر مقداردهی شده است، برای نشان دادن اولین رکورد به کار می‌رود. متغیر max هم برای ذخیره بالاترین اندیس به کار می‌رود که برای برنامه مفید خواهد بود. حال به یک متد نیاز داریم که جزییات رکورد انتخاب شده را در جعبه‌های متن نمایش دهد. این متد را در داخل کلاس Form1 تعریف می‌کنیم:

```
private void ShowDetails(XmlElement currentPerson)
{
    textBoxName.Text = currentPerson.Attributes["name"].Value;
    textBoxAge.Text = currentPerson.GetElementsByTagName("Age")[0].InnerText;
    textBoxGender.Text = currentPerson.GetElementsByTagName("Gender")[0].InnerText;
}
```

متد ایجاد شده اطلاعات فرد انتخاب شده را می‌گیرد و مقدار صفت "name" را در جعبه متن مربوطه (textBoxName) نمایش می‌دهد. همانطور که مشاهده می‌کنید ما از خاصیت Attribute مربوط به کلاس XmlElement و یک رشته که نماینده نام صفت است استفاده کرده‌ایم و سپس با استفاده از خاصیت Value مقدار آن را به جعبه متن انتقال داده‌ایم.

مقدار عنصر Age را به وسیله متد GetElementByTagName() به دست می‌آوریم. ما نام عنصر را ارسال و متد همه عناصر فرزندی که با نام ارسال شده مطابقت دارند را به عنوان یک XmlNodeList (مجموعه‌ای از گره‌ها) بر می‌گرداند. از آنجاییکه XmlNodeList یک مجموعه است و ما فقط انتظار یک نتیجه را داریم بنابراین برای دسترسی به اولین عنصر می‌توانیم از اندیس صفر استفاده کنیم.

```
textBoxAge.Text = currentPerson.GetElementsByTagName("Age")[0].InnerText;
```

خط بالا برای افراد مبتدی کمی گیج کننده است. معنای این خط این است که اولین عنصر از مجموعه را بوسیله متد GetElementByTagName() بگیر و با استفاده از خاصیت InnerText متن آن را به دست بیاور. کد بالا را به صورت چند خط زیر هم می‌توان نوشت:

```
XmlNodeList results = currentPerson.GetElementsByTagName("Age");
XmlNode first = results[0];
textBoxAge.Text = first.InnerText;
```

خط بعدی کد هم شبیه به کد بالا می‌باشد با این تفاوت که عنصر Gender (جنسیت) به وسیله متد GetElementsByTagName() بر گشت داده می‌شود. با تکمیل متد، نوبت به ایجاد کنترل کننده رویداد می‌رسد. به محیط طراحی می‌رویم و بر روی فضای خالی از فرم کلیک می‌کنیم تا کنترل کننده رویداد Load ایجاد شود، سپس کد زیر را به آن اضافه می‌کنیم:

```
1   private void Form1_Load(object sender, EventArgs e)
2   {
3       doc = new XmlDocument();
4       doc.Load(PATH);
5
6       root = doc.DocumentElement;
7
8       max = root.GetElementsByTagName("Person").Count - 1;
9
10      currentPerson = (XmlElement)root.ChildNodes[current];
11
12      ShowDetails(currentPerson);
13  }
```

این کد بعد از بارگذاری فرم اجرا می‌شود و اولین رکورد را بازیابی و نمایش می‌دهد. ابتدا یک شئ XmlDocument ایجاد می‌کنیم (خط ۳). سپس فایل XML را که در مسیری که در متغیر PATH مشخص شده است را با استفاده از متد Load() کلاس XmlDocument بارگذاری می‌کنیم (خط ۴). عنصر ریشه را به وسیله خاصیت DocumentElement و سپس بالاترین اندیس ممکن را با استفاده شمارش عناصر فرزند به دست می‌آوریم (خط ۸). همانطور که مشاهده می‌کنید ما تعداد عناصر فرزند را منهای عدد یک کرده‌ایم چون اندیس‌ها از صفر شروع می‌شوند. برای به دست آوردن اولین شخص، مقدار صفر را به عنوان اندیس به خاصیت ChildNodes اختصاص می‌دهیم.

خاصیت ChildNodes مجموعه‌ای همه گره‌های فرزند یک گره است و با ارسال عدد صفر به عنوان اندیس، می‌توانیم اولین رکورد را بدست بیاوریم. از آنجاییکه ChildNodes عبارت است از اشیاء XmlNode، در نتیجه لازم است که گره را تبدیل به XmlElement کرده و آن را در شئ XmlElement ذخیره کنیم. البته این کار را می‌توان با استفاده از متد GetElementsByTagName() نیز انجام داد و دلیل استفاده از خاصیت ChildNodes نشان دادن روشی جایگزین است. حال عنصر به دست آمده را به متدی که از قبل ساخته‌ایم ارسال می‌کنیم. با ارسال عنصر به متد، جزییات عنصر person جاری بدست می‌آید. به محیط طراحی برگشته و بر روی buttonFirst دوبار کلیک می‌کنیم و کد زیر را در کنترل کننده رویداد آن می‌نویسیم:

```
private void buttonFirst_Click(object sender, EventArgs e)
{
    current = 0;
    currentPerson = (XmlElement)root.ChildNodes[current];
    ShowDetails(currentPerson);
}
```

خط اول اندیس جاری را صفر می‌کند و درنتیجه گره فرزند با اندیس صفر به دست می‌آید. سپس با استفاده از متد ShowDetails() جزییات آن را به دست می‌آوریم. حال اجازه دهید که به کنترل کننده رویداد کلیک دکمه buttonPrevious نگاهی بیندازیم:

```
private void buttonPrevious_Click(object sender, EventArgs e)
{
    current = (current - 1 < 0) ? 0 : current - 1;
    currentPerson = (XmlElement)root.ChildNodes[current];
    ShowDetails(currentPerson);
}
```

کد بالا به جزء در خط اول شبیه به کنترل کننده رویداد دکمه buttonFirst می‌باشد. برای به دست آوردن عنصر قبلی به سادگی از عبارت (current – 1) استفاده می‌کنیم. در خط اول چک می‌کنیم که آیا عنصر قبلی کوچک‌تر از صفر است یا نه؟ اگر کوچک‌تر بود، ما مقدار صفر را در نظر می‌گیریم، چون مقدار صفر اولین عنصر را نشان می‌دهد (عنصر اول دارای اندیس صفر است و اندیس منفی معنا ندارد). این کار از وقوع استثناء (IndexOutOfRangeException) جلوگیری می‌کند. اگر کاربر بر روی اولین رکورد باشد، کلیک بر روی دکمه buttonPrevious بی تأثیر خواهد بود. اگر اندیس بزرگ‌تر یا مساوی صفر بود عبارت current – 1 اولین عنصر یا عنصر قبلی را نشان خواهد داد و در نهایت عنصر با اندیس جدید برگشت و جزییات آن با استفاده از متد ()ShowDetails نمایش داده می‌شود. حال اجازه دهید به کدهای دکمه buttonNext که مسئول نمایش رکورد بعدی است نگاهی بیندازیم:

```
private void buttonNext_Click(object sender, EventArgs e)
{
    current = (current + 1 > max) ? max : current + 1;
    currentPerson = (XmlElement)root.ChildNodes[current];
    ShowDetails(currentPerson);
}
```

تنها تفاوت در به دست آوردن عنصر قبل و عنصر بعد در بدست آوردن اندیس عنصر جدید است. رکورد بعدی با استفاده از عبارت (current + 1) به دست می‌آید. همانطور که در کد بالا مشاهده می‌کنید ما با استفاده از یک دستور شرطی چک کرده‌ایم که آیا مقدار این عبارت از نهایت مقدار اندیسی که در متغیر max ذخیره شده است، بیشتر است یا نه؟ اگر بیشتر بود به راحتری مقدار max را برابر این عبارت قرار می‌دهیم و اگر مساوی یا کوچک‌تر از متغیر max بود در اینصورت عنصر بعدی با همین عبارت (current + 1) نمایش داده می‌شود. و در آخر بر روی دکمه buttonLast دو بار کلیک کرده و کنترل کننده رویداد زیر را به آن اضافه کنید:

```
private void buttonLast_Click(object sender, EventArgs e)
{
    current = max;
    currentPerson = (XmlElement)root.ChildNodes[current];
    ShowDetails(currentPerson);
}
```

از آنجاییکه به دست آوردن آخرین رکورد مد نظر ماست، از مقدار max (که همان تعداد عناصر است) به جای اندیس استفاده می‌کنیم. برنامه را اجرا کنید تا داده‌های اولین رکورد برای شما نمایش داده شود و سپس با کلیک بر روی دکمه‌ها در بین رکوردهای فایل XML حرکت کنید. شما اکنون یک برنامه ایجاد کرده‌اید که رکوردهای موجود در یک فایل XML را با استفاده از کلاس‌های XML DOM می‌خواند.

استفاده از XPath برای انتخاب گره‌ها

XPath یک زبان پرس و جوی ویژه برای انتخاب گره‌ها در یک سند XML می‌باشد. با این زبان، لازم نیست که تمام ساختار درختی یک سند XML را جستجو کنید. در این درس ساختار پایه ای این زبان را فرا گرفته و آن را در برنامه به کار می‌بندید. دو متدی که در این زبان برای انتخاب گره‌ها مورد استفاده قرار می‌گیرند عبارت‌اند از XmlNode.SelectNodes() و XmlNode.SelectSingleNode(). متد SelectNodes() یک XmlNodeList را برمی گرداند که شامل همه گره‌هایی است که با رشته XPath مطابقت دارند. به سند XML زیر توجه کنید:

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folley">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
  <Person name="Jerry Frost">
    <Age>27</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Adam Wong">
    <Age>35</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

فرض کنید که می‌خواهید سن هر شخص را به دست آورید، برای اینکار می‌توانید از کد زیر استفاده نمایید:

```
XmlDocument document = new XmlDocument();
document.Load("Persons.xml");

XmlNodeList nodes = document.DocumentElement.SelectNodes("/Persons/Person/Age");

foreach(XmlNode node in nodes)
{
    textBoxResult.Text += node.InnerText + "rn";
}
```

در مثال بالا متد SelectNodes() یک آرگومان رشته‌ای قبول می‌کند که همان عبارت XPath می‌باشد. مثلاً عبارت XPth در مثال بالا /Persons/Person/Age به ما می‌گوید که عنصر Age فرزند عنصر Person و عنصر Person خود نیز فرزند عنصر Persons می‌باشد. در نهایت همه گره‌های منطبق به صورت XmlList بر گشت داده می‌شوند. سپس از یک حلقه foreach برای چاپ هر سن در کنترل textBox استفاده کرده‌ایم. در جدول زیر برخی از عملیاتی که می‌توان از آنها برای پرس و جو در گره‌ها استفاده کرد آمده است:

عبارت XPath	توضیح
.	گره جاری را انتخاب می‌کند.

..	گره پدر، گره جاری را انتخاب می‌کند.
*	همه گره‌های فرزند گره جاری را انتخاب می‌کند.
nodename	همه گره‌های فرزند با یک نام مشخص را انتخاب می‌کند.
/	گره ریشه را انتخاب می‌کند.
//	گره‌هایی از گره جاری را بدون در نظر گرفتن مکان آنها انتخاب می‌کند.
//*	همه عناصر داخل یک سند را انتخاب می‌کند.
/element	یک عنصر ریشه با نام element انتخاب می‌کند. شروع مسیر با / به معنی استفاده از مسیر مطلق است.
/element/*	همه فرزندهای عنصر ریشه را انتخاب می‌کند.
element/*	همه عناصر فرزند یک عنصر فرزند را انتخاب می‌کند.
element/child	عناصر فرزندی که فرزند یک عنصر فرزند خاص از گره جاری هستند را انتخاب می‌کند.
//element	همه عناصر با یک نام خاص بدون در نظر گرفتن مکان آنها را انتخاب می‌کند.
element//child	همه عناصر فرزند یک عنصر پدر را بدون در نظر گرفتن مکان آنها در داخل گره پدر انتخاب می‌کند.
@attribute	یک صفت از گره جاری را انتخاب می‌کند.
//@attribute	همه صفات با یک نام مشخص را بدون در نظر گرفتن مکان آنها در سند، انتخاب می‌کند.
@*	همه صفات گره جاری را انتخاب می‌کند.
element[i]	یک عنصر با یک نام و اندیش مشخص را انتخاب می‌کند.
text()	متن همه گره‌های فرزند گره جاری را انتخاب می‌کند.
//text()	متن همه عناصر داخل سند را انتخاب می‌کند.
//element/text()	متن همه عناصر مورد نظر را انتخاب می‌کند.
//element[name='value']	همه عناصری که دارای فرزندی با یک مقدار خاص هستند انتخاب می‌کند.
//element[@att='value']	همه عناصری با یک صفت مشخص با مقدار مشخصی هستند را انتخاب می‌کند.

به عنوان مثال برای انتخاب گره جاری از عملگر (.) استفاده می‌شود:

```
XmlNode current = document.DocumentElement.SelectSingleNode(".");
```

به این نکته توجه کنید که متد XmlNode.SelectSingleNode() فقط یک گره را انتخاب می‌کند و در صورتیکه چندین نتیجه وجود داشته باشد، فقط اولین گره منطبق را بر می‌گرداند. برای انتخاب همه عناصر Person که فرزند گره Persons هستند، می‌توانید از عبارت Xpath زیر استفاده کنید:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("/Persons/Person");
```

به این نکته توجه نمایید که شروع عبارت با علامت اسلش (/) به معنای این است که می‌خواهید از مسیر مطلق استفاده کنید. با گره ریشه (Persons) شروع کرده‌ایم و سپس به دنبال گره‌های Person که مستقیماً گره فرزند آن هستند، می‌گردیم. برای به دست آوردن سن هر فرد از عبارت زیر استفاده می‌کنیم:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("/Persons/Person/Age");
```

می‌توان از آدرس دهی نسبی هم به شرطی که جستجو از گره جاری شروع شود، استفاده کرد. به عنوان مثال می‌توان همه گره‌های فرزند گره ریشه DocumentElement را با استفاده از کد زیر به دست آورد:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("Person");
```

یا سن هر شخص را به وسیله کد زیر:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("Person/Age");
```

همانطور که مشاهده می‌کنید در آدرس دهی نسبی از علامت / استفاده نمی‌کنیم. می‌توانید پرس و جوی گره‌ها را بدون در نظر گرفتن مکان آنها در سند انجام داد. این کار زمانی مفید است که نخواهید همه سند را برای یافتن گره‌هایی که با گره مورد نظر شما انطباق دارند، جستجو کنید. به عنوان مثال اگر بخواهید همه گره‌های جنسیت (Gender) را بدست آورید، می‌توانید از کد زیر استفاده نمایید:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("//Gender");
```

قرار گرفتن علامت / قبل از نام عنصر بدین معناست که باید همه سند مورد جستجو قرار گیرد. عبارت پرس و جو تمام گره‌های منطبق با عبارت مورد نظر را در هر جای سند بر می‌گرداند. اگر بخواهید منطقه محدودی از سند را جستجو کنید، می‌توانید گره پدر یا ریشه گره‌ای که قرار است جستجو در آن انجام شود را، مشخص کنید:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("/Persons//Gender");
```

کد بالا همه عناصر Gender که در داخل گره Person قرار دارند را جستجو می‌کند. برای یافتن یک گره خاص می‌توانید از اندیس آن استفاده کنید. کد زیر سومین فرزند عنصر Person را برگشت می‌دهد:

```
XmlNode personNodes = document.DocumentElement.SelectSingleNode("/Persons/Person[3]");
```

در کد بالا از آنجاییکه به دنبال یک گره ساده هستیم از متد SelectSingleNode() استفاده کرده‌ایم. سومین عنصر Person به وسیله Person[3] نمایش داده می‌شود به همین منظور ما از اندیس ۳ استفاده کرده‌ایم. اندیس در XML بر خلاف آرایه‌ها از ۱ شروع می‌شود نه صفر.

برای انتخاب همه افرادی که جنس مذکر هستند، می‌توانید از کد زیر استفاده کنید:

```
XmlNodeList personNodes = document.DocumentElement.SelectNodes("//Person[Gender='Male']");
```

از آنجاییکه از علامت // استفاده کرده‌ایم تمام سند مورد جستجو قرار می‌گیرد. در داخل کروشه هم نام عنصر فرزند و مقدار مخصوص آن را نوشته‌ایم. مقدار عنصر اگر از نوع رشته باشد باید در داخل کوتیشن قرار داده شود. هنگام کار با صفات یک عنصر می‌توانید از علامت @ قبل از نام صفت استفاده کنید. به عنوان مثال کد زیر اسامی همه افراد را چاپ می‌کند:

```
XmlNodeList list = document.DocumentElement.SelectNodes(@"//Person/@name");

foreach (XmlNode node in list)
{
    textBox1.Text += node.Value + "rn";
}
```

XPath مبحث بزرگی است. فقط برخی از اجزای پای‌های آن در این درس را بررسی کردیم. برای یادگیری مطالب بیشتر به آدرس زیر مراجعه کنید :

```
http://www.w3schools.com/xpath/default.asp
```

استفاده از فونت در سی‌شارپ

نحوه تغییر و دستکاری فونت یک کنترل موجود در یک برنامه ویندوزی به چه صورتی است؟ کلاس System.Drawing.Font به شما اجازه می‌دهد تا فونت‌های دلخواه خود را ساخته و به خاصیت Font کنترل‌های مختلف نسبت دهید. در زیر لیستی از خاصیت‌های مفید این کلاس را مشاهده می‌کنید.

خاصیت	توضیح
Bold	مشخص می‌کند که آیا فونت bold است یا نه؟
FontFamily	خانواده فونتی که فونت از آن استفاده می‌کند را مشخص می‌کند.
Height	فاصله بین خطوط فونت را تعیین می‌کند.
Italic	تعیین می‌کند که آیا فونت مورب (italic) است یا نه؟
Name	نام فونت را تعیین می‌کند.
Size	سایز فونت را بر اساس واحد انتخاب شده تعیین می‌کند.
Strikeout	تعیین می‌کند که آیا فونت دارای یک خط افقی در داخل خود باشد یا نه؟
Style	قالب فونت را مشخص می‌کند.

Underline	تعیین می‌کند که آیا فونت، زیر خط داشته باشد یا نه؟
Unit	واحد اندازه فونت را مشخص می‌کند.

ایجاد شیء فونت

در کد زیر سازنده کلاس فونت فراخوانی و از فونت Times New Roman با اندازه ۱۲ استفاده شده است.

```
Font myFont = new Font("Times New Roman", 12);
```

font family نوع فونت را مشخص می‌کند. برخی از فونت‌های معمول و مشهور عبارت‌اند از Arial، Verdana، Courier New، Times New Roman و Sans Serif. به این نکته توجه کنید که قبل از استفاده از یک فونت باید از وجود آن در سیستمتان مطمئن شوید. فونت‌های ذکر شده، در حالت پیش‌فرض در ویندوز وجود دارند. از کلاس System.Drawing.FontFamily برای ایجاد یک شئ FontFamily که از یک فونت با نام مشخص استفاده می‌کند، استفاده می‌شود. می‌توانید از یکی از سربارگذاری‌های کلاس Font استفاده کنید و یک شئ FontFamily را به آن ارسال نمایید:

```
FontFamily family = new FontFamily("Arial");
Font myFont = new Font(family, 12);
```

استفاده از شیء FontFamily به شما اجازه می‌دهد که از برخی از متدهای آن مانند GetCellAscent()، GetCellDescent()، GetEmHeight() و GetLineSpacing() استفاده کنید. سربارگذاری دیگر به شما اجازه می‌دهد که قالب فونت را تعیین کنید. برای این کار از مقادیر نوع شمارشی System.Drawing.FontStyle استفاده می‌کنیم. در مثال زیر یک فونت ضخیم (bold) ایجاد کرده‌ایم.

```
Font myFont = new Font("Verdana", 14, FontStyle.Bold);
```

یکی دیگر از سازنده‌های مفید کلاس Font به شما اجازه می‌دهد که از خواص یک فونت استفاده کرده و قالب جدیدی به آن اعمال نمایید. فرض کنید که یک برچسب (label) دارید و می‌خواهید آن را به صورت italic (حروف کج) در آورید. برای این کار می‌توان از مقدار FontStyle.Italic به صورت زیر استفاده کرد:

```
Font myFont = new Font(label1.Font, FontStyle.Italic);
```

همچنین می‌توان واحد اندازه گیری سایز فونت را مشخص کرد. به عنوان مثال اندازه ۱۲ می‌تواند به صورت اینچ، پیکسل یا میلی متر باشد. برای تعیین واحد اندازه گیری می‌توان از نوع شمارشی System.Drawing.GraphicsUnit استفاده کرد. کد زیر یک فونت با اندازه ۱۲ پیکسل ایجاد می‌کند:

```
Font myFont = new Font("Times New Roman", 12, GraphicsUnit.Pixel);
```

خانواده فونت (font family)

بسته به اینکه چند نوع فونت در سیستم شما نصب است می‌توان از فونت‌های مختلفی استفاده کرد. برای مشاهده فونت‌های نصب شده در ویندوز Y به مسیر Start > Control Panel > Appearance and Personalization > Fonts بروید. شکل زیر چندین خانواده فونت را نمایش می‌دهد.

شما می‌توانید نام یک خانواده فونت را به عنوان رشته ارسال و یا یک شئ FontFamily را هنگام تعریف یک شئ فونت ایجاد کنید.

```
Font myFont = new Font("Consolas", 14);
//or
Font myFont = new Font(new FontFamily("Consolas"), 14);
```

اندازه فونت

هنگامی که با سایز فونت سرکار دارید باید به دو چیز توجه کنید: اندازه واقعی و واحد مورد استفاده. می‌توانید از خاصیت Size اندازه فونت را مشخص کنید:

```
myFont.Size = 12;
```

با استفاده از مقادیر شمارشی System.Drawing.GraphicsUnit که برخی از آنها در جدول زیر آمده است هم می‌توان واحد اندازه گیری اندازه فونت را مشخص کرد:

مقادیر	توضیح
Pixel	مشخص می‌کند که واحد اندازه گیری پیکسل است.
Point	مشخص می‌کند که واحد اندازه گیری نقطه چاپگر است.
Inch	مشخص می‌کند که واحد اندازه گیری اینچ است.
Millimeter	مشخص می‌کند که واحد اندازه گیری میلی متر است.

از خاصیت Unit کلاس فونت هم می‌توان برای تعیین واحد اندازه فونت استفاده کرد:

```
myFont.Unit = GraphicsUnit.Pixel;
```

قالب فونت

می‌توان چندین قالب مختلف را به فونت اعمال کرد. کلاس Font دارای چندین خاصیت است که مقدار بولی قبول می‌کنند. به عنوان مثال می‌توان از خواص Bold یا italic استفاده کرد و مقدار true را به آنها اختصاص داد تا فونت به صورت ضخیم و یا حروف کج در آید.

```
myFont.Bold = true;
myFont.Italicized = true;
```

کد بالا فونت را همزمان به صورت ضخیم و حروف کج نمایش می‌دهد. از خواص Strikeout و Underline هم می‌توان به صورت بالا استفاده نمود. همچنین می‌توان از مقادیر نوع شمارشی System.Drawing.FontStyles برای قالب دهی به فونت استفاده کرد:

مقدار	توضیح
Regular	بدون قالب
Bold	متن را صورت ضخیم در می‌آورد.
Italic	متن را به صورت حروف کج نمایش می‌دهد.
Underline	یک خط به زیر متن می‌کشد.
Strikeout	یک خط را بر روی متن می‌کشد.

شکل زیر نتیجه اعمال هر یک از مقادیر بالا بر روی متن را نشان می‌دهد:

با استفاده از سازنده کلاس Font هم می‌توان یک قالب به فونت اعمال کرد:

```
myFont = new Font(myFont, FontStyle.Bold);
```

از عملگر بیتی OR هم برای اعمال چندین قالب به فونت می‌توان استفاده کرد:

```
myFont = new Font(myFont, FontStyle.Bold | FontStyle.Italic);
```

کد بالا فونت را هم به صورت ضخیم و هم حروف کج در می‌آورد. اگر بخواهید یک قالب خاص را از فونت حذف کنید می‌توانید از عملگر بیتی XOR استفاده کنید:

```
myFont = new Font(myFont, myFont.Style ^ FontStyle.Bold);
```

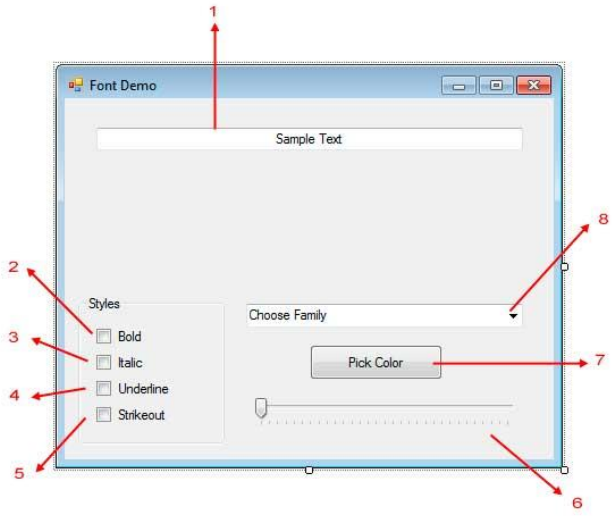
عملگر XOR به صورت دوگانه عمل می‌کند، بدین معنی که اگر فونت دارای قالب خاصی باشد آنرا حذف و اگر قالب به فونت اعمال نشده باشد آنرا به فونت اختصاص می‌دهد. عملگر بیتی AND نیز چک می‌کند که آیا قالب مشخصی به فونت اعمال شده است یا نه؟.

```
if ( (myFont.Style & FontStyle.Bold) == FontStyle.Bold)
   MessageBox.Show("The font is bold.");
```

به این نکته توجه کنید که عبارت در داخل پرانتز اعمال شده است چون عملگر == دارای حق تقدم بالاتری می‌باشد.

ویرایش فونت‌ها (مثال)

در این درس می‌خواهیم در قالب یک برنامه روش دستکاری خاصیت فونت کنترل‌ها را با استفاده از کنترل‌های دیگر، به شما آموزش دهیم. یک برنامه ویندوزی با نام FontDemo ایجاد کنید. نمای کلی برنامه را به صورت زیر در آورید:



برچسب	نوع	نام	خاصیت	مقدار
1	TextBox	fontTextBox	Text	Sample Text
			TextAlignment	Center
2	CheckBox	boldCheckBox	Text	Bold
3	CheckBox	italicCheckBox	Text	Italic
4	CheckBox	underlineCheckBox	Text	Underline
5	CheckBox	strikeoutCheckBox	Text	Strikeout

6	TrackBar	sizeTrackBar	Minimum	10
			Maximum	40
7	Button	colorButton	Text	Pick Color
8	ComboBox	familyComboBox	Text	Choose Family

کاربر با استفاده از چک باکس‌ها (۲ تا ۵) می‌تواند قالب‌های مختلفی را انتخاب کند. وقتی که یکی از این چک باکس‌ها تیک می‌خورد، قالب مرتبط با آن به متن موجود در جعبه متن اعمال و وقتی تیک برداشته شود قالب اعمال شده حذف می‌شود. در برنامه از یک کنترل کمبوباکس هم استفاده شده که فونت‌های موجود در سیستم را در داخل آن می‌ریزیم. با استفاده از این کنترل کاربر می‌تواند فونت مورد نظر خود را به متن اعمال کند. وقتی که بر روی دکمه (۷) کلیک شود، کادر محاوره‌ای رنگ (ColorDialog) ظاهر شده و به کاربر اجازه می‌دهد که رنگ متن را تغییر دهد. با استفاده از بند انگشتی کنترل trackbar(6) اندازه فونت را می‌توان تغییر داد.

استفاده از عملگر بیتی XOR برای اضافه و حذف نمودن قالب‌ها

حال اجازه دهید به قسمت کدنویسی برنامه برویم. با چک باکس‌ها که مسئول قالب دهی به فونت‌ها هستند شروع می‌کنیم. بر روی هر کدام از آنها دو بار کلیک کرده تا رویداد CheckedChange مربوط به هر یک ایجاد شود و سپس کدهای زیر را به آنها اضافه می‌کنیم:

```
private void boldCheckBox_CheckedChanged(object sender, EventArgs e)
{
    fontTextBox.Font = new Font(fontTextBox.Font, fontTextBox.Font.Style ^ FontStyle.Bold);
}

private void italicCheckBox_CheckedChanged(object sender, EventArgs e)
{
    fontTextBox.Font = new Font(fontTextBox.Font, fontTextBox.Font.Style ^ FontStyle.Italic);
}

private void underlineCheckBox_CheckedChanged(object sender, EventArgs e)
{
    fontTextBox.Font = new Font(fontTextBox.Font, fontTextBox.Font.Style ^ FontStyle.Underline);
}

private void strikeThroughCheckBox_CheckedChanged(object sender, EventArgs e)
{
    fontTextBox.Font = new Font(fontTextBox.Font, fontTextBox.Font.Style ^ FontStyle.Strikeout);
}
```

برای هر کنترل کننده رویداد با استفاده از سازنده‌ای که یک فونت و یک قالب جدید قبول می‌کند یک فونت جدید ایجاد کرده‌ایم. در اولین آرگومان هر سازنده فونت را به fontTextBox برای نگهداری تنظیماتی از قبیل اندازه و خانواده فونت ارسال می‌کنیم. در دومین آرگومان هر سازنده از عملگر بیتی XOR در بین دو قالب (قالب فعلی و قالب تغییر یافته) استفاده می‌کنیم. این عملگر مکانیزمی دو گانه دارد. فراخوانی این عملگر باعث می‌شود که اگر تغییری بر روی قالب اعمال نشده باشد آن را اعمال کند و اگر اعمال شده باشد، آن را حذف کند.

تغییر اندازه فونت

با استفاده از کنترل TrackBar به کاربر اجازه داده می‌شود که اندازه فونت fontTextBox را تغییر دهد. این کنترل دارای خواصMinimum و Maximum می‌باشد که با اعداد ۱۰ و ۴۰ مقداردهی شده‌اند و در نتیجه کاربر فقط قادر خواهد بود اندازه‌ای بین ۱۰ تا ۴۰ را برای فونت انتخاب کند. بر روی کنترل مذکور دوبار کلیک کرده تا کنترل کننده رویداد برای رویداد Scroll آن ایجاد شود. کدهای زیر را در داخل کنترل کننده رویداد بنویسید:

```
private void sizeTrackBar_Scroll(object sender, EventArgs e)
{
    fontTextBox.Font = new Font(fontTextBox.Font.FontFamily, sizeTrackBar.Value,
        fontTextBox.Font.Style, fontTextBox.Font.Unit);
}
```

از سازنده کلاس فونت که چهار آرگومان قبول می‌کند، استفاده کرده‌ایم. به سادگی خانواده فونت، قالب و واحد فونت مورد نظرمان را از طریق fontTextBox به سازنده ارسال می‌کنیم. همانطور که در کد بالا مشاهده می‌کنید، دومین آرگومان تعیین کننده اندازه فونت است که از طریق مقدار خاصیت Value مربوط به کنترل TrackBar مشخص می‌شود.

بارگذاری فونت‌های نصب شده در سیستم

از کنترل combo box برای انتخاب یکی از فونت‌های موجود در سیستم استفاده می‌شود. هم می‌توان با استفاده از یک نوع شمارشی لیستی از فونت‌ها را برای این کنترل مشخص کرد و هم می‌توان از فونت‌های موجود در سیستم استفاده نمود. که ما دومی را انتخاب می‌کنیم و لیست فونت‌های موجود در سیستم را هنگام بارگذاری فرم در درون کمبوباکس می‌ریزیم. بر روی قسمتی خالی از فرم دوبار کلیک کنید، تا کنترل کننده رویداد Load آن ایجاد شود:

```
private void Form1_Load(object sender, EventArgs e)
{
    InstalledFontCollection fonts = new InstalledFontCollection();

    foreach (FontFamily font in fonts.Families)
    {
        familyComboBox.Items.Add(font.Name);
    }
}
```

کلاس InstalledFontCollection به شما اجازه دسترسی به فونت‌های موجود در سیستم را می‌دهد. به این نکته توجه کنید که این کلاس در فضای نامی System.Drawing.Text قرار دارد و شما این فضای نامی را باید به برنامه‌تان اضافه کنید. با استفاده از یک حلقه هم فونت‌ها را به خاصیت Families کلاس InstalledFontCollection اضافه می‌کنیم.

خاصیت Famillies شامل مجموعه‌ای از اشیاء FontFamily است که نشان دهنده فونت‌های نصب شده در سیستم می‌باشند. به محیط طراحی بر می‌گردیم و بر روی combo box دو بار کلیک می‌کنیم تا کنترل کننده رویداد SelectedIndexChange تولید شود. سپس کدهای زیر را در داخل آن می‌نویسیم:

```
private void familyComboBox_SelectedIndexChanged(object sender, EventArgs e)
```

```
{
    fontTextBox.Font = new Font(familyComboBox.SelectedItem.ToString(),
        fontTextBox.Font.Size, fontTextBox.Font.Style, fontTextBox.Font.Unit);
}
```

با استفاده از سازنده کلاس Font یک شیء جدید Font ایجاد می‌کنیم، که چهار آرگومان قبول می‌کند. مهم‌ترین آرگومان همان آرگومان اول است که خانواده فونت را مشخص می‌کند. از خاصیت SelectedItem مربوط به familyComboBox برای انتخاب آیتمی که در familyComboBox انتخاب کرده‌ایم استفاده و سپس آن را با استفاده از متد ()ToString به متن تبدیل می‌کنیم. اندازه جاری، قالب و واحد فونت را نیز با استفاده از fontTextBox به سازنده ارسال می‌کنیم.

تغییر رنگ فونت با استفاده از خاصیت ForeColor

رنگ فونت به وسیله خود فونت تعیین نمی‌شود. بلکه باید از یک خاصیت کنترل‌ها به نام ForeColor برای تغییر رنگ متن آنها استفاده کرد. بر روی دکمه colorButton دو بار کلیک کرده و کد زیر را در کنترل کننده رویداد آن بنویسید:

```
private void colorButton_Click(object sender, EventArgs e)
{
    ColorDialog colorDialog = new ColorDialog();

    if (colorDialog.ShowDialog() == DialogResult.OK)
        fontTextBox.ForeColor = colorDialog.Color;
}
```

وقتی که بر روی دکمه کلیک می‌شود یک کادر محاوره‌ای ColorDialog ظاهر می‌شود که کاربر با استفاده از متد ()ShowDialogمی‌تواند یک رنگ را انتخاب نماید. سپس تست می‌کنیم که آیا مقدار برگشتی از متد ()ShowDialog، DialogResult.OK است یا خیر؟ مقدار برگشتی DialogResult.OK بدین معناست که کاربر بر روی OK کلیک کرده است. سپس رنگ انتخاب شده توسط کاربر را به خاصیت ForeColor اعمال می‌کنیم، در نتیجه رنگ فونت به رنگ انتخاب شده تغییر می‌یابد.

مقایسه اشیاء با استفاده از رابط‌های IComparable و IComparer

دو رابط مفید برای مقایسه اشیایی که توسط کاربر تعریف شده‌اند وجود دارد. این دو رابط IComparable<T> و IComparer<T> می‌باشند. رابط IComparable<T> در یک کلاس پیاده سازی می‌شود و اجازه می‌دهد کلاس یا شی ایجاد شده از کلاس با اشیاء دیگر از همین کلاس مقایسه شوند. IComparer<T> در یک کلاس جداگانه پیاده سازی می‌شود. همانطور که از نام این رابط پیداست، پیاده سازی آن باعث ایجاد یک کلاس مقایسه پذیر می‌شود. به این نکته توجه کنید که نسخه‌های غیر جنریک این دو رابط نیز وجود دارد ولی کار کردن با نسخه‌های جنریک آنها بسیار راحت تر بوده و شما نیاز به تبدیل برای مقایسه اشیاء ندارید.

رابط IComparable<T>

حال به نحوه استفاده از رابط IComparable<T> می‌پردازیم. در مثال زیر یک کلاس نشان داده شده است که رابط IComparable<T> را پیاده سازی می‌کند:

```
using System;
using System.Collections.Generic;
using System.Collections;

namespace IComparableInterface
{
    public class Person : IComparable<Person>
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }

        public int CompareTo(Person other)
        {
            if (this.Age > other.Age)
                return 1;
            else if (this.Age < other.Age)
                return -1;
            else
                return 0;
        }
    }

    public class Program
    {
        static void Main(string[] args)
        {
            Person person1 = new Person { FirstName = "John", LastName = "Smith", Age = 21 };
            Person person2 = new Person { FirstName = "Mark", LastName = "Logan", Age = 19 };
            Person person3 = new Person { FirstName = "Luke", LastName = "Adams", Age = 20 };

            Person youngest = GetYoungest(person1, person2, person3);
            Person oldest = GetOldest(person1, person2, person3);

            Console.WriteLine("The youngest person is {0} {1}.",
                youngest.FirstName, youngest.LastName);
            Console.WriteLine("The oldest person is {0} {1}.",
                oldest.FirstName, oldest.LastName);
            Console.ReadKey();
        }

        private static Person GetYoungest(Person person1, Person person2, Person person3)
        {
            Person youngest = person1;

            if (person2.CompareTo(youngest) == -1)
                youngest = person2;

            if (person3.CompareTo(youngest) == -1)
                youngest = person3;

            return youngest;
        }

        private static Person GetOldest(Person person1, Person person2, Person person3)
        {
            Person oldest = person1;

            if (person2.CompareTo(oldest) == 1)
                oldest = person2;

            if (person3.CompareTo(oldest) == 1)
                oldest = person3;

            return oldest;
        }
    }
```

```
68  }
```

```
The youngest person is Mark Logan.
The oldest person is John Smith.
```

وقتی که یک کلاس از رابط IComparable<T> استفاده می‌کند، لازم است که تنها متد آن یعنی متد CompareTo() را نیز پیاده سازی کند. متد CompareTo() یک مقدار صحیح را بر می‌گرداند. این متد یک آرگومان قبول می‌کند که همان شیی است که قرار است با شی جاری مقایسه شود. در داخل متد CompareTo() در مثال بالا ما سن (age) شخص فعلی را با سن شخص دیگر مقایسه کرده‌ایم. طبق قرارداد اگر سن شخص مورد نظر ما از سن شخص دیگر بیشتر بود مقداری بزرگ‌تر از صفر، اگر کمتر بود مقداری کمتر از صفر و اگر مساوی بود مقدار صفر برگشت داده می‌شود. خطوط ۷-۲۲ پیاده سازی رابط IComparable<T> توسط شیء Person را نشان می‌دهد. برنامه جوان‌ترین و پیرترین شخص را تشخیص می‌دهد. در خطوط ۲۸-۳۰ سه شیء Person با مقادیر کاملاً اختیاری ایجاد شده است. در خطوط ۳۳-۳۲ متغیرهایی برای نگهداری جوان‌ترین و پیرترین شخص تعریف شده‌اند.

در خط ۳۲ متد GetYoungest() را فراخوانی کرده‌ایم. این متد در خطوط ۵۳-۴۲ تعریف شده است و سه شخص را که قرار است از لحاظ سنی با هم مقایسه شوند را قبول می‌کند. در خط ۲۱ فرض را بر این گذاشته‌ایم که اولین شخص (person1) جوان‌ترین شخص است. سپس با استفاده از پیاده سازی متد CompareTo() تست می‌کنیم که آیا شخص دوم (person2) از شخص اول جوان‌تر است یا نه. در داخل متد مذکور سن شخص دوم و اول را با هم مقایسه می‌کنیم. اگر سن شخص دوم کمتر بود، باید مقدار ۱- برگشت داده شده و در خط ۴۷، person2 به عنوان جوان‌ترین شخص معرفی شود.

در خطوط ۵۰-۴۹ از تکنیکی مشابه برای شخص سوم استفاده کرده‌ایم. بعد از مقایسه جوان‌ترین شخص در خط ۵۲ به عنوان نتیجه برگشت داده می‌شود. در خط ۳۳ متد GetOldest() که در خطوط ۶۶-۵۵ تعریف شده است فراخوانی می‌شود. کدهای داخل این متد شبیه به متد GetYoungest() است با این تفاوت که تست می‌شود که آیا سن شخص دیگر بزرگ‌تر از سن شخص مورد نظر ماست یا نه؟ بنابراین باید انتظار داشته باشیم که مقدار ۱ به جای ۱- توسط متد برگشت داده شود. در خطوط ۳۸-۳۵ نام جوان‌ترین و پیرترین شخص چاپ می‌شود.

رابط IComparer<T>

IComparer<T> در یک کلاس جداگانه پیاده سازی می‌شود. همانطور که از نام این رابط پیداست، پیاده سازی آن باعث ایجاد یک کلاس مقایسه پذیر می‌شود. به وسیله این رابط می‌توان چندین مقایسه برای کلاس Person ایجاد کرد. مثلاً در مثال کلاس Person، اشیاء ایجاد شده بر اساس سن (Age)، نام (FirstName) و یا نام خانوادگی (LastName) مورد مقایسه قرار می‌گیرند. هنگام استفاده از این رابط لازم است یک متد به نام Compare() که دو شیء قبول می‌کند و یک عدد صحیح را به عنوان نتیجه بر می‌گرداند را پیاده سازی کند. از آنجاییکه از رابط IComparer<Person> استفاده کرده‌ایم، متد Compare() به طور خودکار دو شئ Person قبول می‌کند:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Collections;
4
5   namespace IComparerInterface
6   {
```

```
7       public class Person
8       {
9           public string FirstName { get; set; }
10          public string LastName { get; set; }
11          public int Age { get; set; }
12      }
13
14      public class FirstNameComparer : IComparer<Person>
15      {
16          public int Compare(Person person1, Person person2)
17          {
18              return person1.FirstName.CompareTo(person2.FirstName);
19          }
20      }
21
22      public class LastNameComparer : IComparer<Person>
23      {
24          public int Compare(Person person1, Person person2)
25          {
26              return person1.LastName.CompareTo(person2.LastName);
27          }
28      }
29
30      public class AgeComparer : IComparer<Person>
31      {
32          public int Compare(Person person1, Person person2)
33          {
34              return person1.Age.CompareTo(person2.Age);
35          }
36      }
37
38      class Program
39      {
40          static void Main(string[] args)
41          {
42              List<Person> persons = new List<Person>
43              {
44                      new Person { FirstName = "John", LastName = "Smith", Age = 21 },
45                      new Person { FirstName = "Mark", LastName = "Logan", Age = 19 },
46                      new Person { FirstName = "Luke", LastName = "Adams", Age = 20 }
47              };
48
49              Console.WriteLine("Original Order");
50              foreach (Person p in persons)
51                  Console.WriteLine("{0} {1}, Age: {2}", p.FirstName, p.LastName, p.Age);
52
53              Console.WriteLine("\nSort persons based on their:");
54              Console.WriteLine("[1] FirstName\n[2] LastName\n[3]Age");
55
56              Console.Write("Enter your choice: ");
57              int choice = Int32.Parse(Console.ReadLine());
58
59              ReorderPersons(choice, persons);
60
61              Console.WriteLine("New Order");
62              foreach (Person p in persons)
63                  Console.WriteLine("{0} {1}, Age: {2}", p.FirstName, p.LastName, p.Age);
64          }
65
66          private static void ReorderPersons(int choice, List<Person> persons)
67          {
68              IComparer<Person> comparer;
69
70              if (choice == 1)
71                  comparer = new FirstNameComparer();
72              else if (choice == 2)
73                  comparer = new LastNameComparer();
```

```
74              else
75                  comparer = new AgeComparer();
76
77              persons.Sort(comparer);
78          }
79      }
80  }
```

```
Original Order
John Smith, Age: 21
Mark Logan, Age: 19
Luke Adams, Age: 20

Sort persons based on their:
[1] FirstName
[2] LastName
[3]Age
Enter your choice: 1
New Order
John Smith, Age: 21
Luke Adams, Age: 20
Mark Logan, Age: 19
Original Order
John Smith, Age: 21
Mark Logan, Age: 19
Luke Adams, Age: 20

Sort persons based on their:
[1] FirstName
[2] LastName
[3]Age
Enter your choice: 2
New Order
Luke Adams, Age: 20
Mark Logan, Age: 19
John Smith, Age: 21
Original Order
John Smith, Age: 21
Mark Logan, Age: 19
Luke Adams, Age: 20

Sort persons based on their:
[1] FirstName
[2] LastName
[3]Age
Enter your choice: 3
New Order
Mark Logan, Age: 19
Luke Adams, Age: 20
John Smith, Age: 21
```

در مثال بالا با استفاده از کلاس FirstNameComparer که رابط IComparer را پیاده سازی می‌کند، دو شئ Person بر اساس نام مقایسه می‌شوند (خطوط ۲۰-۱۴). در متد ()Compare به سادگی و با استفاده از متد از پیش تعریف شده ()CompareTo از کلاس String استفاده کرده‌ایم (چون خاصیت FirstName یک رشته است) و یک مقدار را به عنوان نتیجه بر می‌گردانیم.

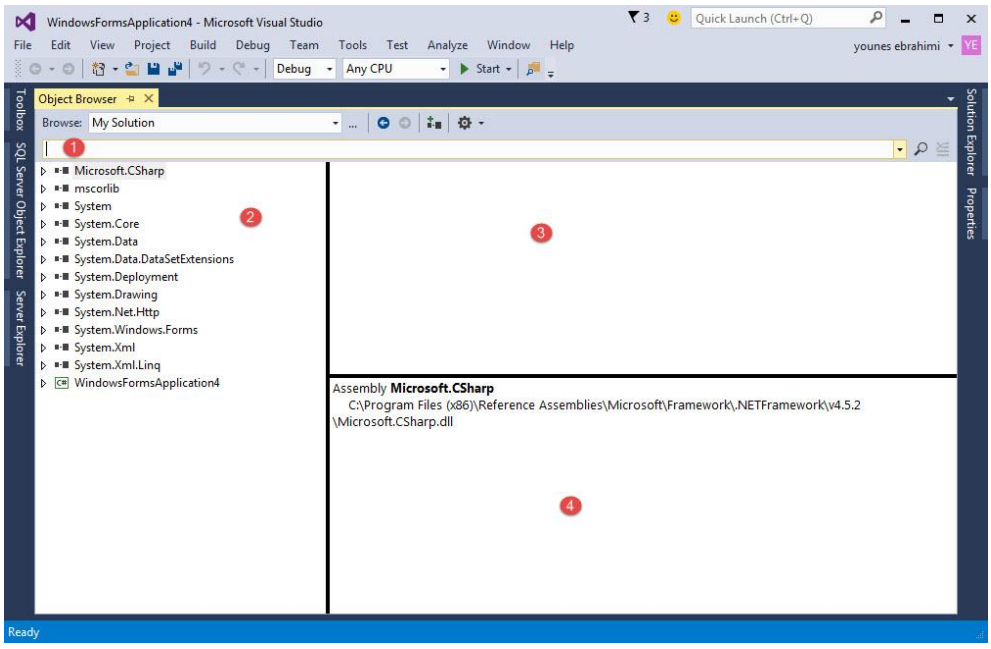
به روشی مشابه از کلاس‌های LastNameComparer و AgeComparer برای مقایسه اشیاء بر اساس نام خانوادگی و سن استفاده می‌کنیم (خطوط ۳۶-۲۲). متد ()Compare در صورتی که دو پارامتر با هم برابر باشند، مقدار ۰، اگر پارامتر اول از پارامتر دوم بزرگ‌تر باشد، مقداری بزرگ‌تر از ۰ و اگر پارامتر اول از پارامتر دوم کوچک‌تر باشد مقداری کوچک‌تر از ۰ را بر می‌گرداند. در خطوط ۱۸، ۲۶ و ۳۴ نحوه مقایسه اشیاء بر اساس خواص مختلف را نشان داده‌ایم.

در خطوط ۴۷-۴۲ اشیا با مقادیر از پیش تعریف شده ای برای هر یک از خاصیت‌هایشان ایجاد شده است. در خطوط ۵۱-۵۰ ترتیب عادی و اصلی این اشیاء نمایش داده شده است. در خطوط ۵۴-۵۳ لیستی از انتخاب‌هایی که کاربر بر اساس آنها می‌تواند عملیات مرتب سازی را انجام دهد آورده شده است.

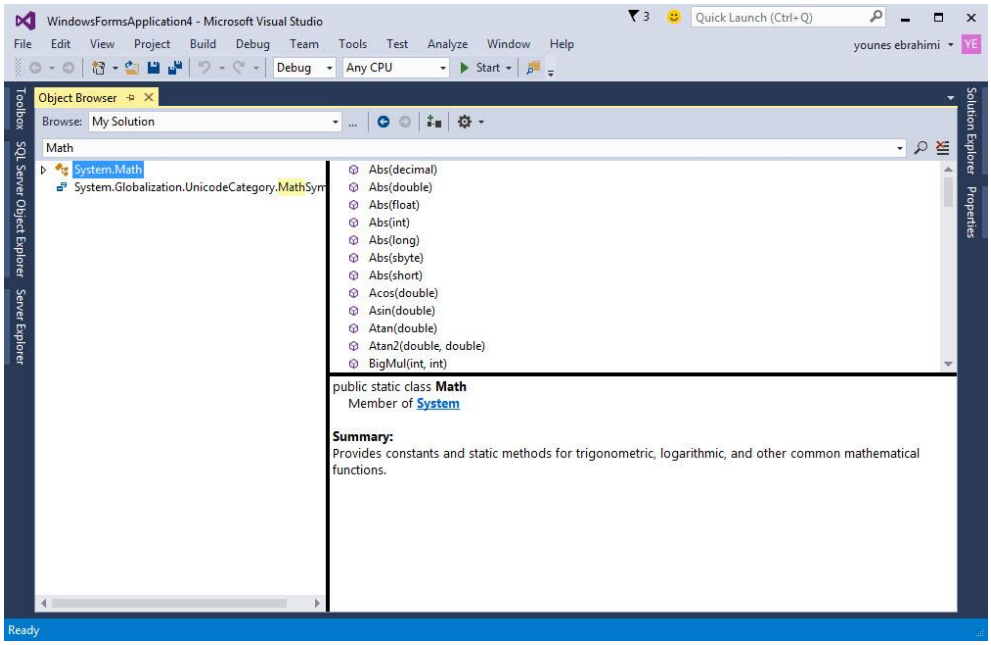
در خطوط ۵۷-۵۶ از کاربر در مورد انتخابش سؤال می‌شود. در خط ۵۹ متد `ReorderPersons()` متد از پیش تعریف شده‌ی خطوط ۷۸-۶۶ را فراخوانی می‌کنیم. این متد انتخاب کاربر و لیستی از اشیا که قرار است بر اساس خاصیتی که کاربر انتخاب کرده است مرتب شوند را قبول می‌کند. در داخل متد یک متغیر تعریف کرده‌ایم که از نوع `IComparer<Person>` است و در نتیجه می‌تواند هر نوع کلاسی که رابط مذکور را پیاده سازی می‌کند را شامل شود. در خطوط ۷۵-۷۰ چک می‌کنیم که اگر کاربر یک مقدار عددی خاص را انتخاب کرد، چه کارهایی انجام شود. در خط ۷۷ از متد `Sort()` کلاس `List<T>` استفاده کرده‌ایم. این متد دارای یک نسخه سربارگذاری شده است که یک شیء `IComparer<T>` را قبول می‌کند. ما در خط ۶۸ کلاس مقایسه کننده بر اساس نوع انتخاب کاربر را به این متد می‌دهیم و سپس متد `Sort()` شی `Person` را بر اساس این کلاس مرتب می‌کند.

Object Browser

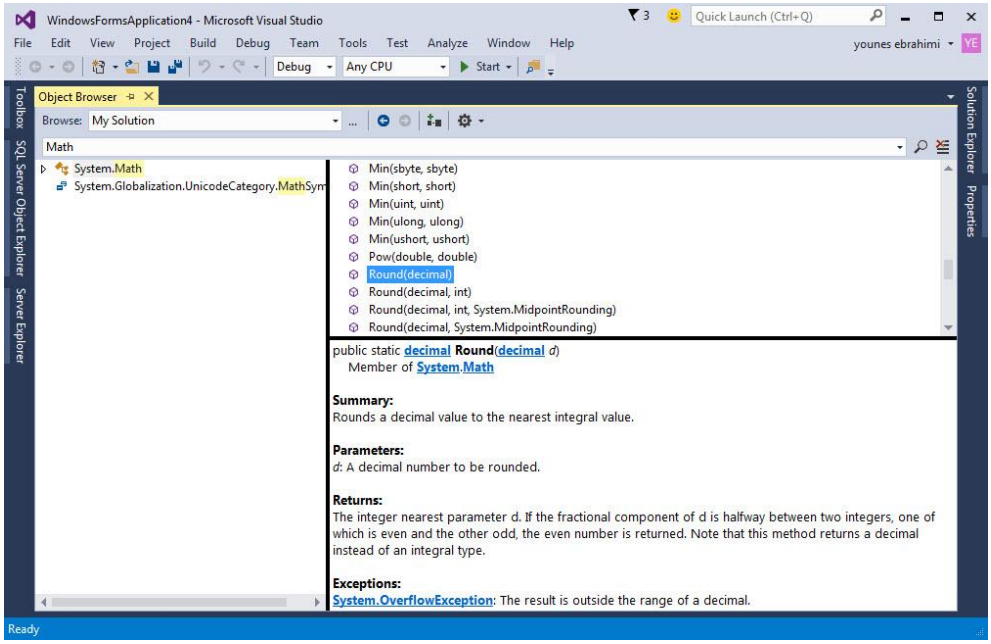
`Object Browser` یک ابزار مهم در ویژوال استودیو برای دسترسی و کسب اطلاعات در مورد فضاهای نام، کلاس ها، متدها، ثابت ها و ... می باشد. برای استفاده از `Object Browser` کافیست که دکمه های ترکیبی `Ctrl+Alt+J` را با هم بفشارید تا پنجره ای به صورت زیر ظاهر شود :



قسمت (۱) در شکل بالا جعبه جستجو است که با استفاده از آن می توانید فضای نام، کلاس، متد، و ... مورد نظرتان را جستجو کنید. قسمت (۲) لیست تمامی فضاهای نام، کلاس ها و موردی را که شما جستجو کرده اید را نشان می دهد .قسمت (۳) محل نمایش اعضای کلاس مورد نظر شماست و قسمت (۴) هم محل نمایش توضیحاتی در مورد اعضای کلاس می باشد. برای روشن شدن مطلب فرض کنید که شما می خواهید اطلاعاتی در مورد کلاس Math به دست آورید. کافیست که نام کلاس را در قسمت جستجو نوشته و دکمه Enter را بزنید:



همانطور که در شکل بالا مشاهده می کنید، نام کلاس در قسمت (۲)، اعضای آن در قسمت (۳) و توضیحات اعضای این کلاس در قسمت (۴) نمایش داده می شود. حال فرض کنید در بین اعضای این کلاس می خواهید اطلاعاتی درباره متد Round() به دست آورید. نام این متد را در بین لیست اعضای کلاس پیدا کرده و بر روی آن کلیک کنید:



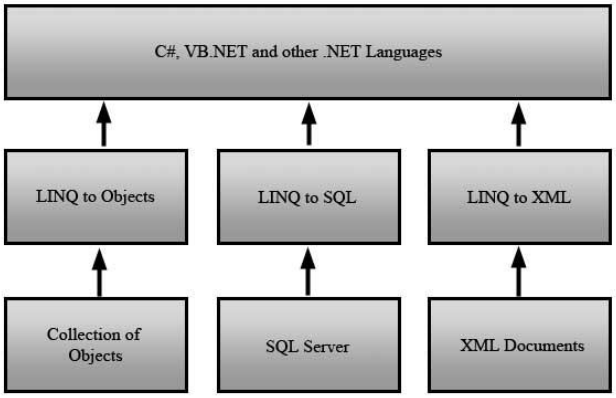
همانطور که در قسمت (۴) مشاهده می کنید، توضیحاتی در مورد این تابع از جمله سطح دسترسی، کاربرد، پارامترهای دریافتی، نوع برگشتی آن و حتی استثنایی که ممکن است در زمان استفاده از این متد رخ دهد به شما نمایش داده می شود. توصیه می کنیم که هر وقت در نحوه استفاده از یک کلاس، متد و ... به مشکل بر خوردید حتما از این ابزار مفید استفاده کنید.

فصل چهارم

LINQ

LINQ چیست؟

LINQ مخفف Language Integrated Query به معنای زبان پرس و جوی یکپارچه است که در دات‌نت نسخه ۳,۵ معرفی شد و به برنامه نویس اجازه می‌دهد داده‌ها را از هر نوع منبع داده‌ای بدون نیاز به دانستن یک زبان دیگر پرس و جو کند. پرس و جو فرایند به دست آوردن داده از منبع داده است. LINQ پرس و جوی داده از منابع داده‌ای مختلف را بسیار راحت کرده است. این زبان با زبان‌های C# و VB آمیخته شده است و برای استفاده از آن چندین کلمه کلیدی و دستور زبان به دو زبان مذکور، اضافه شده است. قبل از ورود LINQ، برنامه نویسان مجموعه کدهای مختلفی برای منابع داده‌ای مختلف می‌نوشتند. به عنوان مثال، برای پرس و جو در یک دیتابیس SQL از دستورات SQL یا برای فایل‌های XML از Xpath استفاده می‌کردند. اما اکنون با استفاده از قدرت LINQ فقط لازم است با کلمات کلیدی LINQ و متدهای آن که در دات‌نت ۳,۵ معرفی شدند آشنا باشید.



چندین نوع LINQ به دلیل وجود providerهای مختلف وجود دارد (شکل بالا). ویژوال استودیو دارای چندین provider مانند LINQ to Objects می‌باشد. در این قسمت تمرکز ما بر LINQ to Objects است که در پرس و جوی مجموعه‌ای از اشیاء در کد شما که رابط IEnumerable<T> را پیاده سازی می‌کنند، مورد استفاده قرار می‌گیرد. مثال‌هایی از این اشیاء، آرایه‌ها و لیست‌ها یا یک مجموعه سفارشی می‌باشد، که شما ایجاد کرده‌اید. LINQ to SQL هم مخصوصاً طوری طراحی شده است که پرس و جوی دیتابیس‌های SQL Server را راحت می‌کند. برای پرس و جوی فایل‌های XML، می‌توان از LINQ to XML استفاده نمود. همچنین می‌توان LINQ را برای پرس و جوی انواع منابع داده‌ای دیگر بسط داد.

تکنیک‌های پرس و جو به کار رفته در درس‌های زیر می‌توانند در انواع مختلف LINQ مورد استفاده قرار بگیرند. شما می‌تونید با استفاده از متدهای الحاقی که در رابط IEnumerable تعریف شده‌اند، از LINQ استفاده کنید. می‌توانید این متدها را مستقیماً فراخوانی کنید، اما باید درباره عبارات لامبدا اطلاعاتی داشته باشید. همچنین می‌توان از عبارات پرس و جو که دستور زبانی شبیه به SQL دارند، استفاده کرد. عبارت‌های پرس و جو ابزار اصلی برای پرس و جو با استفاده از LINQ به شمار می‌آیند. گرچه شما می‌توانید از متدهای الحاقی و عبارات لامبدا هم برای پرس و جو استفاده کنید.

زبان برنامه‌نویسی سی‌شارپ جزء زبان‌های روالمند است. به این معنی که شما برای حل یک مسئله باید قدم به قدم کدهای آن را بنویسید. ولی LINQ جزء زبان‌های غیر روالمند است به این معنی که شما به کامپیوتر می‌گویید که چه چیزی را لازم دارید (صرف نظر از الگوریتم آن) و کامپیوتر آن را برای شما فراهم می‌کند. قبل از LINQ، شما فقط باید با استفاده از زبان‌های روالمند در بین نتایج، پرس و جو می‌کردید. برای مثال فرض کنید که قصد دارید لیست تمامی اعداد زوج داخل یک آرایه را بدست آورید. برای نوشتن راه حل مسئله بالا با استفاده از زبان برنامه‌نویسی سی‌شارپ کد شما چیزی شبیه کد زیر می‌شود:

```
List<int> evenNumbers = new List<int>();
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

foreach (int num in numbers)
{
    if (num % 2 == 0)
        evenNumbers.Add(num);
}
```

با استفاده از کد بالا، شما به کامپیوتر دستور می‌دهید که تک تک مقادیر داخل آرایه را بررسی کند و مقادیری که با شرط داده شده مطابقت دارند را به لیست اضافه کند. حالا مثال بالا را با استفاده از زبان LINQ به شکل زیر بازنویسی کرده‌ایم:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var evenNumbers = from   n in numbers
                  where  n % 2 == 0
                  select n;
```

فعلاً به دستور زبان آن توجهی نکنید، در قسمت‌های بعدی مفصل در این مورد توضیح می‌دهیم. اولین نکته‌ای که در کد بالا قابل توجه است اینست که کد بالا نسبت به کد قبلی ساده تر و قابل فهم تر است.

عبارات پرس و جو

عبارات پرس و جو دستورات ویژه‌ای هستند که برای پرس و جوی یک منبع داده با استفاده از LINQ به کار می‌روند. LINQ مجموعه‌ای از متدهای توسعه یافته است که آنها را صدا می‌زنید و نتایج دلخواه را بدست می‌آورید. این متدها در فضای نامی System.Linq قرار دارند، زمانی که شما قصد دارید از LINQ در پروژه خود استفاده کنید، باید این فضای نامی را به کلاس خود اضافه کنید. در زمان اجرا عبارات پرس و جو به متد معادل خود که توسط CLR قابل فهم است، تبدیل می‌شوند. در درس بعدی شما یاد می‌گیرید که چگونه با استفاده از زبان LINQ در منابع داده‌ای پرس و جویی را انجام دهید.

با یک مثال نحوه‌ی پرس و جو از یک منبع داده‌ای را با استفاده از عبارات پرس و جو تشریح می‌کنیم. به این نکته توجه کنید که در مثال زیر از Linq to Objects استفاده کرده‌ایم، پس برای سادگی از یک آرایه به عنوان منبع داده استفاده می‌کنیم.

```
using System;
using System.Linq;

namespace LinqExample
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            int[] numbers = { 1, 2, 3, 4, 5 };

            var result = from n in numbers
                         select n;

            foreach (var n in result)
            {
                Console.Write(n + " ");
            }
        }
    }
}
```

```
1 2 3 4 5
```

همانطور که قبلاً گفته شد برای استفاده از زبان Linq باید فضای نام System.Linq به پروژه اضافه شود که این کار در خط دوم انجام داده‌ایم. در خط ۱۰ آرایه ای از ۵ عدد صحیح را مشاهده می‌کنید. در خط ۱۳-۱۲ ساده‌ترین عبارت پرس و جویی که می‌توان نوشت، قرار دارد. در درس‌های بعدی عبارات پرس و جوی پیشرفته تری را می‌نویسیم. عبارت پرس و جوی بالا تمامی اعداد داخل آرایه را به عنوان نتیجه بر می‌گرداند. شما با استفاده از متغیر result می‌توانید به آنها دسترسی داشته باشید. ساختار کلی عبارات پرس و جو در زیر نوشته شده است:

```
var query = from rangeVar in dataSource
            <other operations>
            select <projection>;
```

به این نکته توجه کنید که یک عبارت پرس و جو را می‌توان در یک خط کد بنویسید، ولی بهتر آن است که هر قسمت از آن را در یک خط جداگانه نوشت.

هر خط از عبارت پرس و جوی بالا یک عبارت (Clause) نامیده می‌شود. ۷ نوع عبارت وجود دارد که شما می‌توانید از آنها داخل یک عبارت پرس وجو استفاده کنید که شامل عبارات، from، select، where، orderby، let، join و groupby می‌شود.

در این درس فقط از عبارات from و select استفاده می‌شود. عبارات پرس و جو با یک عبارت from شروع می‌شوند. عبارت from از یک متغیر موقت استفاده می‌کند، که وظیفه نگهداری موقت یک مقدار از منبع داده را به عهده دارد، بعد از آن کلمه کلیدی in و سپس نام منبع داده قرار می‌گیرد. این بسیار شبیه مکانیزم حلقه foreach است که در آن هر کدام از اعضای مجموعه در یک متغیر موقت قرار می‌گیرند. متغیر موقت به طور خودکار، بسته به نوع اعضای منبع داده، نوع خود را شناسایی می‌کند.

بعد از عبارت from شما می‌توانید از یک یا چند عبارت where، orderby، let، join استفاده کنید. همچنین می‌توانید از چند عبارت from نیز استفاده کنید که در درس‌های بعدی به آن پرداخته می‌شود. در انتهای عبارت پرس و جو، عبارت select قرار می‌گیرد. بعد از کلمه کلیدی select، نام منبع داده که نوع اعضای برگشتی را مشخص می‌کند قرار می‌گیرد. به عنوان مثال اگر مقدار بعد از عبارت select از نوع int باشد، سپس نوع پرس جو مجموعه‌ای از اعداد صحیح خواهد بود. به این نکته توجه کنید که یک عبارت پرس و جو می‌تواند با یک عبارت group-by خاتمه پیدا کند که در درس‌های بعدی به آن پرداخته می‌شود.

خلاصه آنکه یک عبارت پرس و جوی رایج با یک عبارت from همراه با یک متغیر موقت و نام منبع داده شروع می‌شود، در ادامه عبارت from می‌توان از عبارت‌های where، join، orderby، let و در آخر از عبارت‌های select و group-by استفاده می‌شود. اگر شما با زبان SQL آشنایی داشته باشید دستور زبان عبارات پرس و جو برای شما جالب خواهد بود، زیرا کلمه کلیدی from در اول و کلمه کلیدی select در آخر قرار می‌گیرند (در SQL مکان این کلمات بر عکس است). در این صورت نوع مقادیر منبع داده توسط Visual Studio تشخیص داده می‌شود و شما می‌توانید از طریق ویژگی Intellisene آن نوع را مشاهده کنید.

کلمات کلیدی که در یک عبارت پرس و جو به کار برده می‌شود (مانند from و select) نمونه‌ای از contextual keywordsها یا کلمات کلیدی وابسته هستند. این نوع از کلمات فقط در مکآنها و شرایط خاصی به عنوان کلمه کلیدی رفتار می‌کنند، مثلاً در یک عبارت پرس و جو. به عنوان مثال شما می‌توانید بدون هیچ مشکلی از کلمه select به عنوان نام یک متغیر استفاده کنید به شرطی که از آن متغیر در یک عبارت پرس و جو استفاده نکنید. نتیجه یک پرس و جو از نوع IEnumerable<T> است.

اگر به مثالی که قبلاً توضیح داده شد نگاهی بیندازید، می‌بینید که نتیجه در یک متغیر از نوع var قرار گرفته است، به این معنی که این متغیر برای تشخیص نوع داده‌های بازگشتی، از ویژگی Type Inference استفاده می‌کند. همچنین شما می‌توانید نوع داده‌های برگشتی را به طور صریح مشخص کنید، به شکل زیر:

```
IEnumerable<int> result = from n in numbers
                          select n;
```

در این حالت شما باید نوع داده‌های برگشتی را از قبل بدانید. توصیه می‌شود از کلمه کلیدی var به جای این حالت استفاده کنید تا از برتری‌های زیاد آن، بهره مند شوید. خط ۱۸-۱۵ از مثال قبل مقادیری که توسط پرس و جو برگشت داده شده است را نمایش می‌دهد. برای نمایش مقادیر از یک حلقه ساده foreach استفاده کرده‌ایم. البته یه این نکته توجه داشته باشید که نوع متغیر موقت در مثال قبل var است. در این حالت Compiler نوع عناصر برگشتی را به طور خودکار تشخیص می‌دهد.

در LINQ ویژگی به نام deferred execution (اجرای با تعویق) وجود دارد. به این معنی که عبارات پرس و جو و متدهای LINQ تنها زمانی اجرا می‌شوند که برنامه شروع به خواندن از منبع داده کند یا اینکه بخواهد به یک عنصر از نتیجه پرس وجو دسترسی پیدا کند. در واقع هر عبارت پرس و جو نتیجه یک محاسبه را بر می‌گرداند. نتیجه عبارات تنها یک بار فراخوانی می‌شود و آن زمانی است که کاربر آن را درخواست کند. به عنوان مثال، پرس و جو زمانی اجرا می‌شود که شما بخواهید با استفاده از یک حلقه foreach به نتایج پرس و جو دسترسی داشته باشید.

در درس‌های آینده در رابطه با ویژگی deferred execution توضیحات بیشتری داده خواهد شد. ما با موفقیت اولین عبارت پرس و جوی خود را نوشتیم، اما همانطور که می‌بینید این عبارت فقط داده‌های داخل منبع داده را بر می‌گرداند و کار خاص دیگری را روی آنها انجام نمی‌دهد. در درس‌های آتی تکنیک‌های مختلف دیگر از جمله filtering، ordering، joining و grouping نتایج را یاد می‌گیرید.

استفاده از روش متدی

LINQ مجموعه‌ای از متدهای توسعه یافته هست، که به رابط IEnumerable<T> الحاق شده‌اند. این متدها در فضای نامی System.Linq وجود دارند و عضوی از کلاس استاتیک Enumerable هستند. قبلاً یاد گرفته‌اید که متدهای توسعه یافته نوع خاصی از متدها هستند، که برای توسعه کلاس‌هایی که به سورس آنها دسترسی ندارید، به وجود آمده‌اند. به عنوان مثال شما می‌توانید متدی به نام ToTitleCase() را به کلاس System.String الحاق کنید که با صدا زدن آن به وسیله یک رشته، حروف اول کلمات تشکیل دهنده آن (رشته) به حالت بزرگ نمایش داده شوند. اجازه دهید در مورد متد Select<TSource,TResult>() از فضای نامی System.Linq توضیح دهیم. همانطور که در زیر می‌بینید، این متد به رابط IEnumerable<T> الحاق شده است.

```
public static IEnumerable<TResult> Select<TSource, TResult>(
    this IEnumerable<TSource> source, Func<TSource, TResult> selector)
```

اولین پارامتر از یک متد توسعه یافته، مشخص کننده نوعی است که توسعه می‌دهد. با کلمه کلیدی this شروع و بعد از آن به ترتیب نوع و نام نمونه متغیر قرار می‌گیرد. همانطور که می‌بینید نوع خروجی متد بالا IEnumerable<T> است. این به شما اجازه می‌دهد که متدها را به صورت زنجیره‌ای فراخوانی کنید. همانطور که در درس‌های قبلی گفته شد، برای فراخوانی متدهای LINQ شما باید از عبارات لامبدا استفاده کنید. گرچه شما می‌توانید به جای عبارات لامبدا از متدهای بی نام هم استفاده کنید، ولی استفاده از عبارات لامبدا کوتاه تر و ساده تر بوده و کد شما را خوانا تر می‌کند. در این درس فرض شده است که خواننده اطلاعات خوبی از عبارات لامبدا دارد. در ادامه شما با روش جدیدی برای پرس و جو از منبع داده آشنا می‌شوید و آن استفاده از شکل متدی LINQ است. در این روش شما فقط باید متد مورد نظر خود را صدا زده و اطلاعات مورد نظر خود را صرف نظر از الگوریتم آن بدست آورید.

Net Framework. شامل نماینده‌هایی (delegate) است که می‌توانند متدهایی با تعداد پارامترهای مختلف و نوع خروجی متفاوت در خود نگه دارند. اگر به تعریف متد Select دقت کنید، می‌بینید که پارامتر دوم از این متد شامل نماینده‌ی عمومی با نوع Func<TSource,TResult> است. نماینده می‌تواند متدهایی که پارامتر اول آنها از نوع TSource و نوع خروجی آنها TResult است را قبول کند. به عنوان مثال، Func<string,int> می‌تواند متدی که یک پارامتر از نوع string و خروجی از نوع int دارد را بپذیرد. در زیر لیست نماینده‌هایی را مشاهده می‌کنید که می‌توانید بسته به تعداد پارامترهای متد خود از آنها استفاده کنید.

نماینده	توضیح
Func<T1, TResult>	متدی با ۱ پارامتر ورودی و یک خروجی را در خود ذخیره می‌کند.
Func<T1, T2, TResult>	متدی با ۲ پارامتر ورودی و یک خروجی در خود ذخیره می‌کند.
Func<T1,T2,T3,TResult>	متدی با ۳ پارامتر ورودی و یک خروجی در خود ذخیره می‌کند.
Func<T1,T2,T3,T4, TResult>	متدی با ۴ پارامتر ورودی و یک خروجی در خود ذخیره می‌کند.

با توجه به الگوی موجود در جدول بالا می‌توانید شکل نماینده برای متدهای با تعداد پارامتر بیشتر را حدس بزنید. این الگو را می‌توان تا سقف ۱۶ پارامتر ادامه داد. اگر دوباره به شکل کلی متد Select() نگاهی بیندازید، می‌بینید که دومین پارامتر آن ارجاع به متدی دارد که دارای یک پارامتر و یک مقدار برگشتی است. اجازه دهید مثالی را بررسی کنیم که نحوه‌ی استفاده از متد Select() با عبارات لامبدا را مشخص می‌کند.

```
int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.Select(n => n);

foreach (var n in result)
{
    Console.Write(n + " ");
}
```

```
1 2 3 4 5
```

این نکته را یادآور می‌شویم که اولین پارامتر یک متد الحاقی، در واقع یک پارامتر نیست بلکه مشخص کننده این است که قصد توسعه کدام نوع را داریم. در نتیجه دومین پارامتر که یک عبارت لامبدا را می‌پذیرد، تنها پارامتری است که باید در متد Select() مشخص کنید. در متد Select()، از عبارت لامبدایی استفاده شده است که یک پارامتر صحیح را دریافت می‌کند و خروجی از نوع صحیح را بر می‌گرداند. اگر شما با عبارات لامبدا آشنایی نداشته باشید عبارت موجود در مثال بالا برای شما عجیب است. وظیفه‌ای که عبارت لامبدا در مثال بالا دارد این است که هر عدد موجود در منبع داده را به نتیجه پرس و جو اضافه می‌کند. به این نکته توجه کنید که پرس و جو، نتایج را بدون تغییر آنها بر می‌گرداند. اجازه دهید عبارت لامبدا در مثال قبلی را به شکل مفید تری تغییر دهیم.

```
int[] numbers = { 1, 2, 3, 4, 5 };

var result = numbers.Select(n => n + 1);

foreach (var n in result)
{
    Console.Write(n + " ");
}
```

```
2 3 4 5 6
```

عبارت لامبدا در مثال بالا هر عدد موجود در منبع داده را بدست آورده سپس یک واحد به آن اضافه می‌کند و در آخر آن را به نتیجه پرس و جو اضافه می‌کند. بیشتر متدهای موجود در فضای نامی System.Linq خروجی از نوع IEnumerable<T> دارند. این ویژگی به شما این امکان را می‌دهد که متدها را به صورت زنجیره‌ای فراخوانی کنید. به عنوان مثال، کد زیر را صرف نظر از متدهای جدید آن در نظر بگیرید. نکته مهمی که در خط زیر وجود دارد فراخوانی زنجیره‌ای متدهای LINQ است.

```
var result = numbers.Where(n => n > 3).OrderBy(n => n).Select(n => n);
```

در این درس به شما نحوه‌ی استفاده از متد Select() آموزش داده شد. متدهای زیادی باقی مانده است که در درس‌های بعدی به آن پرداخته می‌شود. حالا شما می‌توانید نتایج بهتری را با استفاده از متد Select() بدست آورید، ولی نکته‌ی مهم این است که حالا شما می‌توانید از شکل متدی LINQ نیز برای پرس و جو از منبع داده استفاده کنید. در واقع کامپایلر برای پرس و جو از منبع داده از شکل متدی استفاده می‌کند. استفاده

از عبارات پرس و جو برای پرس و جو از منبع داده فقط لایه ای برای فراخوانی ساده تر متدهای الحاقی است. به عنوان نمونه، عبارت پرس و جوی زیر را در نظر بگیرید:

```
var result = from     p in persons
             where    p.Age >= 18
             order by p.Name
             select   p.FirstName + " " + p.LastName;
```

در زمان کامپایل عبارت پرس و جوی بالا به دنباله ای از متدهای توسعه یافته، که در فضای نامی System.Linq قرار دارند، تبدیل می‌شود. کد واقعی که کامپایلر از آن برای پرس و جو از منبع داده استفاده می‌کند به شکل زیر است:

```
var result = persons.Where(p => p.Age >= 18)
                    .OrderBy(p => p.Name)
                    .Select(p => p.FirstName + " " + p.LastName);
```

همانطور که می‌بینید متدها به صورت زنجیره‌ای فراخوانی شده‌اند. این به این دلیل است که هر کدام از متدها خروجی از نوع IEnumerable<T> دارند. بعد از فراخوانی متد ()Where، از متد ()OrderBy بر روی خروجی داده‌ها و سپس بر روی مجموعه جدید از متد ()Select استفاده کرده‌ایم. همانطور که مشاهده کردید دو روش برای پرس و جو داده‌ها با استفاده از LINQ وجود دارد:

- استفاده از عبارات پرس و جو.
- استفاده از شکل متدی.

سؤال این است که باید از کدام شکل استفاده کنیم؟ توصیه می‌شود از عبارات پرس و جو استفاده کنید، زیرا نسبت به شکل متدی ساده تر بوده و خوانایی بالاتری دارند. اما بهتر است با شکل متدی هم آشنایی داشته باشید. زیرا تنها راهی است که CLR با استفاده از آن پرس وجوها را انجام می‌دهد.

اجرای با تأخیر (deferred execution)

قبل از پرداختن به درس‌های بعدی به این نکته توجه کنید که LINQ برای انجام پرس و جوها از deferred execution یا اجرای با تأخیر استفاده می‌کند. به زبان ساده به این معنی است که عبارت پرس و جوی واقعی اجرا نمی‌شود مگر زمانی که از نتایج پرس و جو استفاده شود. مثال زیر را در نظر بگیرید:

```
1  int[] numbers = { 1, 2, 3, 4, 5 };
2
3  int i = 3;
4  var result = from n in numbers
5               where n <= i
6               select n;
7  i = 4;
8
9  foreach (var n in result)
10 {
11     Console.WriteLine(n);
12 }
```

```
1
2
```

```
3
4
```

عملگر where را در درس‌های بعدی به طور مفصل توضیح می‌دهیم، ولی برای این مثال باید توضیح مختصری از آن را ارائه دهیم. عملگر where بر اساس یک شرط نتایج حاصل از یک پرس و جو را فیلتر می‌کند. در خط 5 از مثال بالا عملگر where فقط مقادیری را بر می‌گرداند که کوچک‌تر یا مساوی با مقدار i یعنی ۳ باشند. در درس‌های آینده به طور مفصل در رابطه با این عملگر توضیح خواهیم داد. اجازه دهید بر روی deferred execution تمرکز کنیم.

در خط ۳ مقدار ۳ به متغیر i نسبت داده شده است. بنابراین، وقتی عبارت پرس و جوی خطوط ۶-۴ اجرا می‌شود، عملگر where هر عضو منبع داده را با مقدار ۳ مقایسه می‌کند، اگر عضو مورد نظر کوچک‌تر یا مساوی با این مقدار باشد، در نتیجه پرس و جو قرار می‌گیرد.

حالا ممکن است که فکر کنید که متغیر result شامل اعداد ۱، ۲ و ۳ است. اما در حقیقت اینطور نیست. از آنجاییکه در خط ۷ مقدار متغیر i به ۴ تغییر کرده است و به دلیل اینکه Linq از اجرای با تأخیر در بسیاری از عملگرهای خود استفاده می‌کند، مقدار جدید متغیر به طور مستقیم بر روی نتیجه عبارت پرس و جو در خطوط ۴ تا ۶ تأثیر می‌گذارد. همانطور که قبلاً گفته شد، این عبارت پرس وجو فقط یکبار در برنامه اجرا می‌شود. حلقه foreach در خطوط ۹ تا ۱۲ عبارت پرس و جو را اجرا می‌کند و اولین آیتم آنرا بر می‌گرداند. به این نکته توجه کنید چون ما مقدار متغیر iرا تغییر دادیم، عبارت پرس و جو به روز رسانی می‌شود. حلقه foreach به جای نمایش مقادیر ۱ تا ۳ مقادیر ۱ تا ۴ را نمایش می‌دهد (به این خاطر که آخرین مقدار i برابر ۴ است). یکی از کاربردهای اجرای عقب افتاده نیز همین است. برنامه به شما این اجازه را می‌دهد که قبل از اجرای عبارت پرس و جو و بدست آوردن مقادیر آنرا به روزرسانی کنید.

اجرای با تأخیر رفتار پیش‌فرض Linq در اجرای عبارات پرس و جو است، اما شما می‌توانید آن را تغییر داده و کاری کنید که پرس و جو فوراً انجام شود. برای اینکار شما باید از مجموعه‌ای از متدها در فضای نامی System.Linq استفاده کنید که شامل ToArray<T>، ToList<T>، ToDictionary<T>، ToLookUp<T> می‌شود. این متدها جزء متدهای الحاقی رابط IEnumerable<T> هستند. به عنوان مثال، برای تبدیل نتیجه یک عبارت پرس و جو به کلکسیونی از نوع List<T> از متد ToList<T> استفاده می‌شود. مثال زیر نشان می‌دهد که چگونه با استفاده از متد ToList<T> بلادرنگ یک عبارت پرس و جو را اجرا کنیم:

```
1   int[] numbers = { 1, 2, 3, 4, 5 };
2
3   int i = 3;
4
5   var result = (from n in numbers
6                 where n <= i
7                 select n).ToList();
8
9   i = 4;
10
11  foreach (var n in result)
12  {
13      Console.WriteLine(n);
14  }
```

```
1
2
3
```

از آنجاییکه عبارت پرس و جو در کد بالا فوراً اجرا می‌شود، وقتی که برنامه به خط ۷ می‌رسد، نتایج حاصل از پرس و جو در متغیر result قرار می‌گیرند و حتی اگر شما در خط ۹ مقدار متغیر i را تغییر دهید باز هم اعداد ۱ تا ۳ به جای اعداد ۱ تا ۴ در خروجی نمایش داده می‌شوند.

به این نکته توجه کنید که برخی از متدهای توسعه یافته فضای نامی System.Linq باعث اجرای با تأخیر و برخی دیگر از آنها باعث اجرای فوری پرس و جو می‌شوند. مثلاً متد Select() برای اجرای با تأخیر و متد ToList() برای اجرای فوری عبارت پرس و جو به کار می‌رود. مثلاً در پرس و جوی زیر از روش متدی و متد Select() برای به دست آوردن عناصر یک آرایه استفاده شده است:

```
var result = numbers.Select(n => n);
```

از آنجاییکه متد Select() از اجرای با تأخیر استفاده می‌کند، کد فوق تا زمانی که حلقه foreach تمامی نتایج را پیمایش نکند اجرا نمی‌شود. در عوض اگر بعد از متد Select() از متدی که باعث اجرای فوری پرس و جو می‌شود مانند متد ToList() استفاده کنید، عبارت پرس و جو فوراً اجرا می‌شود:

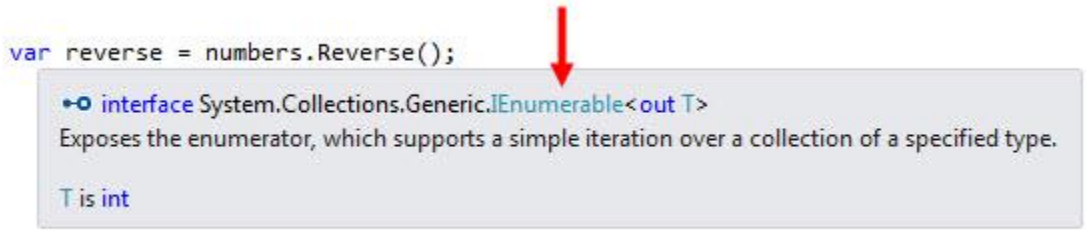
```
var result = numbers.Select(n => n).ToList();
```

پس در نتیجه متد آخر تعیین کننده اجرا یا عدم اجرای با تأخیر عبارت پرس و جو می‌باشد. جدول زیر متدهای فضای نامSystem.Linq را نشان می‌دهد. این جدول همچنین نشان می‌دهد که فراخوانی کدام یک از متدها باعث اجرای با تأخیر عبارات پرس و جو می‌شوند:

متد	اجرا	متد	اجرا
Distinct	با تأخیر	Reverse	با تأخیر
DefaultIfEmpty	با تأخیر	Select	با تأخیر
ElementAt	فوری	SelectMany	با تأخیر
ElementAtOrDefault	با تأخیر	SequenceEqual	فوری
Except	با تأخیر	Single	فوری
First	فوری	SingleOrDefault	فوری
FirstOrDefault	فوری	Skip	با تأخیر
GroupBy	با تأخیر	SkipWhile	با تأخیر
GroupJoin	با تأخیر	Sum	فوری
Intersect	با تأخیر	Take	با تأخیر
Join	با تأخیر	TakeWhile	با تأخیر

Last	فوری	ThenBy	با تأخیر
LastOrDefault	فوری	ThenByDescending	با تأخیر
LongCount	فوری	ToArray	فوری
Max	فوری	ToDictionary	فوری
Min	فوری	ToList	فوری
OfType	با تأخیر	ToLookup	فوری
OrderBy	با تأخیر	Union	با تأخیر
OrderByDescending	با تأخیر	Where	با تأخیر

اگر به خاطر سپردن جدول بالا سخت است، می‌توانید فقط متدهایی را به خاطر بسپارید که استفاده از آنها باعث اجرای با تأخیر می‌شود. اگر نتیجه یک عبارت پرس و جو یک مقدار ساده مانند یک عدد و یا یک شئ باشد، پس این متد از آن دسته ای است که باعث اجرای فوری پرس و جو می‌شوند. اما اگر نتیجه برگشتی از متد از نوع `IEnumerable<T>` باشد، بدانید که بیشتر مواقع این متد از آن نوع دسته متدهایی است که، از اجرای با تأخیر استفاده می‌کند. یک روش برای تشخیص مقدار برگشتی `IEnumerable<T>` این است که با نشانگر ماوس بر روی کلمه کلیدی `var` مکث کنید. که در این صورت با باز شدن یک پنجره popup نوع `IEnumerable<T>` به شما نمایش داده می‌شود. اگر این نوع به شما نمایش داده شد، بدانید که عبارت پرس و جو از اجرای با تأخیر استفاده می‌کند. مثلاً همانطور که در شکل زیر مشاهده می‌کنید از آنجاییکه متد `Reverse()` از اجرای با تأخیر استفاده می‌کند، با مکث بر روی کلمه `var` پنجره popup نوع `IEnumerable<T>` را نشان می‌دهد:



عبارت from

عبارت `from` منبع داده‌ای که می‌خواهید داده‌ای از آن دریافت کنید را، تعریف می‌کند. این عبارت مسئول بدست آوردن عناصر یک منبع داده است. این عبارت در ابتدای یک عبارت پرس وجو می‌آید، بنابراین کل عبارت پرس و جو پس از آن، نوع داده‌ای که بر روی آن کار می‌کنند را می‌داند. این کار باعث می‌شود `Visual Studio` از ویژگی `Intellisense` بعد از این عبارت بهره مند شود، زیرا نوع تک تک اعضای منبع داده را می‌داند. در زیر شکل کلی این عبارت را مشاهده می‌کنید:

```
from dataType rangeVar in dataSource
```

این عبارت با کلمه کلیدی from شروع می‌شود، که جزء Contextual Ketwords در سی‌شارپ است. سپس باید یک متغیر موقت که تک تک اعضای یک مجموعه به ترتیب در آن قرار می‌گیرند را، تعریف کنیم. همچنین شما می‌توانید نوع متغیر موقت را صریحاً مشخص کنید که کاربرد خاص خود را دارد. بعد از اعلان متغیر موقت کلمه کلیدی in همراه با نام منبع داده‌ای که رابط IEnumerable<T> یا IEnumerable را پیاده سازی می‌کند قرار می‌گیرد.

چیزی که در یک پرس و جوی ساده from اتفاق می‌افتد این است که این عبارت به ترتیب اعضای مجموعه را در متغیر موقت قرار می‌دهد. شما می‌توانید عملیات بیشتری را روی این متغیر مانند بررسی کردن یک شرط خاص انجام دهید. در LINQ قابلیتی به نام deferred execution (اجرای با تعویق) وجود دارد. به این معنی است که LINQ اجرا نمی‌شود مگر زمانی که با استفاده از حلقه foreach از نتیجه پرس وجو درخواست داده‌ای را می‌کنید. در زیر مثالی از عبارت from را مشاهده می‌کنید:

```
from int number in numbers
```

عبارت from بالا به کامپیوتر دستور می‌دهد که تک تک اعداد داخل یک مجموعه عددی را بر گرداند (با این فرض که آن مجموعه یک آرایه یا شیء ای است که رابط IEnumerable را پیاده سازی می‌کند). به نوع متغیر موقت در مثال بالا توجه کنید. در بیشتر مواقع، اگر از نوع داده‌های منبع داده مطمئن هستید، می‌توانید از نوشتن این نوع صرف نظر کنید.

```
from number in numbers
```

کامپایلر می‌تواند به صورت خودکار و بسته به نوع داده‌های منبع داده، نوع متغیر موقت را تشخیص دهد. به عنوان مثال اگر مجموعه‌ای داریم که رابط IEnumerable<Person> را پیاده سازی کرده است، نوع متغیر موقت به طور خودکار Person در نظر گرفته می‌شود. اما اگر قصد پرس و جو در مجموعه‌ای از اشیاء مانند ArrayList را دارید بهتر آن است که نوع متغیر موقت که بیانگر نوع عناصر داخل مجموعه است را، صریحاً مشخص نمایید. مثال زیر را در نظر بگیرید:

```
1   using System;
2   using System.Linq;
3   using System.Collections;
4   using System.Collections.Generic;
5
6   namespace FromClause
7   {
8       public class Person
9       {
10          public int    PersonId  { get; set; }
11          public string FirstName { get; set; }
12          public string LastName  { get; set; }
13          public string Gender    { get; set; }
14          public int    Age       { get; set; }
15
16          public Person(string f, string l)
17          {
18              this.FirstName = f;
19              this.LastName = l;
20          }
21      }
22
```

```
23      class Program
24      {
25          static void Main(string[] args)
26          {
27              ArrayList persons = new ArrayList();
28              persons.Add(new Person("John", "Smith"));
29              persons.Add(new Person("Mark", "Chandler"));
30              persons.Add(new Person("Eric", "Watts"));
31
32              var query = from p in persons
33                          select p;
34          }
35      }
36  }
```

اگر مثال بالا را کامپایل کنید با خطای زیر مواجه می‌شوید:

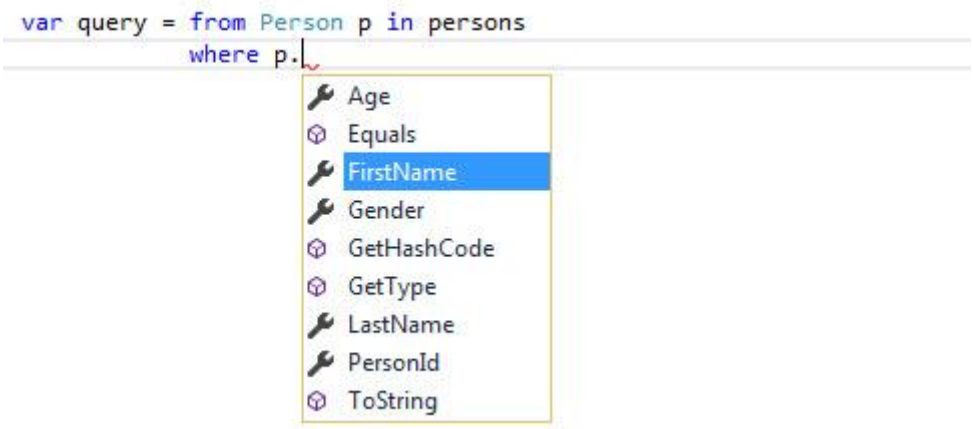
```
Could not find an implementation of the query pattern for source type
'System.Collections.ArrayList'. 'Select' not found. Consider explicitly specifying the type of the range
variable 'p'.
```

به این خاطر که ArrayList می‌تواند شامل اشیایی از انواع مختلف باشد. کامپایلر توصیه می‌کند که نوع عناصر مجموعه را صریحاً مشخص کنید.

برای حل مشکل بالا کافی است نوع متغیر موقت (خط ۳۲) را به شکل زیر مشخص کنید:

```
from Person p in persons
```

LINQ برای انجام پرس و جو هر کدام از اعضای مجموعه را به نوع Person تبدیل می‌کند. اگر شیء ای از مجموعه پیدا شود که قابل تبدیل به نوع Person نباشد یک استثناء رخ می‌شود. عبارت from در ابتدای یک عبارت پرس وجو می‌آید. این کار به Visual Studio امکان می‌دهد که، نوع داده‌هایی که بر روی آنها کار می‌کنید را، تشخیص دهد. همانطور که از شکل زیر پیداست، بعد از عبارت from می‌توانید از مزایای Intellisense بهره مند شوید.



وصل کردن چندین جدول از طریق چندین عبارت from اگر شما دو منبع داده در اختیار دارید، می‌توانید با استفاده از چند عبارت from آنها را به همدیگر وصل کنید. مثال زیر را که باعث اتصال دو مجموعه از اعداد می‌شود را، در نظر بگیرید:

```
int[] numberSet1 = { 1, 2, 3, 4, 5 };
int[] numberSet2 = { 6, 7, 8, 9, 10 };
```

```
var query = from n1 in numberSet1
            from n2 in numberSet2
            select String.Format("{0},{1}", n1, n2);

foreach (var n in query)
{
    Console.WriteLine(n.ToString());
}
```

```
(1,6)
(1,7)
(1,8)
(1,9)
(1,10)
(2,6)
(2,7)
(2,8)
(2,9)
(2,10)
(3,6)
(3,7)
(3,8)
(3,9)
(3,10)
(4,6)
(4,7)
(4,8)
(4,9)
(4,10)
(5,6)
(5,7)
(5,8)
(5,9)
(5,10)
```

در مثال بالا از دو عبارت from که داده‌ها را از دو منبع مختلف واکشی می‌کنند، استفاده کرده‌ایم. در انتها با استفاده از متد Format خروجی را فرمت بندی کرده‌ایم. همانطور که در خروجی می‌بینید، تمامی ترکیب‌های ممکن، در خروجی پرس وجو نمایش داده می‌شوند (ضرب دکارتی). به این ترکیب اصطلاحاً inner join گفته می‌شود.

همچنین شما می‌توانید مستقیماً از عبارت join که بعداً توضیح داده می‌شود، استفاده کنید. به این نکته توجه کنید که لزومی ندارد عبارت from دوم و سوم مستقیماً بعد از عبارت from اول قرار گیرند. شما می‌توانید از پرس وجویی به شکل زیر استفاده کنید:

```
from n1 in numberSet1
where n1 > 2
from n2 in numberSet2
select String.Format("({0},{1})", n1, n2);
```

یک عبارت from تکی، معادل متدی در فضای نامی System.Linq ندارد. شما می‌توانید از متد ()Select و دیگر متدها استفاده کنید. ولی برای چند عبارت from می‌توانید از متد ()SelectMany که در درس‌های بعدی به آن پرداخته می‌شود، استفاده کنید.

ایجاد ارتباط بین کلاس‌ها

عبارات پرس و جوی متشکل از چند عبارت from می‌توانند به شکل موثری در کلاس‌های مرتبط استفاده شوند. اما چنین کلاس‌هایی ابتدا باید به شکل مطلوب ساختار بندی شوند. به عنوان مثال یک مشتری می‌تواند چندین سفارش داشته باشد که این رابطه را می‌توان به عنوان یک رابطه یک به چند در نظر گرفت (یک مشتری با چندین سفارش). اجازه دهید ابتدا کلاس مشتری و سفارش را به شکل مناسب تعریف کنیم:

```
class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public List<Order> Orders { get; set; }
}
```

برای سادگی مثال، کلاس مشتری از سه خاصیت تشکیل شده است. خاصیت سوم رابطه کلاس مشتری با کلاس سفارش را تعریف می‌کند. این خاصیت شامل لیست سفارش‌های مشتری است. کلاس سفارش در زیر تعریف شده است.

```
class Order
{
    public string Name { get; set; }
    public int Quantity { get; set; }
}
```

اجازه دهید چند شیء مشتری ایجاد کرده و چند سفارش را به خاصیت Orders آنها اختصاص دهیم.

```
class Program
{
    static void Main(string[] args)
    {
        List<Customer> customers = new List<Customer>();

        Customer customer1 = new Customer()
        {
            FirstName = "John",
            LastName = "Smith",
            Orders = new List<Order>()
            {
                new Order() { Name = "Pizza", Quantity = 5 },
                new Order() { Name = "Chicken", Quantity = 3 },
                new Order() { Name = "Salad", Quantity = 7 }
            }
        };


        Customer customer2 = new Customer()
        {
            FirstName = "Allan",
            LastName = "York",
            Orders = new List<Order>()
            {
                new Order() { Name = "Orange Juice", Quantity = 2 },
                new Order() { Name = "Soda", Quantity = 4 }
            }
        };

        Customer customer3 = new Customer()
        {
            FirstName = "Henry",
            LastName = "Helms",
            Orders = new List<Order>()
```

```
            {
                new Order() { Name = "Speghetti", Quantity = 7 },
                new Order() { Name = "Nachos", Quantity = 13 }
            }
        };

        customers.Add(customer1);
        customers.Add(customer2);
        customers.Add(customer3);

        var results = from c in customers
                      from o in c.Orders
                      select new { c.FirstName, c.LastName, o.Name, o.Quantity };

        foreach (var result in results)
        {
            Console.WriteLine("Name: {0} {1}, Order: {2}, Quantity: {3}",
                result.FirstName, result.LastName, result.Name, result.Quantity);
        }
    }
}
```

```
Name: John Smith, Order: Pizza, Quantity: 5
Name: John Smith, Order: Chicken, Quantity: 3
Name: John Smith, Order: Salad, Quantity: 7
Name: Allan York, Order: Orange Juice, Quantity: 2
Name: Allan York, Order: Soda, Quantity: 4
Name: Henry Helms, Order: Speghetti, Quantity: 7
Name: Henry Helms, Order: Nachos, Quantity: 1
```

همانطور که در خروجی می‌بینید، سفارشات تمامی مشتریان نشان داده شده است. همچنین برای مرتب کردن خروجی می‌توان از عملگر group-by استفاده کرد. در خط ۴۰-۷ مثال بالا، ۳ شیء Customer ایجاد کرده‌ایم که هر کدام دارای سفارشات مخصوص به خود هستند. سپس هر کدام از این سه شیء را به لیست مشتریان اضافه کرده‌ایم.

خطوط ۴۸-۴۶ پرس وجویی با دو عبارت from را نشان می‌دهد. اولین عبارت from یک شیء Customer را از لیست مشتریان انتخاب می‌کند. دومین عبارت from تک تک اعضای خاصیت Orders مشتری انتخاب شده را بر می‌گرداند. عبارت select با استفاده از projection نام و نام خانوادگی مشتری، نام و تعداد سفارش را انتخاب می‌کند.

خطوط ۵۴-۵۰ نتیجه را در خروجی چاپ می‌کنند. طراحی کلاس‌ها به شکل بالا بسیار مناسب است زیرا تشخیص ارتباط بین اشیاء به راحتی امکان پذیر است. جداول موجود در پایگاه داده معمولاً از این نوع ارتباطات استفاده می‌کنند. ابزارهایی در Visual Studio وجود دارند که جدوال و ارتباطات بین آنها را به کلاس‌ها و ارتباط بین آنها تبدیل می‌کنند. در مبحث LINQ To SQL به این امر پرداخته شده است.

عبارت Select

عبارت select در LINQ با استفاده از projection داده‌ها را در نتیجه پرس و جو قرار می‌دهد. طرح ریزی (projection) فرایند تبدیل شیء به فرم جدیدی است. عبارت select در LINQ باید در پایان پرس و جو نوشته شود.

شما می‌توانید عملیات زیادی را بر روی مقدار پرس و جو شده انجام دهید از جمله، بدست آوردن یک خاصیت مشخص یا ساخت نوع بی نام که از خاصیت‌های شیء پرس و جو شده، استفاده می‌کند. نتیجه یک پرس و جو مجموعه‌ای است که رابط IEnumerable<T> را پیاده سازی می‌کند

که در آن T نوع خروجی عبارت Select است. به عنوان نمونه، اگر عبارت Select به یک شیء person از مجموعه اشیاء Person اشاره کند، مجموعه نهایی، مجموعه‌ای است که رابط IEnumerable<Person> را پیاده سازی می‌کند. اگر دستور select تنها خصوصیت رشته‌ای FirstName از هر شخص را انتخاب کند، مجموعه نهایی رابط IEnumerable<string> را پیاده سازی می‌کند.

برای نمایش نتایج یک پرس و جو می‌توانید از یک حلقه foreach استفاده کنید. همچنین شما با انتخاب متغیر موقت می‌توانید شیء پرس وجو شده را به طور کامل انتخاب کنید. این کار باعث انتخاب شیء پرس و جو شده بدون هیچ تغییری می‌شود.

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var results = from number in numbers
              select number;

foreach (var n in results)
{
     Console.Write("{0, -2}", n);
}
```

```
1 2 3 4 5 6 7 8 9 10
```

شما می‌توانید انواع مختلفی از عبارت را در دستور select بنویسید. به عنوان مثال می‌توانید تک تک اعداد موجود در آرایه ای از اعداد را بدست آورده و سپس یک واحد به آنها اضافه کنید.

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var results = from number in numbers
              select number + 1;

foreach (var n in results)
{
     Console.Write("{0, -2}", n);
}
```

```
2 3 4 5 6 7 8 9 10 11
```

همچنین در طول فرایند انتخاب می‌توانید یک خصوصیت یا ترکیبی از خصوصیات را انتخاب کنید. به عنوان مثال، شما می‌توانید خصوصیت FirstName از هر شیء Person را انتخاب کنید.

```
List<Person> people = new List<Person>
{
    new Person("Johnny" , "Smith"),
    new Person("Mark"   , "Lawrence"),
    new Person("Jessica", "Fisher"),
    new Person("Danny"  , "Mayer"),
    new Person("Raynold", "Alfonzo")
};

var firstNames = from   person in people
                 select person.FirstName;
```

```
Johnny
Mark
Jessica
Danny
Raynold
```

همچنین شما می‌توانید خصوصیات مختلف را با هم ترکیب کنید. به عنوان نمونه می‌توانید خصوصیات FirstNme و LastName از هر شیء Person را با هم ترکیب کرده و به عنوان نام کامل فرد برگردانید.

```
var fullNames = from person in people
                select person.FirstName + " " + .person.LastName;
```

```
Johnny Smith
Mark Lawrence
Jessica Fisher
Danny Mayer
Raynold Alfonzo
```

عبارت select بالا باعث انتخاب نام فرد همراه با یک فضای خالی و سپس نام خانوادگی می‌شود. شما می‌توانید از متدهای مختلف برای تغییر نتیجه خروجی استفاده کنید. به عنوان مثال قصد داریم نام افراد مختلف را به صورت حروف بزرگ در خروجی قرار دهیم.

```
var namesInCaps = from person in people
                    select person.FirstName.ToUpper();
```

```
JOHNNY
MARK
JESSICA
DANNY
RAYNOLD
```

همچنین می‌توانید مقادیری که مستقیماً در منبع داده نیستند را انتخاب کنید. به عنوان نمونه، می‌توان لیستی از اندیس‌ها را انتخاب کرد و به وسیله آنها مقادیر موجود در آرایه دیگر را در خروجی قرار داد.

```
int[] indices = { 0, 1, 2, 3, 4 };
string[] words = { "Example1", "Example2", "Example3", "Example4", "Example5" };

var result = from index in indices
             select words[index];
```

```
Example1
Example2
Example3
Example4
Example5
```

می‌توان انواع بی نامی را در طی عملیات select ساخت، که شامل خصوصیات مشخصی از هر شیء هستند. به عنوان نمونه، فرض کنید یک شیء Person داریم که دارای خصوصیات FirstName، LastName، Age، Gender باشد و ما در طی عملیات select فقط خصوصیات FirstName و LastName آن را لازم داریم، برای حل این مشکل می‌توان یک نوع بی نام را در عبارت select ساخت که شامل این خصوصیات باشد.

```
var annonymous = from person in people
                 select new { person.FirstName, person.LastName };

foreach (var p in annonymous)
{
    Console.WriteLine(p.FirstName + " " + p.LastName);
}
```

نوع نتیجه خروجی، نوع بی نامی است که شامل خصوصیات انتخاب شده است. همچنین شما می‌توانید خصوصیات جدیدی را بر مبنای خصوصیات شیء پرس و جو شده ایجاد کنید.

```
var results = from person in people
              select new { FN = person.FirstName, LN = person.LastName };

foreach (var p in results)
{
    Console.WriteLine(p.FN + " " + p.LN);
}
```

```
Johnny Smith
Mark Lawrence
Jessica Fisher
Danny Mayer
Raynold Alfonzo
```

دستور select در عبارت پرس و جوی بالا مقدار خاصیت FirstName و LastName را به خاصیت‌های جدید با نام‌های متفاوت نسبت می‌دهد. نوع این خاصیت‌های جدید را کامپایلر با توجه به مقادیر نسبت داده شده به آنها تعیین می‌کند (type inference). نوع بی نام ایجاد شده این خاصیت‌های جدید را جذب می‌کند.

```
var results = from person in people
              select new { Upper = person.FirstName.ToUpper(),
               Lower = person.FirstName.ToLower() };

foreach (var p in results)
{
   Console.WriteLine("Upper={0}, Lower={1}", p.Upper, p.Lower);
}
```

```
Upper=JOHNNY, Lower=johnny
Upper=MARK, Lower=mark
Upper=JESSICA, Lower=jessica
Upper=DANNY, Lower=danny
Upper=RAYNOLD, Lower=raynold
```

عبارت select بالا مقدار خاصیت FirstName را به حروف بزرگ تبدیل می‌کند و آن را به یک خاصیت جدید با نام Upper نسبت می‌دهد، در ادامه مقدار همین خاصیت را به حروف کوچک تبدیل می‌کند و آن را به یک خاصیت جدید به نام Lower نسبت می‌دهد. این دو خاصیت جدید در نوع بی نام ساخته شده، قرار می‌گیرند. برای استفاده از این خاصیت‌های جدید از یک حلقه foreach استفاده می‌کنیم.

```
var results = from person in people
              select new { FullName = person.FirstName + " " + person.LastName };

foreach (var p in results)
{
   Console.WriteLine(p.FullName);
}
```

```
Johnny Smith
Mark Lawrence
Jessica Fisher
Danny Mayer
Raynold Alfonzo
```

عبارت select در پرس و جوی بالا یک خاصیت جدید به نام FullName ایجاد می‌کند. سپس جمع مقادیر خاصیت‌های FirstName و LastName را در آن قرار می‌دهد. نوع بی نام یک خاصیت به نام FullName دارد که به طور خودکار مقدار کامل نام افراد را در خود ذخیره می‌کند.

```
var results = from person in people
              select new { person.Age, FullName = person.FirstName + " " + person.LastName };

foreach (p in results)
{
    Console.WriteLine("FullName={0}, Age={1}", p.FullName, p.Age);
}
```

```
FullName=Johnny Smith, Age=22
FullName=Mark Lawrence, Age=24
FullName=Jessica Fisher, Age=19
FullName=Danny Mayer, Age=21
FullName=Raynold Alfonzo, Age=25
```

پرس و جوی قبل شامل یک عبارت select است که یک نوع بی نام شامل دو خاصیت ایجاد می‌کند که خاصیت اول سن افراد را نشان می‌دهد و خاصیت دوم ترکیبی از نام و نام خانوادگی را در خود ذخیره می‌کند.

متد Select()

روش پرس و جوی دیگر که دقیقاً معادل با عبارت select می‌باشد، متد Select() است. این متد یک عبارت لامبدا را به عنوان پارامتر می‌پذیرد. این عبارت لامبدا تمامی عناصر منبع داده را طرح ریزی می‌کند. در مثال زیر با استفاده از متد Select() تمامی عناصر مجموعه people را بدست آورده‌ایم.

```
var result = people.Select(p => p);
```

یک عبارت لامبدا را به عنوان آرگومان به متد Select() ارسال کرده‌ایم. عبارت لامبدا یک پارامتر به نام p دارد که در هر بار یک شیء از نوع Person در آن قرار می‌گیرد، این شیء در صورتی در نتیجه پرس و جو قرار می‌گیرد که با شرطی که در سمت راست عبارت لامبدا نوشته شده است، مطابقت داشته باشد. در پرس و جوی بالا شرط خود شیء انتخابی است پس تمامی عناصر مجموعه people در نتیجه پرس و جو قرار می‌گیرند. در زیر نمونه‌های بیشتری از کاربرد عبارت Select را مشاهده می‌کنید. در این مثال‌ها عبارات لامبدا مختلفی به متد Select() ارسال شده است:

۱ – انتخاب خاصیت FirstName از تمامی عناصر people:

```
var firstNames = people.Select(p => p.FirstName);
```

۲ – دو خاصیت FirstName و LastName از هر عنصر مجموعه را با هم ترکیب کرده و در نتیجه پرس و جو قرار می‌دهد:

```
var fullNames = people.Select(p => p.FirstName + " " + p.LastName);
```

۳ – ابتدا یک نوع بی نام شامل یک خاصیت به نام FullName را ایجاد کرده و ترکیب دو خاصیت FirstName و LastName را در آن قرار می‌دهد:

```
var annonymous = people.Select(p => new { FullName = p.FirstName + " " + p.LastName });
```

نسخه دیگری از متد Select() نیز وجود دارد، این نسخه یک عبارت لامبدا شامل دو پارامتر ورودی می‌پذیرد، اولین پارامتر بیانگر عنصر پرس و جو شده و دومین پارامتر نشان دهنده ایندکس آن عنصر در منبع داده است:

```
var personsWithIndex = people.Select((p, i) => new { Person = p, Index = i });
```

پرس و جوی زیر معادل آن است:

```
var personsWithIndex = from p in people
                       select new { Person = p, Index = people.IndexOf(p) };
```

سپس شما می‌توانید در هنگام چاپ مقادیر، اندیس هر عنصر را نیز بنویسید:

```
foreach (var p in personsWithIndex)
{
    Console.WriteLine(String.Format("[{0}] {1}", p.Index, p.Person.FirstName));
}
```

```
[0] John
[1] Mark
[2] Lisa
```

شما می‌توانید از عبارات پرس و جو برای خوانایی بیشتر کد خود استفاده کنید یا متدهای Linq را با استفاده از شکل متدی Linq فراخوانی کنید. در این حالت باید از عبارات لامبدا استفاده کنید.

عبارت where

شما می‌توانید نتیجه پرس و جو را با استفاده از عبارت where فیلتر کنید. بعد از کلمه کلیدی where یک شرط (یا مجموعه‌ای شروط) قرار می‌گیرد، هر عنصری که در نتیجه پرس و جو قرار می‌گیرد با این شرط مطابقت دارد. نکته بسیار مهمی که اهمیت دارد این است که برای نوشتن شرط می‌توانید از متدهایی که در کتابخانه کلاس .Net Framework قرار دارند، استفاده نمایید. این درس به شما نشان می‌دهد که چگونه با استفاده از عبارت where نتیجه پرس و جو را فیلتر نمایید.

با استفاده از عملگرهای مقایسه ای می‌توان مقدار یا خاصیتی از یک شیء را با یک مقدار دیگر مقایسه کرد. به عنوان مثال، قصد داریم عناصر یک آرایه عددی را که بزرگ‌تر از ۵ هستند را، انتخاب نماییم. پرس و جویی که می‌نویسیم به شکل زیر است:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var greaterThanFive = from  number in numbers
                      where  number > 5
                      select number;
```

در مثال بالا شرط داخل عبارت where مشخص می‌کند که فقط مقادیری از آرایه، که بزرگ‌تر از ۵ باشند، در نتیجه پرس و جو قرار گیرند. با استفاده از عملگرهای منطقی، شرط‌های پیچیده تری را می‌توان نوشت:

```
var sixToTen = from number in numbers
               where number > 5 && number <= 10
               select number;
```

پرس و جوی بالا مقادیری که بزرگ‌تر از ۵ و کوچک‌تر مساوی با ۱۰ هستند را، در نتیجه پرس و جو قرار می‌دهد. اگر مجموعه‌ای از اشیاء با تعدادی خاصیت داشته باشید، می‌توان از خاصیت‌های آنها برای نوشتن شرط عبارت where استفاده کنید:

```
List<Person> people = GetPersonList();

var smiths = from person in people
               where person.LastName == "Smith"
               select person;
```

پرس و جوی بالا اشخاصی که مقدار خاصیت LastName آنها برابر Smith هست را در نتیجه پرس و جو قرار می‌دهد. همانطور که گفته شد می‌توان از متدهای دات نت برای نوشتن شرط استفاده کرد، به عنوان مثال قصد داریم تمامی اشخاصی که مقدار خاصیت lastname آنها با حرف R شروع می‌شود را انتخاب کنیم. عبارت پرس وجو زیر این کار را انجام می‌دهد:

```
var startsWithR = from person in people
                    where person.LastName.StartsWith("R")
                    select person;
```

در پرس و جوی بالا از متد StartsWith() کلاس String استفاده شده است. این متد در صورتی که رشته با آرگومان ارسالی به این متد شروع شود true را بر می‌گرداند. همچنین، می‌توان از متد توسعه یافته Where() از فضای نامی System.Linq برای نوشتن شرط استفاده کرد. این متد یک پارامتر از نوع عبارات لامبدا می‌پذیرد، این عبارت لامبدا یک پارامتر دارد که عنصر پرس و جو شده در آن قرار می‌گیرد و بدنه آن شامل یک شرط است. خروجی این متد مجموعه‌ای از اشیاء است.

```
var greaterThanFive = numbers.Where(number => number > 5);
```

در بدنه عبارت لامبدا یک دستور شرطی ساده قرار دارد که بررسی می‌کند که آیا مقدار عدد بزرگ‌تر از ۵ است یا نه. در زیر مثالی دیگر را مشاهده می‌نمایید که اشخاصی که خصوصیت LastName آنها با حرف R شروع می‌شود را در نتیجه پرس و جو قرار می‌دهد.

```
var startsWithR = people.Where(person => person.LastName.StartsWith("R"));
```

می‌توان از یکی از سربارگذاری‌های این متد که یک عبارت لامبدا با ۲ پارامتر را قبول می‌کند نیز استفاده کنید. اولین پارامتر نماینده‌ی هر یک از اشیاء مجموعه می‌باشد، و پارامتر دوم اندیس آن شیء در مجموعه را تعیین می‌کند. مثال زیر اشیایی از مجموعه را بر می‌گرداند که اندیس آنها در مجموعه زوج باشد.

```
var evenIndices = numbers.Where((number, index) => index % 2 == 0);
```

از متد Where() زمانی می‌توان استفاده کرد که، بخواهید یک مجموعه را بر اساس یک شرط مشخص فیلتر کنید.

عبارت orderby

LINQ این امکان را فراهم کرده است که ترتیب نتایج پرس و جو را تغییر دهید. برای این کار از عبارت orderby استفاده می‌کنیم. در هنگام نوشتن این عبارت باید یک مقدار یا خاصیت را به عنوان کلید مرتب سازی مشخص کنیم. به عنوان مثال، فرض کنید آرایه ای از اعداد را ایجاد کرده‌ایم و

قصد داریم روی این آرایه پرس و جویی را انجام دهیم و سپس نتیجه پرس و جو را از بزرگ‌ترین به کوچک‌ترین عدد مرتب کنیم، می‌توانیم از پرس و جویی به شکل زیر استفاده کنیم:

```
int[] numbers = { 5, 1, 6, 2, 8, 10, 4, 3, 9, 7 };

var sortedNumbers = from number in numbers
                    orderby number
                    select number;
```

مرتب سازی نتایج پرس و جوی بالا را با استفاده از عبارت orderby انجام داده‌ایم و کلید مرتب سازی را خود عدد انتخاب کرده‌ایم. هنگامی که برای کلید مرتب سازی خود شیء را انتخاب می‌کنید رفتار پیش‌فرض آن شیء در نظر گرفته می‌شود. به عنوان مثال، عبارت بالا مقدار هر شیء پرس و جو شده را از لحاظ کوچک‌تر یا بزرگ‌تر بودن با شیء پرس و جو شده دیگر مقایسه می‌کند. مرتب سازی پیش‌فرض این عبارت از پایین‌ترین به بالاترین مقدار است. با استفاده از کلمه کلیدی ascending می‌توان صراحتاً تعیین کرد که نتیجه پرس و جو به صورت صعودی مرتب شود.

```
var sortedNumbers = from    number in numbers
                    orderby number ascending
                    select  number;
```

برای معکوس کردن نتیجه یعنی مرتب کردن نتایج به صورت نزولی از کلمه کلیدی descending استفاده می‌شود.

```
var sortedNumbers = from    number in numbers
                    orderby number descending
                    select  number;
```

در هنگام کار با انواع پیچیده تر مانند کلاس‌ها، می‌توانید از خصوصیت آنها به عنوان کلید مرتب سازی استفاده کنید. به عنوان مثال، فرض کنید مجموعه‌ای از اشیاء که همگی دارای نوع Person هستند در اختیار داریم. این نوع دارای خصوصیاتی به نام‌های FirstName، LastName و Age است. با استفاده از پرس و جوی زیر کلکسیون را از جوان‌ترین به مسن‌ترین فرد مرتب می‌کنیم:

```
var sortedByAge = from    person in people
                  orderby person.Age
                  select  person;
```

اگر قصد دارید نتیجه عبارت بالا را از مسن‌ترین به جوان‌ترین تغییر دهید در جلوی کلید از کلمه کلیدی descending استفاده کنید. همچنین می‌توان نتیجه را بر اساس حرف اول نام خانوادگی اشخاص مرتب کنیم.

```
var sortedByFirstName = from    person in people
                        orderby person.LastName
                        select  person;
```

اگر دو شخص با نام خانوادگی یکسان در مجموعه داشته باشیم مرتب سازی آنها به چه شکلی است؟ برای حل این مشکل می‌توانید چند کلید را برای مرتب سازی تعیین کنید.

```
var sortedByFnThenLn = from     person in people
                       orderby person.LastName, person.FirstName
                       select  person;
```

کلیدها را با استفاده از کاما از یکدیگر جدا می‌کنیم. پرس و جو بالا نتیجه را بر اساس نام خانوادگی اشخاص مرتب می‌کند و اگر دو شخص نام خانوادگی یکسانی داشته باشند با استفاده از خاصیت نام، آنها را مرتب می‌کند. همچنین می‌توانید کلیدهای بیشتری را نیز مشخص کنید. مثلاً برای زمانی که دو شخص نام یکسانی داشته باشند. به این نکته توجه کنید که کلمات کلیدی ascending و descending فقط روی یک مقدار در عبارت orderby عمل می‌کنند. به عنوان مثال، پرس و جوی زیر را در نظر بگیرید:

```
var orderByResults = from    person in people
                     orderby person.LastName, person.FirstName descending
                     select  person;
```

کلمه کلیدی descending فقط بر روی مقدار خاصیت پیش از خود (در مثال بالا FirstName) تأثیر می‌گذارد. مقدار خاصیت LastName به طور پیش‌فرض به صورت صعودی مرتب می‌شود. برای جلوگیری از آشفتگی توصیه می‌شود به طور صریح کلمه کلیدی ascending را بنویسید:

```
var orderByResults = from    person in people
                     orderby person.LastName ascending, person.FirstName descending
                     select  person;
```

برای اینکه رفتار پیشفرض یک کلاس را هنگام مرتب سازی تعیین کنید باید رابط IComparable<T> را در کلاس پیاده سازی نمایید. در مثال زیر کلاس Person رابط IComparable<T> را پیاده سازی کرده است:

```
1   using System;
2   using System.Linq;
3   using System.Collections.Generic;
4
5   namespace OrderByClause
6   {
7       class Person : IComparable<Person>
8       {
9           public string FirstName { get; set; }
10          public string LastName  { get; set; }
11          public int    Age       { get; set; }
12
13          public Person(string f, string l, int a)
14          {
15              FirstName = f;
16              LastName  = l;
17              Age       = a;
18          }
19
20          public int CompareTo(Person other)
21          {
22              if (this.Age > other.Age)
23                  return 1;
24              else if (this.Age < other.Age)
25                  return -1;
26              else
27                  return 0;
28          }
29      }
30
31      class Program
32      {
33          static void Main(string[] args)
34          {
35              List<Person> people = new List<Person>()
36              {
37                  new Person("Peter" , "Redfield", 32),
38                  new Person("Marvin", "Monrow"  , 17),
```

```
                new Person("Aaron" , "Striver" , 25)
            };

            var defaultSort = from    person in people
                              orderby person
                              select  person;

            foreach (var person in defaultSort)
            {
                Console.WriteLine(String.Format("{0} {1} {2}",
                    person.Age, person.FirstName, person.LastName));
            }
        }
    }
}
```

```
17 Marvin Monrow
25 Aaron Striver
32 Peter Redfield
```

خط ۷ بیانگر پیاده سازی رابط IComparable<T> است. باید نوع اشیایی که با یکدیگر مقایسه می‌شوند را به جای T قرار دهیم. در مثال، این نوع برابر Person است. پیاده سازی این رابط مستلزم پیاده سازی یک متد با نام ()CompareTo است. این متد یک شیء به عنوان پارامتر قبول می‌کند که باید با شی جاری مقایسه شود سپس یک عدد صحیح به عنوان نتیجه بر می‌گرداند. در مثال بالا این متد در خطوط ۲۰ تا ۲۸ پیاده سازی شده است. در داخل متد، مقدار خاصیت Age شیء جاری با خاصیت مشابه شیء دیگر (که در این مثال از نوع Person است) مقایسه شده است. اگر مقدار شیء جاری از مقدار دیگری بزرگ‌تر باشد عددی بزرگ‌تری از صفر، اگر مساوی باشد مقدار صفر و اگر کوچک‌تر از صفر باشد مقداری منفی به عنوان نتیجه برگشت داده می‌شود.

با توجه به اینکه این کلاس رابطه IComparable<T> را پیاده سازی کرده است می‌توانیم پرس و جوی خود را ساده تر بنویسیم. چون در داخل متد ()CompareTo از خاصیت Age برای مقایسه استفاده کرده‌ایم، اگر در داخل پرس و جو، کلید مرتب سازی نوشته نشود، به طور پیشفرض خاصیت Age به عنوان کلید مرتب سازی در نظر گرفته می‌شود. متدهای ()OrderBy و ()OrderByDescending موجود در فضای نامی System.Linq معادل عبارت orderby هستند. متد ()OrderBy نتیجه پرس و جو را بر اساس یک کلید به صورت صعودی و OrderByDescending نتیجه را به صورت نزولی مرتب می‌کنند. برای مشخص کردن کلید مرتب سازی از یک عبارت لامبدا استفاده می‌کنیم.

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace OrderByClause
{
    class Person
    {
        public string FirstName { get; set; }
        public string LastName  { get; set; }
        public int Age          { get; set; }

        public Person(string f, string l, int a)
        {
            FirstName = f;
            LastName  = l;
            Age       = a;
        }
    }
```

```
    class Program
    {
        private static List<Person> GetPersonList()
        {
            List<Person> people = new List<Person>()
            {
                new Person("Peter" , "Redfield", 32),
                new Person("Marvin", "Monrow"  , 17),
                new Person("Aaron" , "Striver" , 25)
            };

            return people;
        }

        static void Main(string[] args)
        {
            List<Person> people = GetPersonList();

            Console.WriteLine("Persons OrderBy FirstName : ");

            var orderByQuery3 = people.OrderBy(person => person.FirstName);
            foreach (var person in orderByQuery3)
            {
                Console.WriteLine(String.Format("{0}", person.FirstName));
            }

            Console.WriteLine("\nPersons OrderByDescending Age : ");

            var orderByQuery4 = people.OrderByDescending(person => person.Age);
            foreach (var person in orderByQuery4)
            {
                Console.WriteLine(String.Format("{0}", person.Age));
            }
        }
    }
}
```

متدهای ThenBy() و ThenByDescending()

اگر چندین کلید مرتب سازی باشید باید از متدهای ThenBy() و ThenByDescending() استفاده کنید. به عنوان مثال پرس و جوی زیر:

```
var orderByQuery5 = from p in people
                    orderby p.LastName, p.FirstName
                    select p;
```

معادل پرس و جو به شکل زیر است:

```
people.OrderBy(p => p.LastName).ThenBy(p => p.FirstName);
```

اگر قصد داشته باشید با استفاده از کلید دوم نتیجه را به صورت نزولی مرتب کنید باید از متد ThenByDescending() استفاده کنید:

```
var orderByQuery6 = people.OrderBy(p => p.LastName).ThenByDescending(p => FirstName);

var orderByQuery7 = people.OrderByDescending(p => p.LastName)
                          .ThenByDescending(p => p.FirstName);
```

به این نکته توجه کنید که متدهای ()ThenBy و ()ThenByDescending عضوی از رابط IOrderedEnumerable<T> هستند. متد ()OrderBy کلکسیونی را بر می‌گرداند که این رابط را پیاده سازی می‌کند. پس قبل از فراخوانی متدهای ()ThenBy و ()ThenByDescending باید متد ()OrderBy یا ()OrderByDescending را فراخوانی کنید.

استفاده از رابط IComparer<T>

شکل دیگری از ۴ متد مرتب سازی پیشین نیز وجود دارد، که شیء ای از نوع IComparer<T> را به عنوان آرگومان قبول می‌کنند. یک کلاس را ایجاد و رابط IComparer<TKey> را در آن پیاده سازی می‌کنیم. با این کار کلاس ایجاد شده مقایسه پذیر خواهد بود.

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace OrederByClause
{
    class Person
    {
        public string FirstName { get; set; }
        public string LastName  { get; set; }
        public int    Age       { get; set; }

        public Person(string f, string l, int a)
        {
            FirstName = f;
            LastName  = l;
            Age       = a;
        }
    }

    class FirstNameComparer : IComparer<Person>
    {
        public int Compare(Person x, Person y)
        {
            return x.FirstName.CompareTo(y.FirstName);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Person> people = new List<Person>()
            {
                new Person("Peter" , "Redfield", 32),
                new Person("Marvin", "Monrow"  , 17),
                new Person("Aaron" , "Striver" , 25)
            };

            var orderByQuery8 = people.OrderBy(person => person,
                new FirstNameComparer());

            foreach (var person in orderByQuery8)
            {
                Console.WriteLine(String.Format("{0}", person.FirstName));
            }
        }
    }
}
```

پیاده سازی این رابط مستلزم پیاده سازی متدی به نام Compare() است که عملکرد آن بسیار شبیه به متد CompareTo() در رابط IComparable<T> است، با این تفاوت که این متد، دو شیء را به عنوان آرگومان قبول می‌کند (خط 23). در مثال بالا با استفاده از متد CompareTo() مربوط به کلاس String که یک عدد صحیح را برمی گرداند، خاصیت FirstName اشیاء را با یکدیگر مقایسه کرده‌ایم (خط ۲۵). در انتها می‌توانیم یک نمونه از این کلاس را به عنوان آرگومان به متدهای OrderBy()، OrderByDescending()، ThenBy()، ThenByDescending() ارسال کنیم (خط ۴۱-۴۰):

```
var orderByQuery8 = people.OrderBy(person => person, new FirstNameComparer());
```

آرگومان دوم کلید مرتب سازی را مشخص می‌کند. در متد بالا کلید person است و با کمک شئ FirstNameComparer، متد به صورت خودکار نتایج را بوسیله خاصیت FirstName مرتب می‌کند. راه دیگر برای مرتب سازی نزولی، استفاده از متد Reverse() بر روی مجموعه‌ای است که به صورت صودی مرتب سازی شده است.

```
var orderByQuery9 = (from person in people
                     orderby person.FirstName
                     select person).Reverse();
```

در مثال اول، کل پرس و جو را داخل پرانتز قرار داده و سپس متد Reverse() را فراخوانی کرده‌ایم. مثال دوم شکل متدی پرس و جوی موجود در مثال اول است.

عبارت let

عبارت let به شما امکان می‌دهد که نتیجه یک عبارت را در عبارت پرس و جو ذخیره و در قسمت‌های بعدی پرس و جو از آن استفاده کنید. عبارت let زمانی مفید واقع می‌شود که یک عبارت چندین بار در قسمت‌های مختلف پرس و جو به کار برده شود یا اینکه قصد داشته باشید برای یک خاصیت با نام طولانی یک نام کوتاه‌تر انتخاب کنید.

```
List<Person> persons = new List<Person>
{
    new Person { FirstName="John", LastName="Smith" },
    new Person { FirstName="Henry", LastName="Boon" },
    new Person { FirstName="Jacob", LastName="Lesley" }
};

var query = from p in persons
            let fullName = p.FirstName + " " + p.LastName
            select new { FullName = fullName };

foreach (var person in query)
{
    Console.WriteLine(person.FullName);
}
```

در عبارت پرس و جو بالا، خاصیت‌های FirstName و LastName را با هم ترکیب و نتیجه آن را با استفاده از عبارت let در یک متغیر ذخیره کرده‌ایم. در ادامه‌ی پرس و جو می‌توانید از مقدار داخل این متغیر (fullName) استفاده کنید. در انتهای عبارت پرس و جو در مثال بالا یک

خاصیت جدید به نام FullName را ساخته و مقدار متغیر را به آن نسبت داده‌ایم. عبارت let شکل متدی ندارد. اما می‌توانید با استفاده از متد Select() آن را شبیه سازی کنید:

```
var query = persons.Select(p => new { fullName = p.FirstName + " " + p.LastName })
                   .Select(p => new { FullName = p.fullName });
```

در زیر مثال‌های بیشتری از کاربرد این عبارت را مشاهده می‌کنید:

```
int[] numbers = { 1, 2, 3, 4, 5 };

//Query Expression
var query1 = from n in numbers
             let squared = n * n
             select squared;

//Method Syntax
var query2 = numbers.Select(n => new { squared = n * n })
                    .Select(n => n.squared);
//Query Expression
var query1 = from p in persons
             let isTeenager = p.Age < 20 && p.Age > 12
             where isTeenager
             select new { p.FirstName, p.LastName };

//Method Syntax
var query2 = persons.Select(p => new { isTeenAger = p.Age < 20 && p.Age > 12 })
                    .Where(p => p.isTeenager)
                    .Select(p => new { p.FirstName, p.LastName });
```

در پرس و جوی بالا، مقدار متغیر حاصل از عبارت let را به عنوان آرگومان به متد Where() ارسال کرده‌ایم زیرا نتیجه عبارت، یک مقدار منطقی است.

عبارت group-by

عبارت group-by با استفاده از مقدار یک کلید، آیتم‌ها را گروه بندی می‌کند. این عبارت می‌تواند بین عبارت from و select یا در آخر عبارت پرس و جو قرار گیرد. ساختار ابتدایی یک عبارت group-by به شکل زیر است:

```
group item by key into groupVar
```

عبارت group-by با کلمه کلیدی group و نام آیتمی که قصد گروه بندی آن را داریم شروع می‌شود. سپس بعد از آن، کلمه کلیدی by را که کلید گروه بندی را مشخص می‌کند می‌نویسیم. به عنوان مثال کلید گروه بندی می‌تواند نام شهر باشد تا تمامی آیتم‌هایی که دارای شهر یکسانی هستند در یک گروه قرار گیرند. و در آخر کلمه کلیدی into با نام گروه جدیدی که آیتم در آن قرار می‌گیرد (groupVar) نوشته می‌شود.

groupVar دارای نوع System.Linq.IGrouping<TKey, TElement> می‌باشد که مشخص می‌کند که کلید گروه از نوع TKey و آیتم‌های گروه از نوع TElement است. به عنوان مثال، فرض کنید شما می‌خواهید تعدادی بازیکن را در تیم‌های مختلف قرار دهید. در زیر کلاسی با نام Player (خطوط ۱۰-۶) با دو خاصیت Name و Team تعریف شده است.

```
1  using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;

public class Player
{
    public string Name { get; set; }
   public string Team { get; set; }
}

public class Program
{
    public static void Main()
    {
       List<Player> players = new List<Player>
       {
           new Player { Name = "Johnny", Team= "Red Team" },
           new Player { Name = "Ross", Team = "Blue Team" },
           new Player { Name = "Eric", Team = "Black Team" },
           new Player { Name = "Josh", Team = "White Team" },
           new Player { Name = "Mandy", Team = "Blue Team" },
           new Player { Name = "Flora", Team = "White Team" },
           new Player { Name = "Garry", Team = "Red Team" },
           new Player { Name = "Joseph", Team = "Blue Team"},
           new Player { Name = "Murray", Team = "Black Team"},
           new Player { Name = "Henry", Team = "Black Team"},
           new Player { Name = "Watson", Team = "Red Team"},
           new Player { Name = "Linda", Team = "White Team"}
       };

       var groups = from p in players
                    group p by p.Team into g
                    select new { GroupName = g.Key, Members = g };

       foreach (var g in groups)
       {
           Console.WriteLine("Members of {0}", g.GroupName);

           foreach (var member in g.Members)
           {
               Console.WriteLine("---{0}", member.Name);
           }
       }
    }
}
```

```
Members of Red Team
---Johnny
---Garry
---Watson
Members of Blue Team
---Ross
---Mandy
---Joseph
Members of Black Team
---Eric
---Murray
---Henry
Members of White Team
---Josh
---Flora
---Linda
```

در خطوط ۳۰-۱۶ لیستی از بازیکن‌ها را ایجاد می‌کنیم. قصد داریم که بازیکن‌ها را بر اساس نام تیم گروه بندی کنیم، به صورتی که بازیکن‌های تیم‌های مشابه در یک گروه قرار گیرند. عبارت پرس و جو در خطوط ۳۴-۳۲ از بک عبارت group-by برای گروه بندی استفاده کرده است. بازیکن‌های با مقدار خاصیت Team مشابه در یک گروه که رابط IGrouping<string, Player> را پیاده سازی می‌کند قرار می‌گیرند.

از آنجاییکه ما از تیم که از نوع رشته است به عنوان کلید استفاده کرده‌ایم خاصیت Key هم از نوع رشته خواهد بود. سپس در عبارت select (خط ۳۴) یک نوع بی نام با دو خاصیت GroupName و Members را ساخته و به ترتیب مقدار کلید و اعضای گروه را به آنها نسبت می‌دهیم. خطوط ۴۴-۳۶ با استفاده از حلقه foreach پرس و جو را اجرا می‌کند برای چاپ مقادیر نیاز به دو حلقه foreach داریم. حلقه اول برای چرخش در بین گروه‌های مختلف و چاپ نام آنها و حلقه دوم برای چاپ اعضای گروه‌ها. برای چاپ نام از مقدار خاصیت GroupName و برای چاپ اعضای گروه از خاصیت Members استفاده می‌کنیم.

خاتمه یک عبارت پرس و جو با استفاده از عبارت group-by

عبارت group-by می‌تواند در انتهای عبارت پرس و جو نیز قرار گیرد. به خاطر داشته باشید که یک عبارت پرس و جو می‌تواند با یک عبارت select یا group-by خاتمه پیدا کند. در زیر مثالی را مشاهده می‌کنید که در آن عبارت پرس وجو با یک عبارت group-by خاتمه پیدا می‌کند.

```
var groups = from p in players
             group p by p.Team;

foreach (var g in groups)
{
    Console.WriteLine("Members of {0}", g.Key);

    foreach (var member in g)
    {
        Console.WriteLine("---{0}", member.Name);
    }
}
```

عبارت پرس وجو بالا نتیجه‌ای مشابه مثال قبلی تولید می‌کند. نوع نتیجه پرس و جو IEnumerable<IGrouping<string, Player>> است، به این معنی که شامل مجموعه‌ای از گروه‌ها با کلیدی از نوع رشته و اشیاء Player می‌باشد. در داخل اولین حلقه foreach، ابتدا با استفاده از مقدار Key هر گروه، نام گروه را در خروجی می‌نویسیم. حلقه داخلی برای چاپ اعضای گروه از نام متغیر گروه به عنوان منبع داده استفاده می‌کند. نکته قابل توجه این است که هر متغیر موقت قبل از عبارت group-by، بعد از آن قابل دسترسی نیست.

متد GroupBy()

هر عبارت group-by به یک فراخوانی متد GroupBy() تبدیل می‌شود که یک متد توسعه یافته از رابط IEnumerable<T> است. مثال زیر به شما نشان می‌دهد که چگونه عبارت پرس وجو بالا را با استفاده از شکل متدی بنویسید:

```
var groups = players.GroupBy(p => p.Team);
```

متد GroupBy() یک عبارت لامبدا که دارای یک پارامتر و یک خروجی است را به عنوان پارامتر قبول می‌کند. پارامتر و خروجی عبارت لامبدا به ترتیب بیانگر یک عضو گروه و کلید گروه بندی می‌باشند. عبارت پرس و جو مثال قبلی را می‌توانید به شکل زیر نیز بنویسید:

```
var groups = players.GroupBy(p => p.Team)
        .Select(g => new { GroupName = g.Key, Members = g });
```

اتصال منابع داده ای

بعضی مواقع قصد دارید که منابع داده‌ای مختلف را با هم ترکیب کنید و به عنوان یک نتیجه نشان دهید. البته این منابع داده‌ای باید به طریقی با یکدیگر مرتبط باشند. در LINQ، با استفاده از عبارت join یا متد ()Join می‌توان منابع داده‌ای مختلف را به یکدیگر وصل کرد، البته این منابع داده‌ای باید دارای خواصی باشند که بتوان آنها را از لحاظ مساوی بودن مقایسه کرد. با استفاده از عبارت join یا متد ()Join می‌توان عملیات inner joins، group joins و left outer joins را انجام داد. مفهوم join در LINQ مشابه با join در زبان SQL می‌باشد. اگر شما با مفهوم join در SQL آشنایی دارید، این مفاهیم برای شما آشنا هستند. ممکن است عمل اتصال در LINQ برای افراد تازه کار مشکل باشد، بنابراین بنده تلاش می‌کنم این مفهوم را به شکل بسیار ساده برای شما بیان کنم. در درس بعدی شما یاد می‌گیرید که چگونه عمل اتصال (join) را در LINQ انجام دهید. فرض کنید که یک جدول در بانک اطلاعاتی موجود است که لیست تمامی مؤلف‌ها در آن ذخیره می‌شود.

یک جدول دیگر در بانک اطلاعاتی وجود دارد که مشخصات کتاب‌های مختلف با شماره مؤلفی که آن را تألیف کرده است، در خود ذخیره می‌کند. نتیجه پرس و جویی که باعث شود دو جدول فوق به یکدیگر متصل شود، دارای رکوردهایی است که اطلاعات هر مؤلف و کتابی که تألیف کرده است را در خود دارد.

به عنوان مثال یک رکورد ترکیبی می‌تواند شامل نام مؤلف و نام کتابی که تألیف کره است، باشد. برای اتصال دو منبع داده‌ای مختلف، هر منبع باید دارای خاصیتی باشد که با خاصیت منبع دیگر از لحاظ مساوی بودن مقایسه شود. شرط اصلی برای ترکیب دو رکورد این است که مقدار خاصیت آن دو رکورد با یکدیگر برابر باشد.

در مثال بالا، می‌توان در هر دو جدول (مؤلف و کتاب) فیلدی به نام شماره مؤلف اضافه کرد. هر مؤلف در جدول Auhtors(مؤلف‌ها) می‌تواند دارای یک شماره منحصر به فرد باشد، در حالی که هر کتاب در جدول Books(کتاب‌ها) می‌تواند دارای یک شماره مؤلف باشد، که مشخص می‌کند که چه کسی کتاب را تألیف کرده است. در هر عمل اتصال، یک منبع داده‌ی داخلی و یک منبع داده‌ی خارجی وجود دارد. منبع داده‌ی داخلی شامل رکوردهایی است که با منبع داده‌ی خارجی ترکیب می‌شود.

هر رکورد در منبع داده‌ی داخلی با یک رکورد نظیر خود را در منبع داده‌ی خارجی ترکیب می‌شود و یک رکورد جدید را ایجاد می‌کنند. در درس‌های بعدی ۳ نوع عمل اتصال را بررسی می‌کنیم. inner joins به شما این امکان را می‌دهد که دو منبع داده را به یکدیگر متصل کنید و یک نتیجه مشابه یک جدول را بدست آورید. در طول عمل inner join، رکوردهای داخلی که یک رکورد خارجی نظیر ندارند در نتیجه پرس و جو قرار نمی‌گیرند. inner join ساده‌ترین و آسان‌ترین نوع join است. نوع دیگر عمل join، group join است که یک مجموعه سلسله مراتبی از نتایج را تولید می‌کند. این نوع join، رکوردهایی که با یک رکورد از منبع داده‌ی دیگر ارتباط دارند، در یک گروه قرار می‌دهد. به عنوان مثال تمامی کتاب‌های یک مؤلف خاص را در یک گروه قرار می‌دهد.

نوع دیگر اتصال، left outer join است. این نوع اتصال همانند inner join یک نتیجه همانند جدول تولید می‌کند، با این تفاوت رکوردهای خارجی که معادلی در منبع داده داخلی ندارند بازهم در نتیجه پرس و جو قرار می‌گیرند. در درس‌های بعدی توضیحات بیشتری در رابطه با انواع اتصال داده می‌شود.

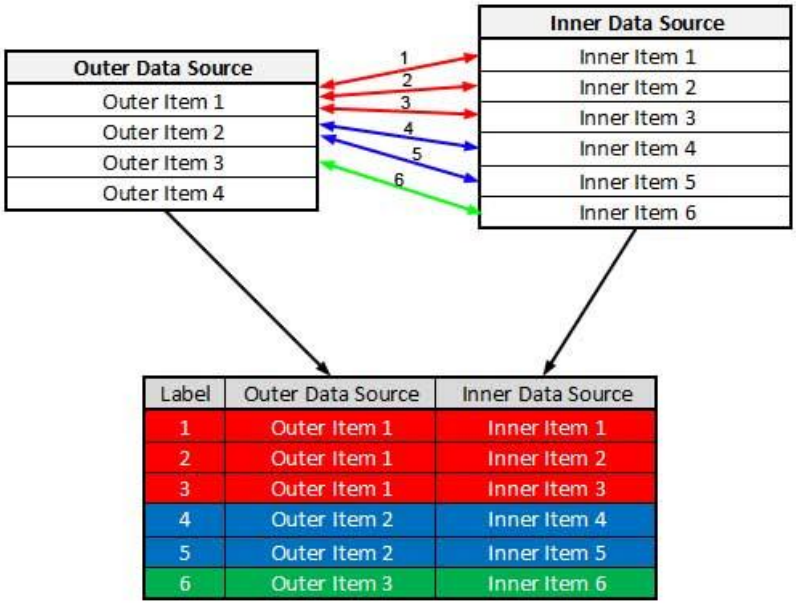
به این نکته توجه کنید که اتصالات را می‌توانید با استفاده از چندین عبارت from انجام دهید اما برای انجام این کار باید به درستی کلاس‌های خود را طراحی کنید. به عنوان مثال کلاس مؤلف می‌تواند یک خاصیت به نام books داشته باشد که مجموعه‌ای از کتاب‌های تألیفی مؤلف را در خود نگه دارد. عبارت Joined می‌تواند به صورت موثری حتی وقتی در بین کلاس‌ها ارتباطی وجود ندارد به کار برده شود. برای این کار کافی است در هر دو کلاس کلیدی را تعریف کنیم که در طول عمل join از آن استفاده کنیم.

عبارت join - انجام عمل inner join

Inner joins ساده‌ترین عمل اتصال است. این نوع اتصال یک خروجی شبیه یک جدول بر می‌گرداند. به این معنی اگر شما به نتیجه پرس و جو نگاه کنید شبیه به یک جدولی خواهد بود که هر سلول از آن یک مقدار دارد. فرض کنید دو جدول در پایگاه داده به نام‌های Authors و Books دارید. جدول Author شامل اطلاعاتی در مورد مؤلف‌های مختلف است. در جدول Books نیز اطلاعاتی در مورد کتاب‌ها همراه با شناسه مؤلف آن کتاب وجود دارد. اتصال دو جدول با استفاده از Inner joins یک خروجی شبیه شکل زیر بر می‌گرداند.

Author	Book
John Smith	Little Blue Riding Hood
John Smith	Snow Black
John Smith	Hanzel and Brittle
Harry Gold	My Rubber Duckie
Harry Gold	He Who Doesn’t Know His Name
Ronald Schwimmer	The Three Little Piggy Banks

هر مؤلف می‌تواند چندین کتاب مربوط به خود داشته باشد. همانطور که در بالا مشاهده می‌کنید مؤلف John Smith سه کتاب، مؤلف Harry Gold دو کتاب و Ronald Schwimmer یک کتاب تألیف کرده‌اند.



هر آیتم در منبع داده‌ی داخلی با آیتم نظیر خود در منبع داده‌ی خارجی ترکیب شده و سپس یک آیتم جدید را تشکیل می‌دهند. اولین آیتم داخلی به دنبال تمامی آیتم‌های نظیر خود در منبع خارجی می‌گردد و با آنها ترکیب می‌شود و تشکیل یک آیتم جدید می‌دهد. برای دومین، سومین و... داخلی هم همین عمل انجام می‌گیرد. برای تشخیص اینکه آیا دو آیتم با هم نظیر هستند، هر دو آیتم باید دارای یک خاصیت یا عضو باشند که از لحاظ مساوی بودن با یکدیگر مقایسه شوند. همانطور که در شکل بالایی می‌بینید آیتم چهارم در منبع داده‌ی خارجی در نتیجه پرس و جو قرار نگرفته است. هر آیتم در منبع داده‌ی خارجی که دارای عضو نظیر در منبع داده‌ی داخلی نباشد، در نتیجه قرار نمی‌گیرد. همین عمل در رابطه با آیتم‌های موجود در منبع داده‌ی داخلی هم صدق می‌کند. به مثال زیر توجه کنید. در این مثال دو کلاس به نام‌های Author و Book تعریف شده‌اند.

```
class Author
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
}

class Book
{
    public int AuthorId { get; set; }
    public string Title { get; set; }
}
```

همانطور که در مثال بالا مشاهده می‌کنید هر دو کلاس دارای خاصیت مشابهی به نام AuthorId هستند. این خاصیت برای تشخیص آیتم‌های نظیر در طی عملیات اتصال استفاده می‌شود. به این نکته توجه کنید که نام خاصیت‌ها لازم نیست با یکدیگر مساوی باشد اما باید هر دو خاصیت دارای نوع داده‌ای یکسانی باشند. در کد زیر اشیایی از هر دو کلاس Author و Books ایجاد شده‌اند. هر کتاب با استفاده از خاصیت AuthorId خود به یک مؤلف اشاره می‌کند.

```
class Program
```

```
{
    public static void Main(string[] args)
    {
        Author[] authors = new Author[]
        {
            new Author() { AuthorId = 1, Name = "John Smith" },
            new Author() { AuthorId = 2, Name = "Harry Gold" },
            new Author() { AuthorId = 3, Name = "Ronald Schwimmer" },
            new Author() { AuthorId = 4, Name = "Jerry Mawler" }
        };

        Book[] books = new Book[]
        {
            new Book() { AuthorId = 1, Title = "Little Blue Riding Hood" },
            new Book() { AuthorId = 3, Title = "The Three Little Piggy Banks" },
            new Book() { AuthorId = 1, Title = "Snow Black" },
            new Book() { AuthorId = 2, Title = "My Rubber Duckie" },
            new Book() { AuthorId = 2, Title = "He Who Doesn't Know His Name" },
            new Book() { AuthorId = 1, Title = "Hanzel and Brittle" }
        };

        var result = from a in authors
                     join b in books on a.AuthorId equals b.AuthorId
                     select new { a.Name, b.Title };

        foreach (var r in result)
        {
            Console.WriteLine("{0} - {1}", r.Name, r.Title);
        }
    }
}
```

```
John Smith - Little Blue Riding Hood
John Smith - Snow Black
John Smith - Hanzel and Brittle
Harry Gold - My Rubber Duckie
Harry Gold - He Who Doesn't Know His Name
Ronald Schwimmer - The Three Little Piggy Banks
```

در خطوط ۲۳-۲۵ از یک عبارت `join` استفاده شده است. عبارت پرس و جو ابتدا یک شیء مؤلف را از منبع داده‌ی خارجی که همان `Authors` است می‌گیرد، سپس در عبارت `join` یک شیء کتاب را از منبع داده‌ی داخلی که همان `Books` است انتخاب می‌کند در نهایت مقدار خاصیت `AuthorId` از هر دو شیء انتخاب شده را از لحاظ یکسان بودن با هم مقایسه می‌کند.

به این نکته توجه کنید که از کلمه کلیدی `equal` به جای عملگر == استفاده شده است. عبارت `join` فقط از لحاظ تساوی دو شیء را مقایسه می‌کند. شما نمی‌تواند از عملگرهایی مانند > یا < برای مقایسه کلیدها استفاده کنید. بنابراین مایکروسافت برای یادآوری این نکته کلمه کلیدی `equals` را به وجود آورده است.

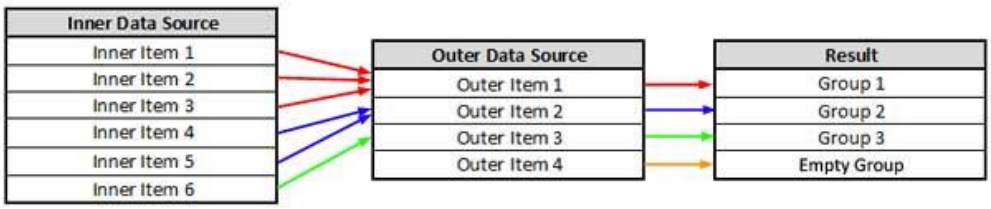
اگر خاصیت `AuthoId` کتاب با `AuthorId` مؤلف برابر باشد، می‌توانید با استفاده از عبارت `select` نام مؤلف و عنوان کتاب را ترکیب کنید و در نتیجه پرس و جو قرار دهید. هر کتاب با تمامی مؤلف‌ها مقایسه می‌شود و با آنهایی که نظیر باشد (تعلق داشته باشد) در نتیجه پرس و جو قرار می‌گیرد. به این نکته توجه کنید که مؤلفی با نام `Jerry Mawler` در نتیجه پرس وجو قرار نمی‌گیرد زیرا دارای کتابی در منبع داده‌ی داخلی وجود ندارد که با آن ترکیب شود. همچنین هر کتابی که آیتم نظیری نداشته باشد، در نتیجه پرس وجو قرار نمی‌گیرد. عبارت `join` مانند سایر عبارات در طی عملیات کامپایل به شکل متدی خود تبدیل می‌شود. عبارت پرس و جوی موجود در مثال بالا را می‌تواند به شکل زیر نوشت:

```
var result = authors.Join(books,
                author => author.AuthorId,
                book => book.AuthorId,
                (author, book) => new { author.Name, book.Title }
                );
```

این نسخه از متد Join() نقش یک inner join را بازی می‌کند. این متد چهار پارامتر قبول می‌کند. اولین پارامتر منبع داده داخلی که در اینجا Books است، می‌باشد. دومین پارامتر یک عبارت لامبدا که شامل همه آیتم‌های منبع داده حارجی همراه با کلیدی است که هر آیتم از منبع داده خارجی بر اساس آن با آیتم نظیر خود در منبع داده داخلی ترکیب می‌شود (در این مثال AuthorId به عنوان کلید به کار رفته است). سومین پارامتر هم یک عبارت لامبدا که شامل همه آیتم‌های منبع داده داخلی همراه با کلیدی که با کلید منبع داده خارجی مقایسه می‌شود، می‌باشد. و در نهایت چهارمین پارامتر، که عبارت لامبدایی با دو پارامتر است، یکی آیتم خارجی و دیگری آیتم داخلی که دارای کلیدهای یکسانی هستند و نتیجه نهایی از ترکیب این دو آیتم می‌باشد.

عبارت Join - انجام یک عمل Group Join

با استفاده از عبارت join شما می‌توانید یک اتصال group join را انجام دهید. یک اتصال group join عناصری از منبع داده‌ی داخلی را با استفاده از عنصر نظیر آنها در منبع داده‌ی خارجی را گروه بندی می‌کند. به عنوان مثال، همه‌ی کتاب‌های نوشته شده به وسیله‌ی Smith John گروه بندی می‌شوند، همچنین تمامی کتاب‌های نوشته شده توسط Gold Harry در یک گروه جداگانه قرار می‌گیرند. شکل زیر یک عمل group join نشان می‌دهد.



همه‌ی عناصر موجود در منبع داده‌ی داخلی که دارای کلیدی مشترک هستند و دارای یک عنصر نظیر در منبع داده‌ی خارجی هستند تشکیل یک گروه را می‌دهند. همانطور که مشاهده می‌کنید، نتیجه‌ی یک اتصال group join کلکسیونی از گروه‌ها هست. هرکدام بیانگر گروهی برای یک کلید خاص می‌باشند.

به عنوان مثال، کلید اتصال می‌تواند مؤلف کتاب باشد. می‌توان کتاب‌های هر مؤلف را در گروهی جداگانه قرار داد. در این صورت نتیجه عمل پرس و جو کلکسیونی از گروه‌ها است که هر گروه شامل کتاب‌های نوشته شده توسط یک مؤلف خاص می‌باشد. هر عنصر خارجی (عنصری که در منبع داده‌ی خارجی است) که عنصری نظیر در منبع داده‌ی داخلی ندارد تشکیل یک گروه خالی را می‌دهد و همچنان در لیست نتایج ظاهر می‌شود. به مثال زیر دقت کنید. دو کلاس به نام‌های Author و Book تعریف می‌کنیم.

```
class Author
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
}
```

```
class Book
{
    public int AuthorId { get; set; }
    public string Title { get; set; }
}
```

عبارت پرس و جوی موجود در مثال زیر از اتصال group join استفاده می‌کند.

```
1  Author[] authors = new Author[]
2  {
3      new Author() { AuthorId = 1, Name = "John Smith" },
4      new Author() { AuthorId = 2, Name = "Harry Gold" },
5      new Author() { AuthorId = 3, Name = "Ronald Schwimmer" },
6      new Author() { AuthorId = 4, Name = "Jerry Mawler" }
7  };
8
9  Book[] books = new Book[]
10 {
11     new Book() { AuthorId = 1, Title = "Little Blue Riding Hood" },
12     new Book() { AuthorId = 3, Title = "The Three Little Piggy Banks" },
13     new Book() { AuthorId = 1, Title = "Snow Black" },
14     new Book() { AuthorId = 2, Title = "My Rubber Duckie" },
15     new Book() { AuthorId = 2, Title = "He Who Doesn't Know His Name" },
16     new Book() { AuthorId = 1, Title = "Hanzel and Brittle" }
17 };
18
19 var result = from a in authors
20              join b in books on a.AuthorId equals b.AuthorId into booksByAuthor
21              select new { Author = a.Name, Books = booksByAuthor };
22
23 foreach (var r in result)
24 {
25     Console.WriteLine("Books written by {0}:", r.Author);
26
27     foreach (var b in r.Books)
28     {
29         Console.WriteLine("---{0}", b.Title);
30     }
31 }
```

```
Books written by John Smith:
---Little Blue Riding Hood
---Snow Black
---Hanzel and Brittle
Books written by Harry Gold:
---My Rubber Duckie
---He Who Doesn't Know His Name
Books written by Ronald Schwimmer:
---The Three Little Piggy Banks
Books written by Jerry Mawler:
```

به عبارت join در خط ۲۰ توجه کنید. این عبارت یک کتاب از منبع داده‌ی books را به یک مؤلف در منبع داده‌ی authors به وسیله‌ی خاصیت AuthorId متصل می‌کند. کلمه کلیدی into که بعد از آن متغیر گروه بندی قرار می‌گیرد مشخص می‌کند که عمل اتصال از نوع group join می‌باشد. کلید تمامی عنصرهای داخلی (عناصری که در منبع داده‌ی داخلی قرار دارند) که با کلید عناصر متناظر خارجی خود برابر باشند با یکدیگر ترکیب می‌شوند و تشکیل یک گروه جدید می‌دهند. عبارت select در خط ۲۱ نتیجه را که شامل نام مؤلف و گروه کتاب‌های متعلق به آن است را در خروجی قرار می‌دهد. نتیجه‌ی عبارت پرس و جو کلکسیونی از گروه‌های کتاب است. حلقه‌ی foreach تو در تو در خطوط ۲۳ تا ۳۱ نتایج عبارت

پرس و جو را نشان می‌دهد. در داخل اولین حلقه foreach نام مؤلف نمایش داده می‌شود. در داخل حلقه داخلی تمامی کتاب‌های نوشته شده توسط مؤلف که در خاصیت Books قرار دارند، نمایش داده می‌شود.

به یاد داشته باشید که این خاصیت، کلکسیونی از کتاب‌ها را که به تفکیک مؤلف گروه بندی شده‌اند را در خود قرار می‌دهد. همانطور که در خروجی مشاهده می‌کنید Jerry Mawler هیچ کتابی ننوشته است، در نتیجه، خاصیت Books خالی است. بنابراین هیچ کتابی نمایش داده نمی‌شود.

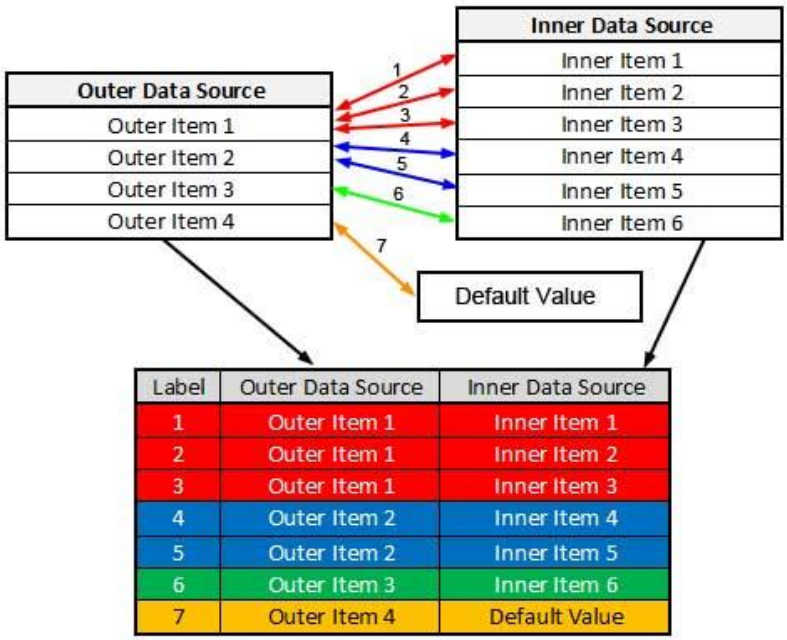
متد GroupJoin() معادل متدی عبارت group join می‌باشد. معادل متدی مثال بالا به صورت زیر می‌باشد:

```
var result = authors.GroupJoin(books,
                               author => author.AuthorId,
                               book => book.AuthorId,
                               (author, booksByAuthor) =>
                                   new { Author = author.Name, Books = booksByAuthor });
```

اولین پارامتر مشخص کننده منبع داده‌ی داخلی است که ما قصد داریم آن را منبع داده‌ی خارجی متصل کنیم. پارامتر دوم یک نماینده است که یک عبارت لامبدا را برای تشخیص کلید خارجی برای عمل الحاق، قبول می‌کند. سومین پارامتر کلید داخلی و پارامتر چهارم برای ایجاد شکل خروجی برای هر گروه می‌باشد.

عبارت Join - انجام یک عمل Left Outer Join

با استفاده از عبارت join شما می‌توانید یک اتصال از نوع left outer join انجام دهید. همانند عمل inner join این نوع اتصال نیز یک خروجی مسطح بر می‌گرداند. عمل inner join عنصری که معادلی در منبع داده‌ی دیگر نداشته باشد را حذف می‌کند. به عنوان مثال اگر یک مؤلف کتابی ننوشته باشد، در نتیجه‌ی عبارت پرس و جو نمایش داده نمی‌شود. در عمل left outer join عناصری که در منبع داده‌ی دیگر معادلی نداشته باشند باز هم در نتیجه پرس و جو قرار می‌می گیرند. این عمل با استفاده از متد DefaultIfEmpty() امکان پذیر است.



در زیر مثالی را مشاهده می‌کنید که از عمل left outer join استفاده می‌کند.

```
Author[] authors = new Author[]
{
    new Author() { AuthorId = 1, Name = "John Smith" },
    new Author() { AuthorId = 2, Name = "Harry Gold" },
    new Author() { AuthorId = 3, Name = "Ronald Schwimmer" },
    new Author() { AuthorId = 4, Name = "Jerry Mawler" }
};

 Book[] books = new Book[]
{
     new Book() { AuthorId = 1, Title = "Little Blue Riding Hood" },
     new Book() { AuthorId = 3, Title = "The Three Little Piggy Banks" },
     new Book() { AuthorId = 1, Title = "Snow Black" },
     new Book() { AuthorId = 2, Title = "My Rubber Duckie" },
     new Book() { AuthorId = 2, Title = "He Who Doesn't Know His Name" },
     new Book() { AuthorId = 1, Title = "Hanzel and Brittle" }
 };

 var result = from a in authors
              join b in books on a.AuthorId equals b.AuthorId into booksByAuthors
              from x in booksByAuthors.DefaultIfEmpty(new Book {AuthorId=0,Title="None"})
              select new { Author = a.Name, x.Title };

 Console.WriteLine("{0, -20} {1}", "Author", "Book");
 foreach (var r in result)
 {
     Console.WriteLine("{0, -20} {1}", r.Author, r.Title);
 }
```

برای انجام یک عمل left outer join با استفاده از عبارت join، ابتدا لازم است که دو منبع داده را با استفاده از group join متصل نمایید. سپس بر روی نتیجه‌ی حاصل یک پرس و جوی دیگر را انجام دهید. با استفاده از متد DefaultIfEmpty() برای گروهی که عنصری ندارد، یک مقدار پیش‌فرض مشخص می‌کنیم. دو خط ابتدای عبارت پرس و جوی موجود در خطوط ۱۹ تا ۲۳ با استفاده از عمل group join تمامی مؤلف‌ها و کتاب‌هایی که خاصیت AuthorId آنها دارای مقدار برابری است را با هم ترکیب می‌کند. خط بعدی پرس و جوی دیگری برای روی نتیجه‌ی اولین پرس و جو به عنوان منبع داده انجام می‌دهد (عمل ترکیب را انجام می‌دهد). با استفاده از DefaultIfEmpty() یک مقدار پیش‌فرض را برای گروهی که هیچ عنصری ندارد مشخص می‌کنیم. متد DefaultIfEmpty() یک آرگومان که نوعی برابر هر کدام از اعضای گروه را دارد می‌پذیرد. از آنجایی که هر گروه در عبارت پرس و جوی ما دارای نوعی برابر Book می‌باشد بنابراین یک نمونه از کلاس Book را ساخته و به خاصیت‌های آن مقدار پیشفرضی را می‌دهیم.

بر اساس منبع داده‌های موجودر Jerry Mawler با شماره برابر ۴ هیچ کتابی در منبع داده‌ی books ندارد. اگر از عمل inner join استفاده کنید این مؤلف در نتیجه پرس و جو قرار نمی‌گیرد. و اما از آنجاییکه ما از عمل left outer join استفاده کرده‌ایم این مؤلف در نتیجه‌ی پرس و جو قرار می‌گیرد. و مقدار پیشفرضی را که تعیین نموده‌اید، به عنوان کتاب او نمایش داده می‌شود.

LINQ to XML

دات نت چندین تکنیک برای دسترسی و دستکاری یک فایل XML در اختیار ما می‌گذارد. در درس‌های قبلی از XmlReader یا دیگر کلاس‌های DOM XML مانند XmlDocument، XmlElement، و XmlComment استفاده کردید. همچنین با استفاده از زبان‌های پرس و جوی XML از جمله

XPath و XQuery توانستید عناصر خاصی از سند XML را انتخاب نمایید. حال مشکل اینجاست که شما با زبان‌هایی سر و کار دارید که هیچ چیزی برای کار با سی‌شارپ در اختیار شما قرار نمی‌دهند. اینجاست که LINQ to XML وارد عمل می‌شود. با استفاده از LINQ to XML شما می‌توانید از عملگرهای LINQ برای دستکاری و ایجاد اسناد XML استفاده کنید. اجازه دهید که یک مثال ساده از LINQ To XML را بررسی کنیم. یک سند XML با مقادیر زیر ایجاد کنید.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folley">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
  <Person name="Jerry Frost">
    <Age>27</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Adam Wong">
    <Age>35</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

ویژوال استودیو را باز کرده و یک برنامه کنسول ایجاد و نام آن را LinqToXmlPractice بگذارید. بر روی Solution Explorer کلیک راست کرده و از منوی باز شده گزینه New item را انتخاب کنید. از لیست ظاهر شده گزینه Xml file را انتخاب و نام آن را به sample.xml تغییر دهید. محتویات فایل ایجاد شده را پاک و مقادیر بالا را در آن کپی کنید.

فایل sample.xml شامل عناصر XML ی است که می‌خواهیم با آنها کار کنیم. فایل XML شامل یک عنصر ریشه به نام Persons است. این عنصر به نوبه خود دارای چندین عنصر فرزند به نام Person می‌باشد. هر عنصر Person دارای خواصی مانند نام (name)، سن (Age) و جنسیت (Gender) می‌باشد. الان می‌خواهیم با استفاده از LINQ نام هر شخص به دست بیاوریم. به محیط کدنویسی رفته و فضای نام زیر را به بخش فضاهای نامی اضافه کنید:

```
using System.Xml.Linq;
```

در داخل متد Main() هم کد زیر را بنویسید:

```
XDocument doc = XDocument.Load(@"..\..\sample.xml");

var names = from d in doc.Root.Elements("Person")
            select d.Attribute("name").Value;

Console.WriteLine("Names of every person from the XML file.");

foreach (var name in names)
{
```

```
    Console.WriteLine("{0}", name);
}
```

```
Names of every person from the XML file.
John Smith
Mike Folley
Lisa Carter
Jerry Frost
Adam Wong
```

توضیح کامل کدها در درس‌های قبلی آمده است. مشاهده می‌کنید که با استفاده از LINQ to XML چقدر راحت می‌توانید با اسناد XML کار کنید.

ایجاد یک سند XML با استفاده از LINQ to XML

اگر با کلاس‌های XML DOM برای ایجاد اسناد XML کار کرده باشید به شما پیشنهاد می‌کنیم از کلاس‌هایی که در فضای نامی System.Xml.Linq قرار دارند، استفاده کنید که کار با عناصر XML را بسیار راحت تر می‌کنند. ابتدایی‌ترین روش برای بارگذاری یک سند XML استفاده از کلاس XmlDocument می‌باشد که در فضای نامی System.Xml قرار دارد:

```
XmlDocument doc = XmlDocument.Load("myXmlFile.xml");
```

به جای کلاس بالا می‌توان از کلاس جدیدتری با نام XDocument نیز استفاده کرد:

```
XDocument doc = XDocument.Load("myXmlFile.xml");
```

این کلاس جدید که دارای نام کوتاه‌تری نسبت به نسخه قدیمی تر است چه کاری انجام می‌دهد؟ جواب ساختار وظیفه‌ای است. با ساختار وظیفه‌ای لازم نیست که عناصر XML DOM را یک به یک تعریف کنید. شما می‌توانید با استفاده از سربارگذاری‌های کلاس XDocument هر کاری که دوست دارید انجام دهید. برای روشن شدن موضوع به مثال زیر توجه کنید. با استفاده از کلاس‌های قدیمی XML DOM موجود در فضای نام System.Xml یک فایل XML ساده ایجاد می‌کنیم:

```
//A new XML Document
XmlDocument doc = new XmlDocument();

//Xml Declaration
XmlDeclaration declaration = doc.CreateXmlDeclaration("1.0", "utf-8", "yes");
//Attach declaration to the document
doc.AppendChild(declaration);

//Create a comment
XmlComment comment = doc.CreateComment("This is a comment");
//Attach comment to the document
doc.AppendChild(comment);

//Create root element
XmlElement root = doc.CreateElement("Persons");
//Attach the root node to the document
doc.AppendChild(root);

//Create a Person child element
XmlElement person1 = doc.CreateElement("Person");
//Add an attribute name with value John Smith
person1.SetAttribute("name", "John Smith");
//Crate Age element
XmlElement person1Age = doc.CreateElement("Age");
```

```
person1Age.InnerText = "30";
//Create Gender element
XmlElement person1Gender = doc.CreateElement("Gender");
person1Gender.InnerText = "Male";

//Attach Age and Gender element to the Person element
person1.AppendChild(person1Age);
person1.AppendChild(person1Gender);

//Attach Person child element to the root Persons element
doc.DocumentElement.AppendChild(person1);

//Create another Person child element
XmlElement person2 = doc.CreateElement("Person");
//Add attribute name with value Mike Folley
person2.SetAttribute("name", "Mike Folley");
//Crate Age element
XmlElement person2Age = doc.CreateElement("Age");
person2Age.InnerText = "25";
//Create Gender element
XmlElement person2Gender = doc.CreateElement("Gender");
person2Gender.InnerText = "Male";

//Attach Age and Gender element to the Person element
person2.AppendChild(person2Age);
person2.AppendChild(person2Gender);

//Attach second Person child element to the root Persons element
doc.DocumentElement.AppendChild(person2);

//Save the constructed XML into an XML file
doc.Save(@"C:\sample1.xml");
```

کد بالا، فایل XML زیر را به وجود می‌آورد:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--This is a comment-->
<Persons>
   <Person name="John Smith">
      <Age>30</Age>
      <Gender>Male</Gender>
   </Person>
   <Person name="Mike Folley">
      <Age>25</Age>
      <Gender>Male</Gender>
   </Person>
</Persons>
```

حال همین فایل XML را با استفاده از کلاس‌های جدید LINQ to XML ایجاد می‌کنیم:

```
XDocument doc = new XDocument(
    new XDeclaration("1.0", "utf-8", "yes"),
    new XComment("This is a comment"),
    new XElement("Persons",
        new XElement("Person", new XAttribute("name", "John Smith"),
            new XElement("Age", new XText("30")),
            new XElement("Gender", new XText("Male"))),
        new XElement("Person", new XAttribute("name", "Mike Folley"),
            new XElement("Age", new XText("25")),
            new XElement("Gender", new XText("Male")))));

doc.Save(@"C:\sample2.xml");
```

همانطور که مشاهده می‌کنید، هنگام استفاده از کلاس‌های DOM XML برای ایجاد یک سند XML لازم است که شما تک تک عناصر را تعریف کنید. اما با استفاده از کلاس‌های LINQ to XML هر قسمت از سند را در داخل سازنده هر کلاس ایجاد می‌کنید. این روش ایجاد سند XML به ساختار وظیفه‌ای معروف است. هر سربارگذاری کلاس LINQ to XML آرگومانهایی قبول می‌کند که همان عناصر فرزند یا مقادیری می‌باشد، که قرار است به عنصر اختصاص داده شوند. به عنوان مثال یکی از سربارگذاری‌های سازنده کلاس XElement به صورت زیراست:

```
XElement(XName name, params object[] content)
```

اولین آرگومان نام عنصر است که در قالب یک رشته به سازنده ارسال می‌شود و به صورت خودکار به یک نمونه XName تبدیل می‌شود. دومین پارامتر هم کمی خاص است چون به شما اجازه می‌دهد که هر تعداد از انواع مختلف شئ مانند XText، XAttribute و یا XElement را به عنصر فرزند از عنصر جاری XElement ارسال کنید. کلاس دیگر LINQ to XML فقط یک آرگومان مانند XComment و XText قبول می‌کند که هر دو یک آرگومان رشته‌ای که نماینده یک متن است را، تحویل می‌دهند. در جدول زیر برخی از کلاس‌های LINQ to XML که هر کدام برای ساخت بخش‌های مختلف یک سند به کار می‌روند، ذکر شده‌اند:

کلاس	توضیح
XDocument	نماینده یک سند XML
XDeclaration	نماینده یک تعریف در XML
XElement	نماینده یک عنصر در XML
XAttribute	نماینده صفت یک عنصر در XML
XComment	نماینده یک توضیح
XText	نماینده متن داخلی یک عنصر XML

به عنوان مثال اگر بخواهید یک عنصر با نام Person و صفت و مقدار John Smith ایجاد کنید می‌توانید از کد زیر استفاده نمایید:

```
var personElement = new XElement("Person", new XAttribute("name", "John Smith"));
```

برای اضافه کردن یک عنصر فرزند به عنصر فوق هم می‌توان به راحتی یک آرگومان به سازنده XElement اضافه کرد:

```
var personElement = new XElement("Person", new XAttribute("name", "John Smith"),
                        new XElement("Age", new XText("30")));
```

عنصر فرزند دارای خصوصیت Age (سن) است که با استفاده از آرگومان بعدی در سازنده، متن داخلی آن را با استفاده از کلاس XText تعیین می‌کنیم. در مثال ۲ مشاهده می‌کنید که هر آرگومان XElement دارای تورفتگی می‌باشد که شبیه به تو رفتگی عناصر XML می‌باشد. در آخر مثال ۲ متد XDocument.Save() را فراخوانی کرده‌ایم که به شما اجازه می‌دهد، ساختار XML را در حافظه ذخیره کنید. این متد یک رشته را به عنوان

آرگومان قبول می‌کند که همان مسیر فایل می‌باشد. اگر فایل وجود نداشته باشد آن را ایجاد و اگر وجود داشته باشد، آنرا بازنویسی (overwritte) می‌کند. این متد دارای سربارگذاری‌های دیگری هم می‌باشد، که ما از یکی از آنها برای ذخیره فایل XML استفاده کردیم.

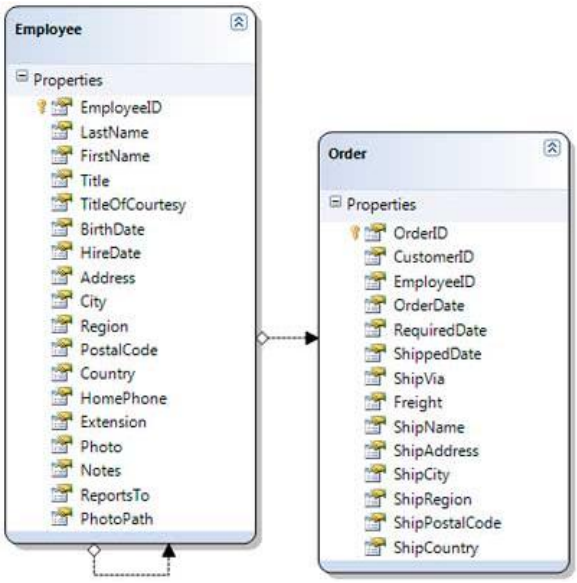
LINQ To SQL چیست؟

LINQ to SQL ابزاری قدرتمند است که به برنامه نویسان اجازه دسترسی به دیتابیس‌ها را مانند اشیاء در سی‌شارپ می‌دهد. با استفاده از LINQ to SQL قادر خواهید بود به جای یادگیری SQL از عملگرهای پرس جوی LINQ و متدهای آن استفاده کنید. LINQ to SQL دارای یک API برای ایجاد و دستکاری دیتابیس می‌باشد. پرس و جوهای LINQ و فراخوانی‌های متدهای API به دستورات SQL تبدیل می‌شوند، که این دستورات برای ایجاد تغییرات یا دریافت پرس و جوها از بانک اطلاعاتی، اجرا می‌شوند. LINQ to SQL یکی از روش‌های بسیار پیشرفته برای دسترسی به بانک اطلاعاتی توسط سی‌شارپ و دات‌نت می‌باشد.

به این نکته توجه کنید که LINQ to SQL فقط زمانی قابل اجرا است که شما از SQL Server به عنوان دیتابیس استفاده می‌کنید. دیتابیسی که قرار است در این آموزش مورد استفاده قرار بگیرد دیتابیس Northwind می‌باشد و می‌توانید آن را از لینک زیر دانلود کنید:

```
http://www.w3-farsi.com/?p=15664
```

اگر نسخه آزاد Visual C# Express را در اختیار دارید لازم است به یک فایل دیتابیس با پسوند mdf. دسترسی داشته باشید. دیتابیس Northwindدر صورتی که درست نصب شده باشد در مسیر C:\SQL Server 2000 Sample Databases قرار دارد. برای قابل مشاهده کردن پسوند فایل دیتابیس به Control Panel رفته و از گزینه Folder Options بر روی سربرگ دوم آن یعنی View کلیک و تیک Hide" extensions for known file types" را برداشته و دکمه OK را کلیک نمایید. فایل‌های دیتابیسی که در SQL ساخته می‌شوند از جمله Northwind دارای پسوند mdf. هستند. SQL Server Express 2008 یک پوشه پیش‌فرض برای ذخیره فایل‌های دیتابیس دارد که در مسیر C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data می‌باشد. از آنجاییکه یک نسخه از دیتابیس Northwind.mdf در این مسیر قرار دارد، نیازی به ایجاد آن نیست. ویژوال استودیو دارای ابزارهای قدرتمندی برای تولید کلاس‌های LINQ to SQL با استفاده از Object Relational Designer می‌باشد. شما می‌توانید به راحتی با ماوس جداول را در این محیط کشیده و رها کنید (drag and drop) تا ویژوال استودیو به طور خودکار کلاس‌های لازم برای مرتبط کردن جداول و سطرهای هر جدول خاص و دیتابیس را تولید کند. مشاهده می‌کنید که جداول از تعدادی ستون یا فیلد تشکیل شده است، این ستون‌ها شامل محتوای جدول مورد نظر می‌باشد. در این محیط فلش‌هایی هم دیده می‌شوند که ارتباط بین جداول را نشان می‌دهند. به شکل زیر توجه کنید:



کلاس ایجاد شده سطرهای هر جدول را به شما نشان می‌دهد. به عنوان مثال ما یک جدول به نام Employees داریم. ویژوال استودیو به طور خودکار نام جدول را به شکل مفرد در می‌آورد و یک شئ با نام Employee ایجاد می‌کند که نماینده هر سطر یا رکورد جدول مذکور می‌باشد. یک کلاس متناظر از نوع Table<TEntity> واقع در فضای نامی System.Data.Linq برای هر جدول موجود در محیط LINQ to SQL ایجاد می‌شود. کار TEntity ساخت یکی شیء نظیر به نظیر می‌باشد، وقتی که دو جدول مثل هم در یک کلاس داشته باشید. به عنوان مثال جدول Employeesدارای کلاس متناظر Table<Employee> می‌باشد. برای جلوگیری از خطا در ستون‌ها و رکوردهای موجود، این شیء یک نمونه از جدول مورد نظر را می‌سازد.

Table<TEntity> رابط IQueryable<TEntity> از فضای نام System.Linq را پیاده سازی می‌کند. وقتی یک عملیات پرس و جوی LINQ یک شیء از این رابط را پیاده سازی می‌کند و نتایجی از یک دیتابیس را بدست می‌آورد، نتایج به صورت خودکار در کلاس‌های LINQ to SQL ذخیره می‌شوند. برای اینکه بتوانید از کلید داخلی و خارجی در این محیط استفاده کنید ابتدا در محیط ویژوال استدیو باید ارتباط بین دو جدول را از طریق فیلدهای مشترک ایجاد کنید برای ایجاد فیلدهای مشترک بهتر است ارتباط از طریق کلید اصلی هر دو جدول انجام شود. با ارتباط دادن دو جدول از طریق فیلدهای مشترک می‌توانید از طریق کلید خارجی به محتویات جدول ارتباط داده شده دسترسی پیدا کنید.

به عنوان مثال، دو جدول با نام‌های Employees و Companies هر دو فیلدی به نام CompanyID دارند. CompanyID کلید اصلی جدول Companies بوده ولی در جدول Employees یک کلید خارجی است که به CompanyID در جدول Companies اشاره دارد. وقتی که ویژوال استدیو جداول کلاس را می‌سازد ارتباط آنها را نیز از طریق کلید خارجی مورد بررسی قرار می‌دهد، کلاسEmployee از جدول Employee دارای یک ارتباط با جدول Order می‌باشد، به همین دلیل با فلش به هم مرتبط هستند، زمانیکه به اطلاعات جدول Employee نیاز داشته باشیم از طریق ستون مشابه (کلید خارجی) به اطلاعات جدولEmployee دسترسی خواهیم داشت.

کلاس DataContext در LINQ to SQL که از System.Data.Linq.DataContext ارث بری می‌کند، ایجاد می‌کند. این کلاس مسئول اتصال برنامه و دیتابیس است. به ازای هرگونه تغییر در جداول، این تغییر در کلاس LINQ to SQL نیز باید اعمال گردد این تغییرات باید به صورتی ایجاد شود که جداول تغییر یافته جایگزین جداول قبلی شوند.

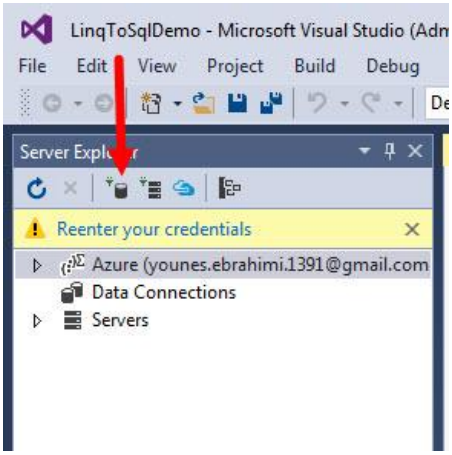
ویژوال استودیو به طور خودکار یک DataContext در قالب DataContext <Database> ایجاد می‌کند که در آن <Database> نام دیتابیس می‌باشد. به عنوان مثال هنگام استفاده از دیتابیس Northwind یک NorthwindDataContext با خواص مربوط به هر جدول ایجاد می‌شود. این خواص شامل مجموعه‌ای از اشیاء که نماینده سطرهای هر جدول می‌باشند، هستند. به عنوان مثال کلاس NorthwindDataContext یک خاصیت Employees که مربوط به جدول Employees است، دارد. این خاصیت مجموعه‌ای از اشیاء می‌باشد که نماینده سطرهای جدول می‌باشد. در درس بعد با یک مثال نحوه ارتباط با دیتابیس را با استفاده از تکنولوژی LINQ to SQL را آموزش می‌دهیم.

پرس و جو در دیتابیس با استفاده از LINQ to SQL

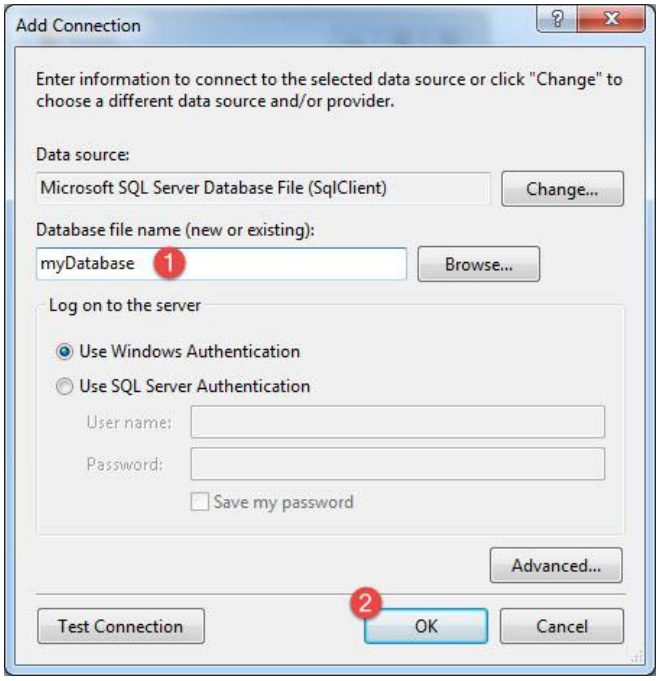
می‌خواهیم یک برنامه ویندوزی ایجاد کنیم که به شما اجازه پرس و جوی یک رکورد از یک جدول خاص با استفاده از کلاس‌های LINQ to SQL را می‌دهد. شما می‌آموزید که چگونه با استفاده از محیط Object Relational Designer کلاس‌های LINQ to SQL را تولید و از آنها در کدنویسی استفاده کنید .

ایجاد دیتابیس و جدول

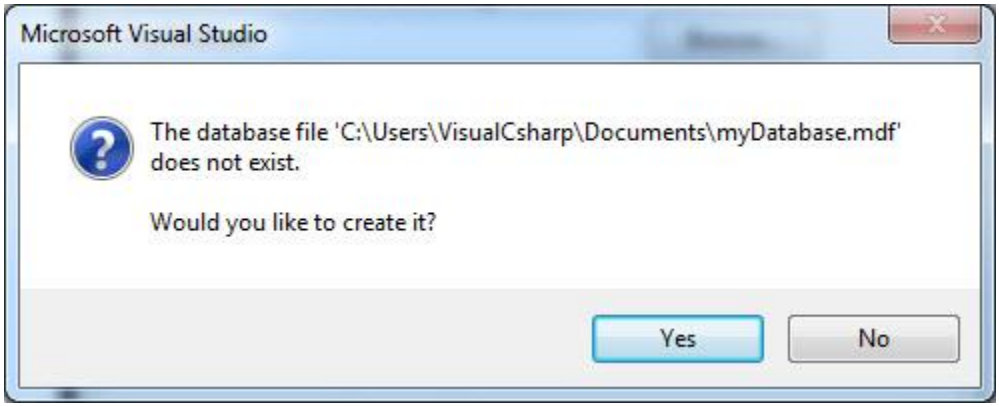
قبل از شروع این آموزش، ابتدا یک دیتابیس با یک جدول ایجاد کرده و سپس چندین داده را در داخل جدول وارد می‌کنیم. برای این منظور ابتدا پنجره‌ی Server Explorer را ظاهر می‌کنیم. برای نمایش این پنجره به مسیر Views > Server Explorer بروید. بعد از باز شدن این پنجره بر روی آیکن Connect to database کلیک کنید:



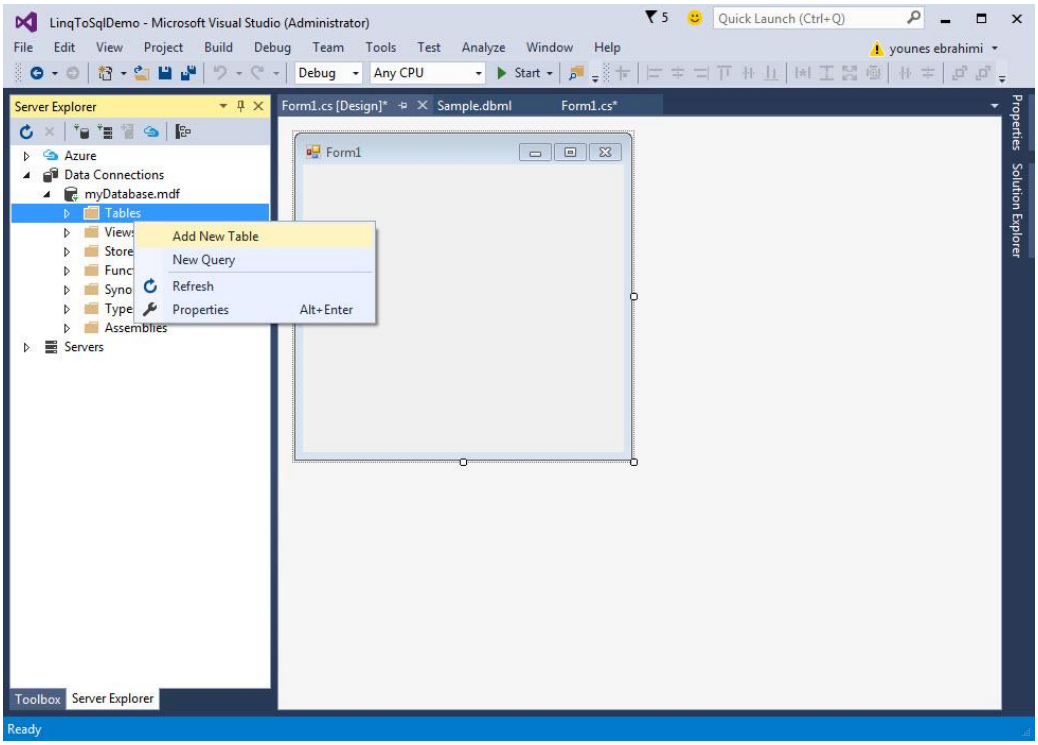
سپس پنجره ای به صورت زیر باز می‌شود که از شما می‌خواهد نام یا مسیر یک دیتابیس را وارد کنید. چون قرار است یک دیتابیس جدید ایجاد کنیم، یک نام برای دیتابیس انتخاب کرده و سپس بر روی دکمه OK کلیک می‌کنیم:



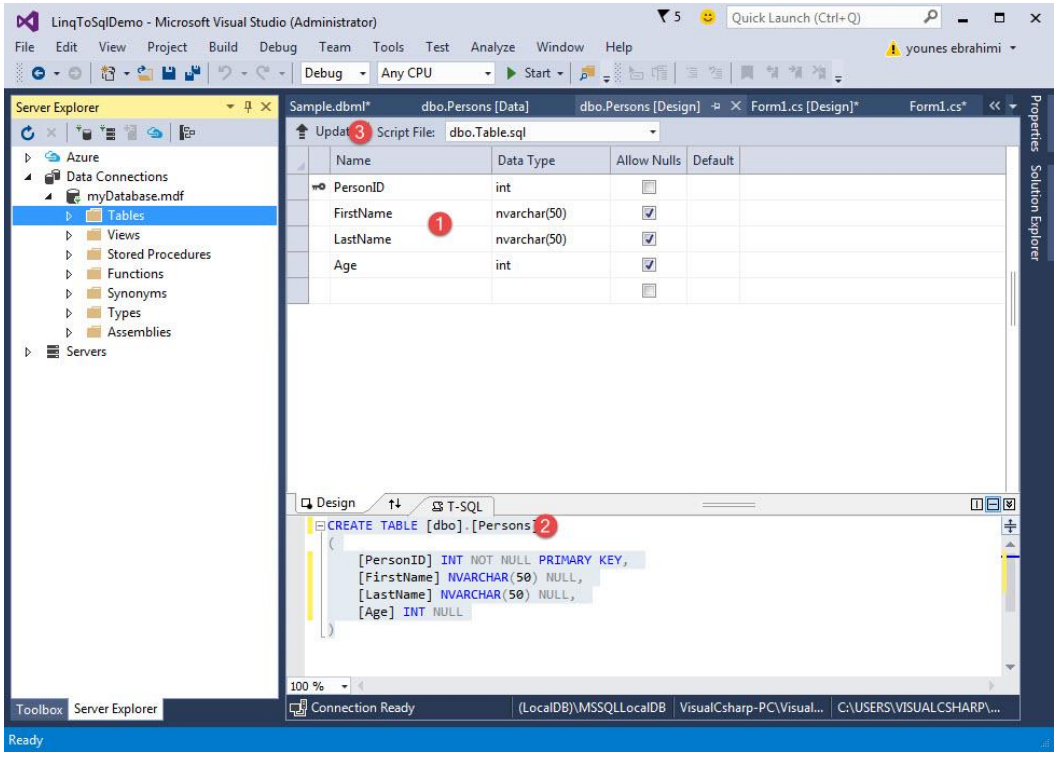
با کلیک بر روی دکمه `OK` پیغامی به صورت زیر به نمایش در می‌آید که به شما می‌گوید که این دیتابیس از قبل وجود ندارد، آیا می‌خواهید آن را ایجاد کنید؟ که با زدن بر روی دکمه `Yes` دیتابیس ما که در اینجا نام آن `sample` است ایجاد می‌شود:



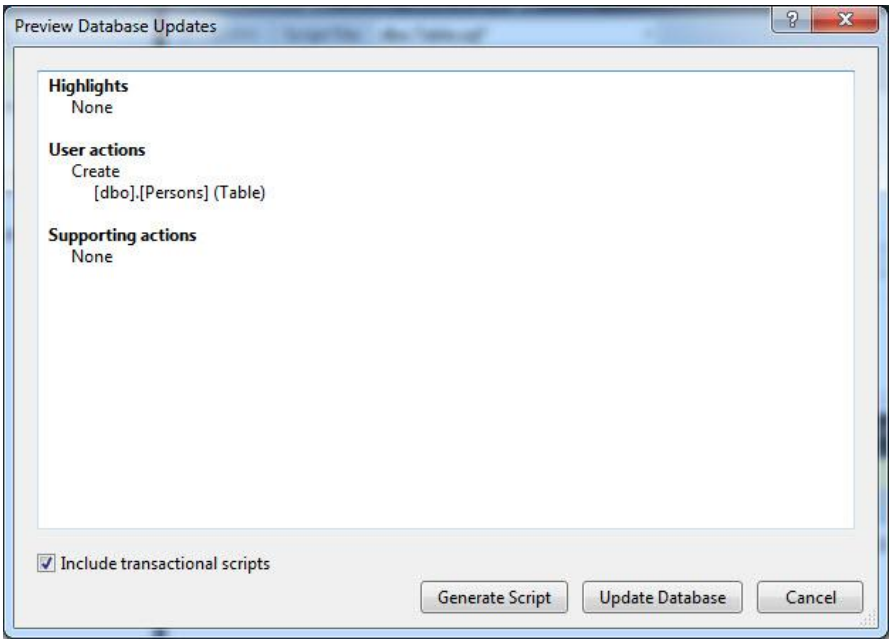
بعد از ایجاد دیتابیس مانند شکل زیر بر روی یکی از زیر پوشه‌های آن به نام `Tabels` راست کلیک کرده و گزینه `Add New Table` را می‌فشاریم :

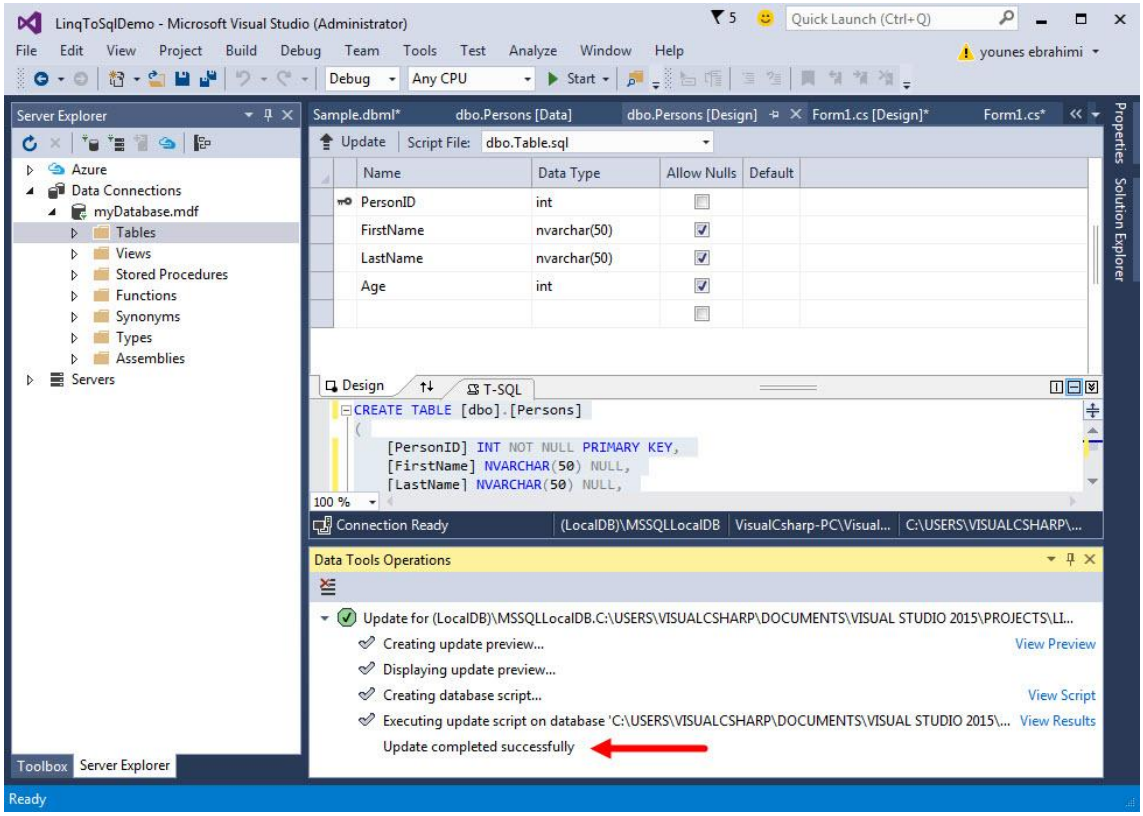


با کلیک بر روی گزینه مذکور صفحه ای به صورت زیر به نمایش در می‌آید که شما در این صفحه نام ستون‌های جدول را انتخاب کرده (۱)، نام جدول را به Persons تغییر داده (۲) و سپس بر روی دکمه Update کلیک کنید :

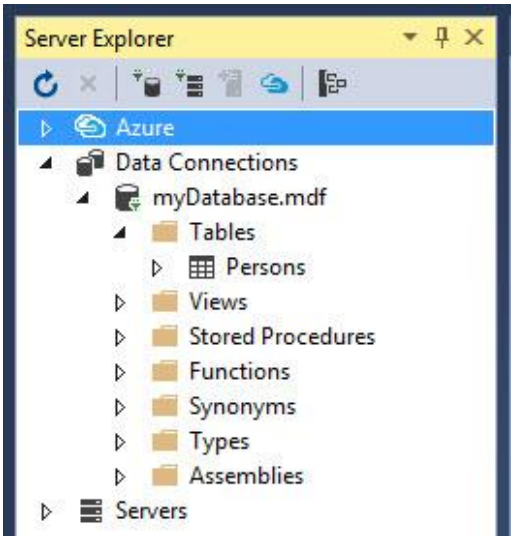


با کلیک بر روی دکمه Update پنجره ای به صورت زیر ظاهر می‌شود که بعد از زدن دکمه Update Database پیغامی مبنی بر موفقیت آمیز بود ایجاد جدول به شما نمایش داده می‌شود:

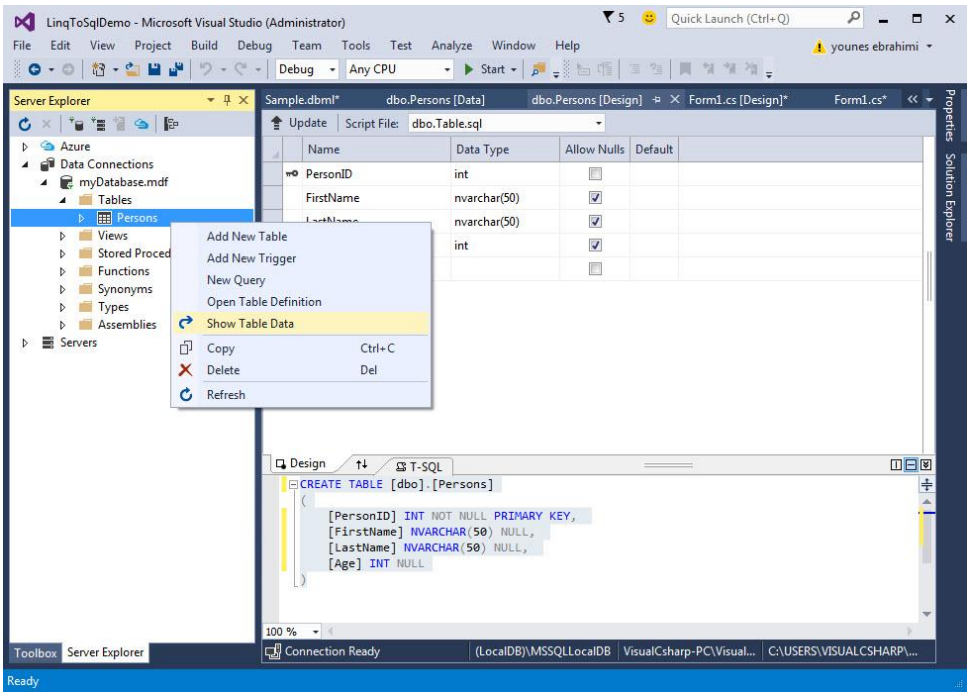




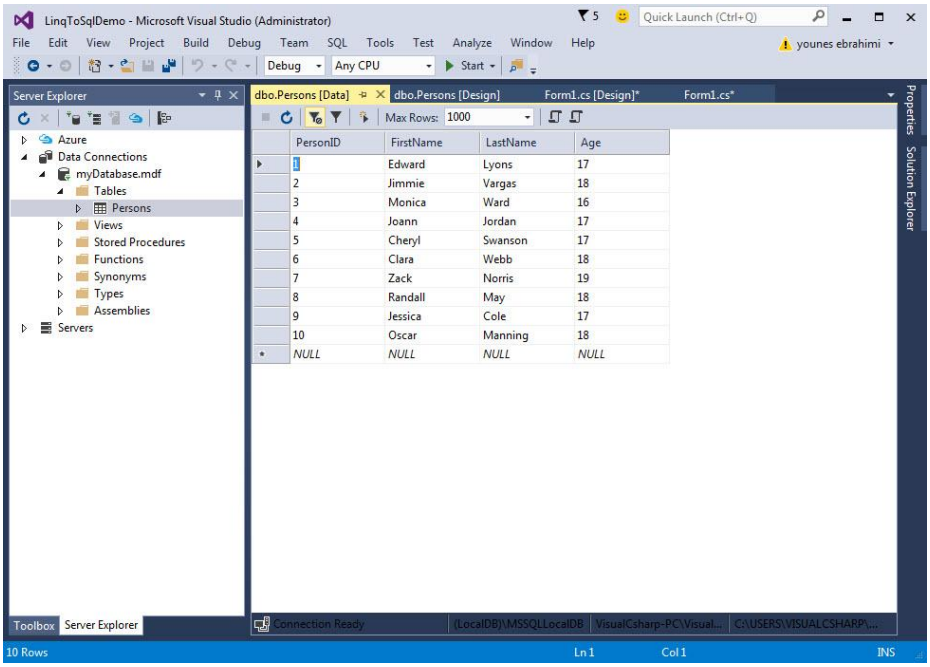
حال اگر به پنجره Server Explorer نگاه کنید مشاهده می‌کنید که یک جدول به نام Persons به دیتابیس Sample اضافه شده است :



بعد از ایجاد جدول نوبت به اضافه کردن داده‌ها به جدول مذکور می‌رسد، به صورت زیر بر روی نام جدول راست کلیک و سپس بر روی گزینه Show Table Data کلیک کنید :

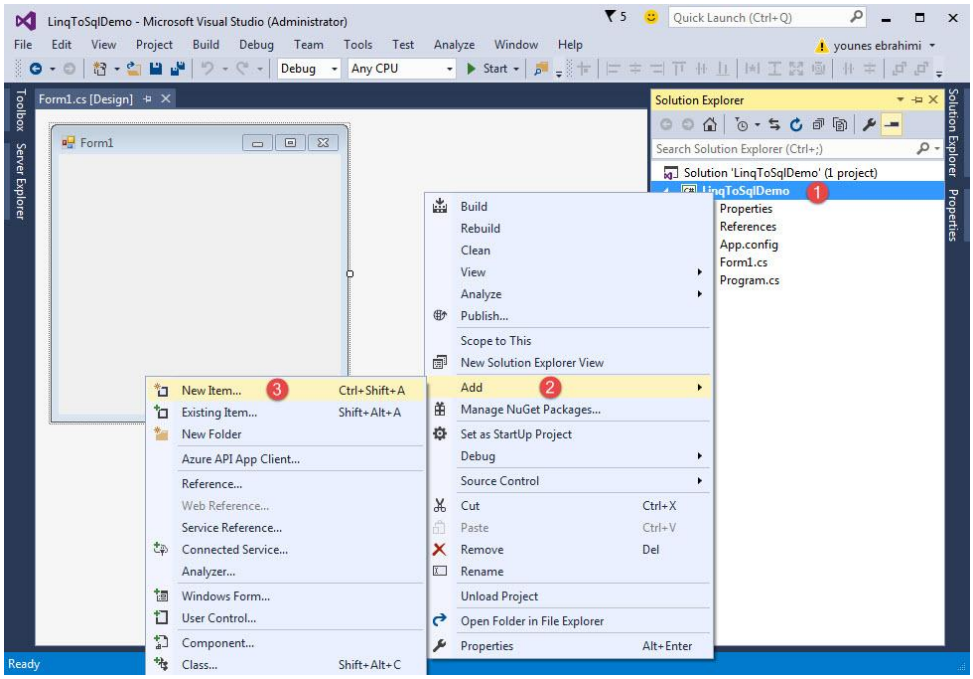


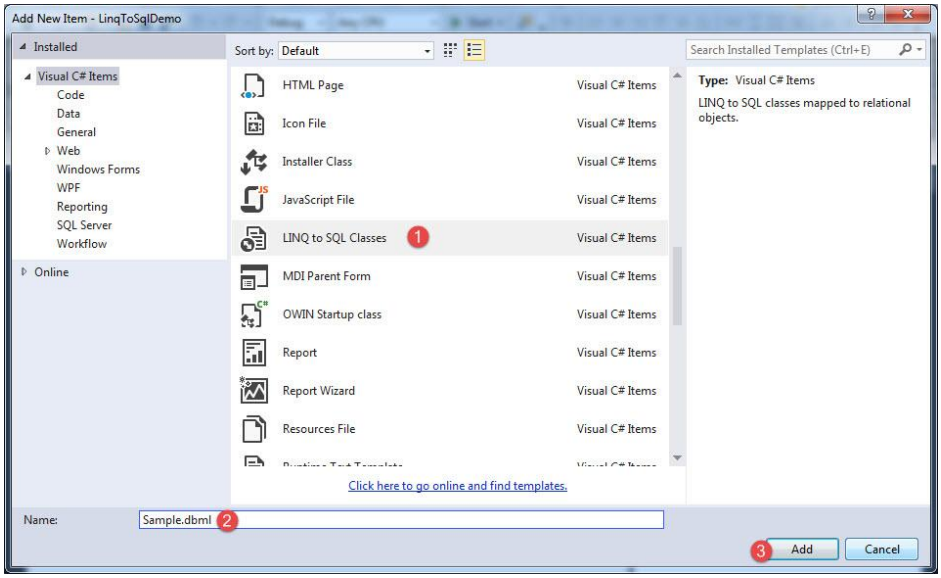
و در نهایت در زیر ستون‌های مربوطه داده‌های مورد نظرتان را وارد کنید :



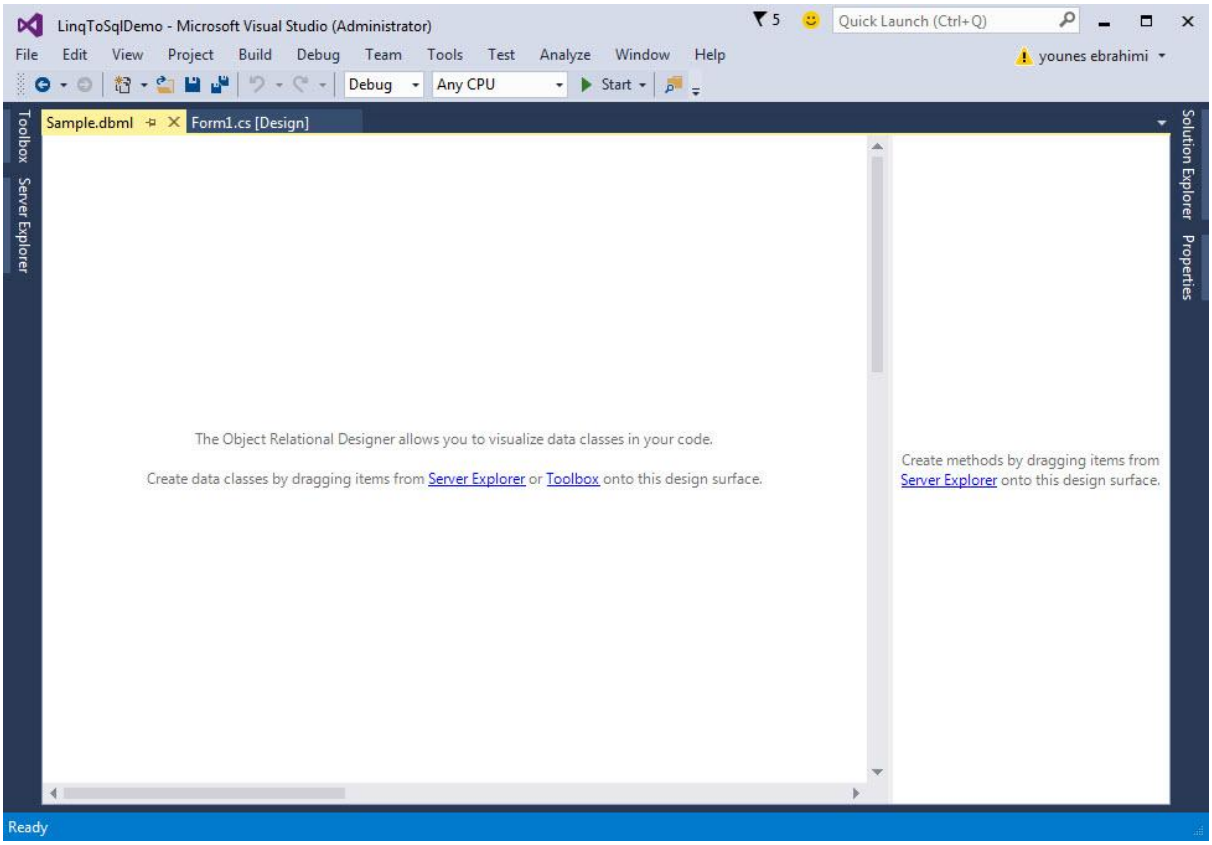
ایجاد کلاس‌های LINQ to SQL

یک برنامه ویندوزی ایجاد کرده و نام آن را LinqToSqlDemo بگذارید. وقتی که پروژه ایجاد شد، لازم است که یک فایل LINQ to SQL به آن اضافه کنیم. به صورت زیر بر روی نام پروژه راست کلیک کرده و گزینه Add و سپس New Item را بفشارید. سپس از صفحه باز شده گزینه LINQ to SQL Classes را انتخاب و نام آن را Sample گذاشته و بر روی دکمه Add کلیک کنید.



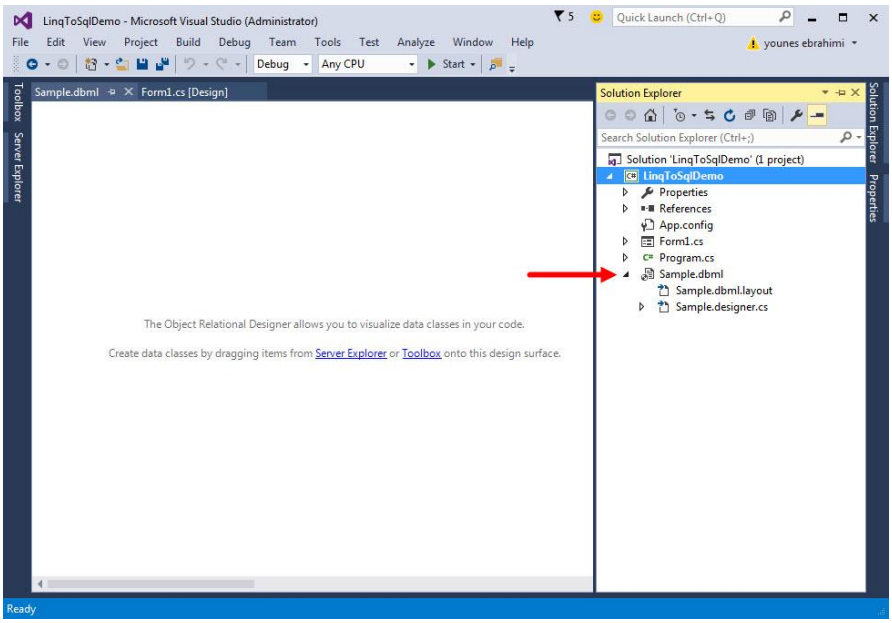


وقتی که بر روی دکمه Add کلیک کنید محیط Object Relational Designer ظاهر می‌شود.

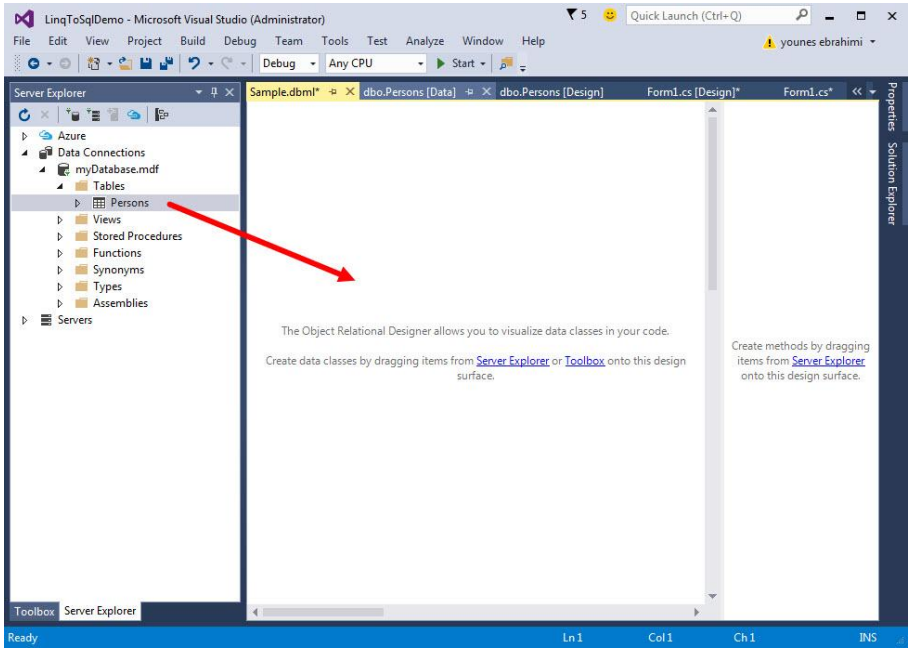


اکنون ToolBox شامل ابزارهایی برای ایجاد کلاس‌ها و ارتباط بین آنها می‌باشد. اما از آن جاییکه ما می‌خواهیم کلاس‌ها را با استفاده از یک جدول موجود در دیتابیس ایجاد کنیم، فعلاً از این ابزارها استفاده نمی‌کنیم. یک فایل DBML یا Database Markup Language با پسوند

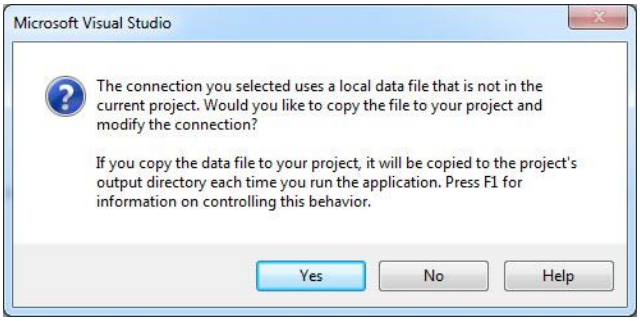
.dbml در Solution Explorer ایجاد و نمایش داده می‌شود. بر روی فلش کنار این فایل کلیک کرده تا فایل‌های زیر مجموعه آن نمایش داده شوند. بر روی فایل با پسوند DBML دوبار کلیک کرده تا Object Relational Desinger برای شما نمایش داده شود:



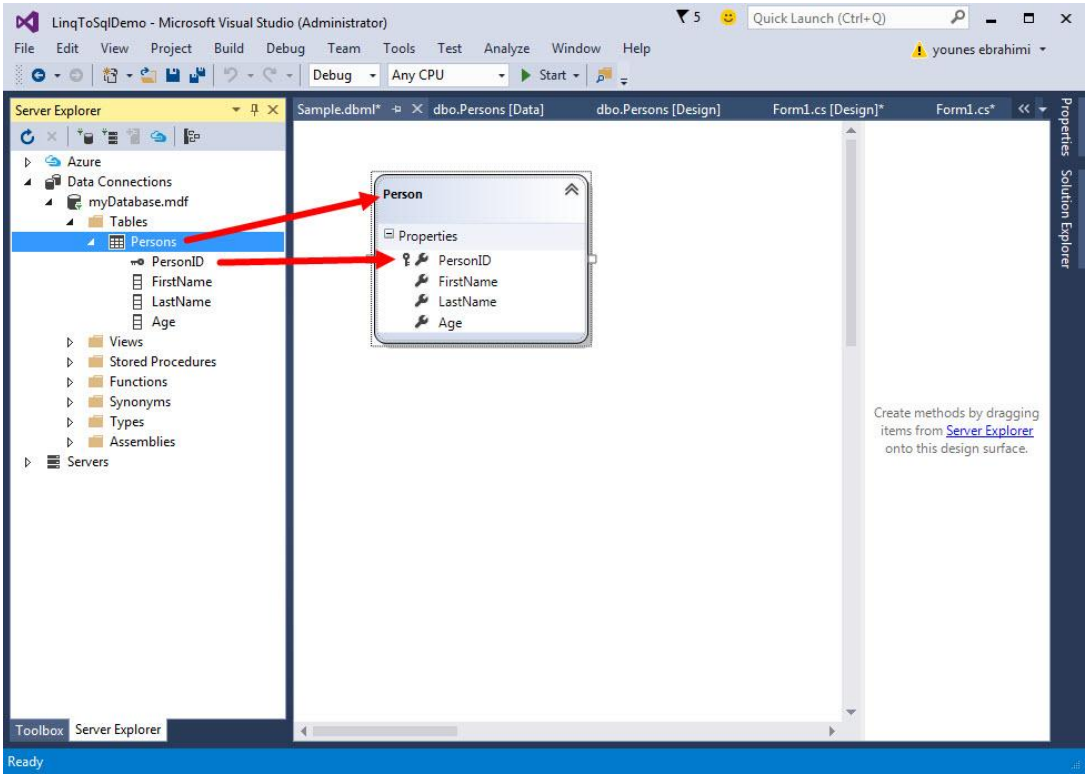
حال می‌خواهیم دیتابیس Sample را به پنجره‌ی Server Explorer اضافه نماییم. بر روی فلش کنار آن کلیک کرده تا زیر پوشه‌های این دیتابیس نمایش داده شوند، یکی از این پوشه‌ها Tables است که لیست تمامی جداول دیتابیس در آن قرار دارد. حال که تمامی جداول نمایش داده شدند، لازم است که جداول دلخواه را به محیط Object Relational Designer با ماوس بکشیم (Drag & Drop). در این درس ما جدول Persons را به این محیط می‌کشیم.



با کشیدن جدول به محیط Object Relational Designer پیغامی به شما نمایش داده می‌شود مبنی بر اینکه فایل دیتابیس در پوشه‌ی پروژه‌ی شما وجود ندارد، آیا مایل هستید که یکی نسخه (کپی) به پوشه پروژه انتقال داده شود؟ بر روی دکمه‌ی Yes کلیک کنید.



بعد از زدن دکمه‌ی Yes، به ازای هر جدول کشیده شده به محیط (مثلاً در اینجا Persons) یک کلاس همنام اما به صورت مفرد ایجاد می‌شود (مثلاً نام Persons به Person تبدیل می‌شود) و به ازای هر فیلد در جدول کشیده شده، یک خاصیت هم نام و با نوع سازگار در آن ایجاد می‌شود (به عنوان مثال نوع سازگار با nvarchar در دات نت string می‌باشد). در کل مهم‌ترین نکته در مورد LINQ to SQL همین است که، جدول موجود در دیتابیس به یک کلاس و فیلدهای جدول هم به خاصیت‌های کلاس تبدیل می‌شود.



همانطور که در شکل بالا مشاهده می‌کنید اسم جدول به Person و نوع PersonID به int تغییر داده می‌شود. حال فرض کنید که یک فیلد به نام Person در جدول موجود باشد. در این صورت اگر جدول را به محیطObject Relational Designer بکشید یک تداخل به وجود

می‌آید (بین اسم کلاس و اسم خصوصیت که هر دو Person می‌شود). برای حل این مشکل Visual Studio یک شماره به انتهای نام خصوصیت اضافه می‌کند. در این مثال اسم خصوصیت به Person1 تغییر داده می‌شود.

وقتی که جدول را به Object Relational Designer می‌کشیم، یک کلاس با پسوند DataContext ایجاد می‌شود. نامگذاری آن هم به این صورت است که قبل از نام این کلاس نامی که برای کلاس LINQToSQL انتخاب کرده‌ایم، می‌آید. مثلاً در مثال بالا چون ما موقع ایجاد کلاس‌های LINQToSQL نام Sample را انتخاب کردیم، در نتیجه کلاسی به نام SampleDataContext ایجاد می‌شود. با کلیک بر روی یک فضای خالی از Object Relational Designer به شما اجازه ویرایش خاصیت‌های کلاس DataContext با استفاده از پنجره Properties داده می‌شود. شما همچنین می‌توانید خواص کلاس سطر ایجاد شده و خواص اعضای آن را تغییر دهید. اما پیشنهاد می‌شود این نام‌ها و تنظیمات پیشفرض را تغییر ندهید.

اما اگر در مورد نحوه تولید کلاس‌ها کنجکاو هستید و می‌خواهید به نحوه پیاده سازی آنها نگاهی بیندازید به پنجره Solution Explorer رفته و بر روی فلش کنار فایل DBML ایجاد شده کلیک کنید. با اینکار دو فایل زیر مجموعه برای شما نمایش داده می‌شود. بر روی فایلی که پسوند آن designer.cs. دو بار کلیک کنید. با این کار، کلاس‌های جداول و کلاس DataContext برای شما نمایش داده می‌شوند. شما باید قبل از استفاده از فایل DBML آن را ذخیره کنید.

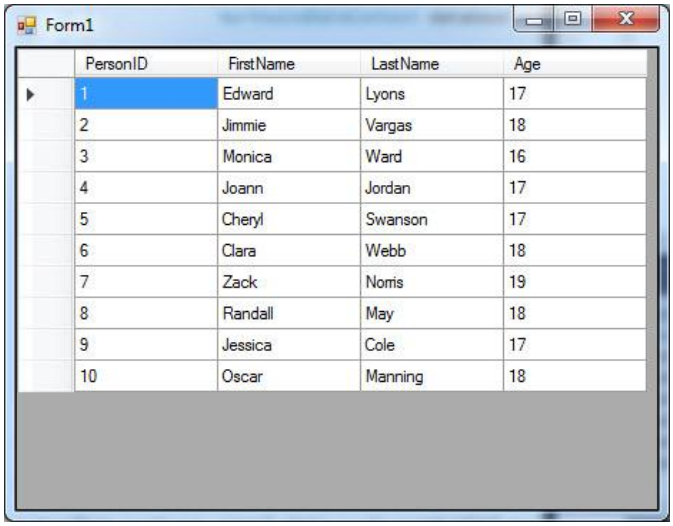
استفاده از کلاس‌های LINQ to SQL

وقتی که کلاس‌های LINQ to SQL با موفقیت تولید شدند می‌توانید از آنها در برنامه‌تان استفاده کنید. در این برنامه ما می‌خواهیم به طور ساده و فقط با استفاده از یک کنترل DataGridView نتایج پرس و جو را نمایش دهیم. به محیط طراحی رفته و یک کنترل DataGridView بر روی فرم و خاصیت Dock آن را بر روی Fill قرار دهید تا کل فضای فرم را در بر بگیرد. سپس فرم را به اندازه ای بزرگ کنید که کل نتایج پرس و جو را بتوان در دیتاگریدویو نمایش داد. سپس بر روی فرم دو بار کلیک کرده و کد زیر را در رویداد Load فرم بنویسید:

```
using System;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace LinqToSqlDemo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            SampleDataContext database = new SampleDataContext();

            var allPersons = from person in database.Persons
                             select new
                             {
                                 person.PersonID,
                                 person.FirstName,
                                 person.LastName,
```

```
25                              person.Age
26                          };
27
28              dataGridView1.DataSource = allPersons;
29          }
30      }
31  }
```

با دو بار کلیک بر روی عنوان فرم در محیط طراحی رویداد Load تولید می‌شود. سپس خطوط ۲۸-۱۷ را به آن اضافه کنید. در خط ۱۷ یک شیء از SampleDataContext ایجاد کرده‌ایم. با این کار دسترسی به جداول دیتابیس و سطرهای آن امکان پذیر می‌شود. در خطوط ۲۶-۱۹ با استفاده از یک کوئری Linq به تمامی رکوردهای جدول از طریق خصوصیت Persons مربوط به SampleDataContext دسترسی پیدا می‌کنیم. از طریق دستور select فقط چند خاصیت را انتخاب کرده‌ایم. و در نهایت در خط ۲۸ نتیجه را در خاصیت Datasource از کنترل Datagridview قرار داده‌ایم که این کار باعث نمایش داده‌ها در این کنترل می‌شود. با اجرای برنامه تمام رکوردهای جدول Persons را مشاهده خواهید کرد.



در خطوط ۲۶-۱۹ با استفاده از یک کوئری ساده Linq برخی از فیلدهای جدول Persons را انتخاب و نمایش دادیم. شما می‌توانید بسته به نیاز خود کوئری های مختلفی را بنویسید. به عنوان مثال می‌توانیم کوئری خطوط ۲۶-۱۹ را به شکلی تغییر دهیم که فقط افرادی که ۱۷ سال سن دارند نمایش داده شوند:

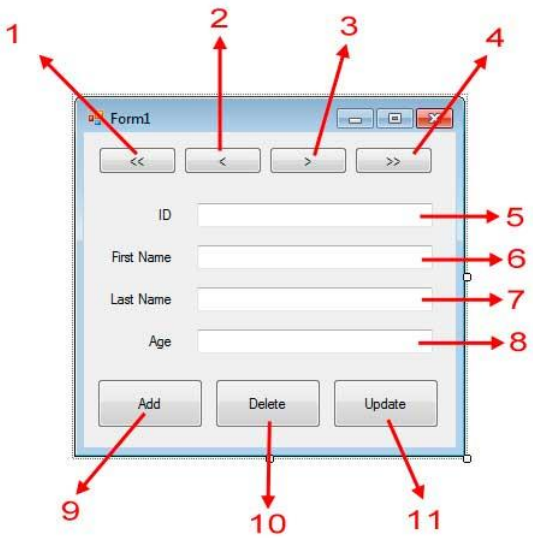
```
var allPersons = from person in database.Persons
                 where person.Age == 17
                 select new
                 {
                     person.PersonID,
                     person.FirstName,
                     person.LastName,
                     person.Age
                 };
```

ویرایش بانک اطلاعاتی با استفاده از LINQ to SQL

تبدیل جداول دیتابیس و رکوردهای آن به کلاس‌های متناضر در LINQ to SQL کار دستکاری دیتابیس را راحت تر می‌کند. وقتی که کلاس‌های LINQ to SQL تولید شدند شما به طور مستقیم می‌توانید اشیاء آنها را دستکاری کنید. کلاس DataContext دارای متدهایی برای حذف، اضافه و ویرایش کردن رکوردها می‌باشد. این متدها عبارت‌اند از DeleteOnSubmit()، InsertOnSubmit()، UpdateOnSubmit().

اگر بخواهید یک خاصیت از شیء که در اصل یک رکورد از جدول است را بروزرسانی کنید می‌توانید به طور مستقیم این کار را انجام دهید. همه این اعمال فوراً بر روی جداول اصلی و رکورهای آنها تأثیر نمی‌گذارند. لازم است متد SubmitChanges() از کلاس DataContext را فراخوانی کنیم. برای دستیابی به یک عنصر یا رکورد خاص از جدول می‌توانیم از متد ElementAt() استفاده کنیم. این متد یک مقدار از نوع صحیح می‌گیرد که بیانگر اندیس رکورد می‌باشد.

در برنامه ای که قرار است برای شما توضیح دهیم کاربر می‌تواند مشخصات افراد را مشاهده و همچنین آنها را حذف و ویرایش کند. برای کار با برنامه از دیتابیسی که در درس قبل ایجاد کردیم استفاده می‌کنیم. می‌خواهیم برنامه را به صورتی طراحی کنیم که کاربر در لحظه بتواند اطلاعات یک شخص (person) را به دست آورده و با استفاده از دکمه‌ها در بین سایر رکوردها حرکت کند. این برنامه، همچنین به کاربر اجازه حذف، اضافه و ویرایش اطلاعات را می‌دهد. مشاهده خواهید کرد که این کار با استفاده از کلاس‌های LINQ to SQL بسیار راحت است. حال به محیط طراحی رفته و برنامه مان را که دارای دکمه‌هایی برای حذف و اضافه و ویرایش رکوردها است را با توجه به شکل و جدول زیر طراحی می‌کنیم :



Number	Name
1	firstButton
2	prevButton
3	nextButton
4	lastButton

5	idTextBox
6	firstNameTextBox
7	lastNameTextBox
8	ageTextBox
9	addButton
10	deleteButton
11	updateButton

خاصیت ReadOnly جعبه متن idTextBox را بر روی true قرار می‌دهیم تا همانند یک کلید اصلی رفتار کرده و قابل تغییر نباشد. خاصیت StartPosition فرم را بر روی CenterScreen قرار می‌دهیم. دکمه‌های بالای هم برای نمایش رکوردهای قبلی، بعدی، اول و آخر و جعبه‌های متن هم برای نمایش اطلاعات شخص فعلی (current person) به کار می‌روند. دکمه‌های پایینی هم برای اضافه، حذف و ویرایش رکوردها مورد استفاده قرار می‌گیرند. با کلیک بر روی دکمه addButton محتوبات جعبه‌های متن پاک شده و شما می‌توانید مقادیر جدید را در داخل آنها نوشته و به جدول اضافه کنید. دکمه deleteButton برای حذف سطر فعلی و دکمه updateButton هم برای ویرایش آن به کار می‌رود. ما از کد زیر در برنامه مان استفاده می‌کنیم:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Windows.Forms;
5
6   namespace LinqToSqlDemo2
7   {
8       public partial class Form1 : Form
9       {
10          private int currentIndex;
11          private int minIndex;
12          private int maxIndex;
13          private bool addPending;
14          private SampleDataContext database;
15          private IEnumerable<Person> persons;
16
17          public Form1()
18          {
19              InitializeComponent();
20
21              database = new SampleDataContext();
22              persons = from p in database.Persons
23                        select p;
24
25              currentIndex = 0;
26
27              minIndex = 0;
28              maxIndex = persons.Count() - 1;
29
30              DisableButtons();
31
32              addPending = false;
33          }
34
35          private void Form1_Load(object sender, EventArgs e)
36          {
```

```
            ShowPersonInfo(currentIndex);
        }

        private void firstButton_Click(object sender, EventArgs e)
        {
            ShowPersonInfo(minIndex);
            currentIndex = minIndex;
            DisableButtons();
        }

        private void lastButton_Click(object sender, EventArgs e)
        {
            ShowPersonInfo(maxIndex);
            currentIndex = maxIndex;
            DisableButtons();
        }

        private void prevButton_Click(object sender, EventArgs e)
        {
            ShowPersonInfo(--currentIndex);
            DisableButtons();
        }

        private void nextButton_Click(object sender, EventArgs e)
        {
            ShowPersonInfo(++currentIndex);
            DisableButtons();
        }

        private void addButton_Click(object sender, EventArgs e)
        {
            if (addPending == false)
            {
                ClearFields();
                int newID = persons.Count() == 0 ? 1 : persons.Last().PersonID + 1;
                idTextBox.Text = newID.ToString();
                addButton.Text = "Done";
                addPending = true;
            }
            else
            {
                try
                {
                    //Create new person
                    Person newPerson = new Person();
                    newPerson.PersonID = Int32.Parse(idTextBox.Text);
                    newPerson.FirstName = firstNameTextBox.Text;
                    newPerson.LastName = lastNameTextBox.Text;
                    newPerson.Age = Int32.Parse(ageTextBox.Text);

                    //Add newPerson
                    database.Persons.InsertOnSubmit(newPerson);
                    database.SubmitChanges();
                    maxIndex++;
                    currentIndex = maxIndex;
                    DisableButtons();
                    MessageBox.Show("Successfully added to database.", "Success",
                                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                    addButton.Text = "Add";
                    addPending = false;
                }
                catch
                {
                    MessageBox.Show("Failed to add new record to database. Make sure " +
                                    "that every field is not empty and in a correct " +
                                    "format", "Failed",
                                    MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
104                 }
105             }
106         }
107 
108         private void deleteButton_Click(object sender, EventArgs e)
109         {
110             try
111             {
112                 database.Persons.DeleteOnSubmit(persons.ElementAt(currentIndex));
113                 database.SubmitChanges();
114 
115                 maxIndex--;
116 
117                 if (currentIndex > maxIndex)
118                     currentIndex--;
119 
120                 MessageBox.Show("Successfully removed from the database.", "Success",
121                     MessageBoxButtons.OK, MessageBoxIcon.Information);
122 
123                 ShowPersonInfo(currentIndex);
124                 DisableButtons();
125             }
126             catch
127             {
128                 MessageBox.Show("Unable to delete.", "Error",
129                                 MessageBoxButtons.OK, MessageBoxIcon.Error);
130             }
131         }
132 
133         private void updateButton_Click(object sender, EventArgs e)
134         {
135             try
136             {
137                 Person modifiedPerson = persons.ElementAt(currentIndex);
138                 modifiedPerson.FirstName = firstNameTextBox.Text;
139                 modifiedPerson.LastName = lastNameTextBox.Text;
140                 modifiedPerson.Age = Int32.Parse(ageTextBox.Text);
141 
142                 database.SubmitChanges();
143                 MessageBox.Show("Update success!", "Success",
144                                 MessageBoxButtons.OK, MessageBoxIcon.Error);
145             }
146             catch
147             {
148                 MessageBox.Show("Error on updating.", "Error",
149                                 MessageBoxButtons.OK, MessageBoxIcon.Information);
150             }
151         }
152 
153         private void ShowPersonInfo(int index)
154         {
155             if (persons.Count() == 0)
156             {
157                 ClearFields();
158                 MessageBox.Show("Nothing to show.", "Error",
159                                 MessageBoxButtons.OK, MessageBoxIcon.Error);
160                 return;
161             }
162 
163             Person currentPerson = persons.ElementAt(index);
164 
165             idTextBox.Text = currentPerson.PersonID.ToString();
166             firstNameTextBox.Text = currentPerson.FirstName;
167             lastNameTextBox.Text = currentPerson.LastName;
168             ageTextBox.Text = currentPerson.Age.ToString();
169         }
170 
```

```
171         private void DisableButtons()
172         {
173             if (persons.Count() <= 1)
174             {
175                 firstButton.Enabled = false;
176                 prevButton.Enabled = false;
177                 nextButton.Enabled = false;
178                 lastButton.Enabled = false;
179                 return;
180             }
181 
182             if (currentIndex == minIndex)
183             {
184                 firstButton.Enabled = false;
185                 prevButton.Enabled = false;
186                 nextButton.Enabled = true;
187                 lastButton.Enabled = true;
188             }
189             else if (currentIndex == maxIndex)
190             {
191                 firstButton.Enabled = true;
192                 prevButton.Enabled = true;
193                 nextButton.Enabled = false;
194                 lastButton.Enabled = false;
195             }
196             else if (currentIndex > minIndex && currentIndex < maxIndex)
197             {
198                 firstButton.Enabled = true;
199                 prevButton.Enabled = true;
200                 nextButton.Enabled = true;
201                 lastButton.Enabled = true;
202             }
203         }
204 
205         private void ClearFields()
206         {
207             idTextBox.Text = String.Empty;
208             firstNameTextBox.Text = String.Empty;
209             lastNameTextBox.Text = String.Empty;
210             ageTextBox.Text = String.Empty;
211             firstNameTextBox.Focus();
212         }
213     }
214 }
```

در خطوط ۱۵-۱۰ متغیرهای لازم که قرار است از آنها در داخل برنامه استفاده کنیم را، تعریف کرده‌ایم. در خط ۱۰ متغیری که اندیس شخص (person) فعلی را در خود ذخیره می‌کند را، تعریف کرده‌ایم. در خطوط ۱۲-۱۱ متغیرهایی تعریف شده‌اند که برای نگهداری اندیس‌های اول و آخر در بانک به کار می‌روند. متغیر خط ۱۳ همانطور که بعداً مشاهده خواهید کرد، در دکمه addButton به کار می‌رود. در خط ۱۴ یک شیء SampleDataContext که مترادف با دیتابیس Sample ایجاد کرده‌ایم که از آن در فراخوانی متدهای حذف، اضافه، ویرایش و واکشی رکوردهای جداول دیتابیس Sample استفاده خواهیم کرد. در خط ۱۵ یک شیء از نوع IEnumerable تعریف کرده‌ایم و همه رکوردهای Person پرس و جو شده از دیتابیس را در آن قرار می‌دهیم.

از آنجاییکه خروجی پرس و جوی Linq از نوع IEnumerable است پس نوع متغیر persons در خط ۱۵ را Ienumerable قرار می‌دهیم. این کار را برای این انجام می‌دهیم که هر بار نیازی به واکشی تمامی اطلاعات نداشته باشیم. تنها یک بار این کار را بر روی دیتابیس انجام می‌دهیم و خروجی آنرا در یک متغیر سراسری (persons) قرار می‌دهیم. ابتدا متدهای کاربردی که در برنامه بالا مورد استقاده قرار گرفته‌اند را توضیح

می‌دهیم. متد ()ClearFields که در خطوط ۲۱۲-۲۰۵ تعریف شده است برای پاک کردن جعبه‌های متن و قرار دادن Focus بر روی جعبه متن firstNameTextBox به کار می‌رود.

از متد ()DisableButtons (خطوط ۲۰۳-۱۷۲) برای غیر فعال کردن دکمه‌هایی که نشان دهنده‌ی رکوردهای اول و آخر دیتابیس هستند، استفاده می‌شود. متد مذکور اینکار را با استفاده از چک کردن اینکه آیا کاربر به ابتدا یا انتهای مجموعه رکوردها رسیده است، انجام می‌دهد. متد ()ShowPersonInfo (خطوط ۱۶۹-۱۵۳) یک اندیس می‌گیرد و شیء Person متناظر با آن اندیس را بر می‌گرداند. در خطوط ۱۶۱-۱۵۵ این متد چک می‌کند که آیا رکوردی برای نمایش وجود دارد یا خیر؟ اینکار با استفاده از متد ()Count فیلد persons انجام می‌شود. در صورتی که رکوردی برای نمایش وجود نداشته باشد یک پیغام خطا به کاربر نمایش می‌دهیم و با دستور return از اجرای بقیه دستورات متد جلوگیری می‌کنیم. در خط ۱۶۳ این متد با استفاده از متد ()ElementAt از فیلد persons شیء با اندیس مناسب را برگشت می‌دهیم. سپس خصوصیات آن شیء را نمایش در جعبه‌های متن متناظر نمایش می‌دهیم. حال اجازه دهید به سازنده فرم در خطوط (۳۳-۱۷) نگاهی بیندازیم. در خط ۲۱ یک نمونه از SampleDataContext ایجاد کرده‌ایم در نتیجه، می‌توانیم عملیات دلخواه را بر روی دیتابیس انجام دهیم. در خطوط ۲۳-۲۲ یک پرس و جوی ساده LINQ که لیست تمامی رکوردها را برگشت می‌دهد را روی دیتابیس انجام می‌دهیم. سپس مقدار متغیر currentIndex را بر روی ۰ قرار می‌دهیم. با این کار به برنامه می‌گوییم که مشخصات اولین رکورد را نمایش دهد. در خطوط ۲۸-۲۷ متغیرهای minIndex و maxIndex را مقداردهی می‌کنیم. که نشان دهنده‌ی اندیس اولین و آخرین رکوردها هستند. اولین رکورد در مجموعه دارای اندیس ۰ می‌باشد. به همین دلیل مقدار minIndex روی ۰ تنظیم شده است. مقدار تعداد کل رکوردها منهای ۱ را به متغیر maxIndex اختصاص می‌دهیم زیرا، اندیس‌ها از ۰ شروع می‌شوند.

متد ()DisableButtons بسته به نوع شرایط رفتارهای متفاوتی از خود نشان می‌دهد. به عنوان مثال، اگر به اول مجموعه رکوردها رسیده باشیم (یعنی مقدار currentIndex برابر 0 باشد) دکمه‌ی قبلی و اولین را غیر فعال می‌کنیم، زیرا نیازی به آنها نیست. و در آخر مقدار متغیر addPendingرا برابر false قرار می‌دهیم. همانطور که بعداً مشاهده خواهید کرد از این متغیر در کنترل کننده رویداد addButton استفاده خواهد شد. به محیط طراحی برگشته و با دوبار کلیک بر روی عنوان فرم یک کنترل کننده رویداد برای رویداد load فرم ایجاد می‌کنیم (خطوط ۳۸-۳۵). در داخل کنترل کننده رویداد متد ()ShowPersonInfo را فراخوانی کرده و مقدار فعلی currentIndex که همان ۰ می‌باشد را، به آن ارسال می‌کنیم. این کار باعث نمایش اطلاعات اولین رکورد در جعبه‌های متن می‌شود. حال نوبت به اضافه کردن رویداد کلیک به دکمه‌ی پیمایشگر رسیده است.

در محیط طراحی بر روی دکمه‌ی firstButton کلیک کنید و خطوط ۴۴-۴۲ را به کنترل کننده رویداد آن اضافه کنید. در اولین خط با فراخوانی متد()ShowPersonInfo و ارسال مقدار minIndex به آن اولین رکورد را نمایش می‌دهیم. سپس مقدار متغیر minIndex را به متغیر currentIndex نسبت می‌دهیم. این بدین معنی است که اولین رکورد همان رکورد جاری است. سپس با فراخوانی متد ()DisableButtons دکمه‌های firstButton و prevButton را غیر فعال می‌کنیم. با کمی دقت در کدهای بالا که کدهای prevButton و nextButton مشابه کدهای دکمه‌ی firstButton می‌باشند. در کدهای این دو دکمه نیز متد ()ShowPersonInfo فراخوانی شده و مقدار فیلد currentIndex برای نمایش رکورد مناسب به آن ارسال می‌شود. این کار با افزایش یا کاهش مقدار currentIndex و ارسال آن به این متد انجام می‌شود.

به عنوان مثال زمانی که قصد داریم رکورد قبلی را نمایش دهیم. ابتدا باید یک واحد از مقدار این متغیر کم کنیم و سپس مقدار جدید را به متد ShowPersonInfo() ارسال کنیم. کنترل کننده رویداد addButton (خطوط ۱۰۶-۶۶) دارای قابلیت‌های زیر است. دکمه‌ی addButton دارای ۲ حالت و وضعیت می‌باشد، اولین حالت زمانی است که مقدار فیلد addPending برابرfalse باشد. در این حالت زمانی که بر روی این دکمه کلیک شود مقادیر موجود در جعبه‌های متن پاک می‌شود (خط ۷۰). سپس مقدار مناسب برای خاصیت PersonID رکورد جدید محاسبه می‌شود. زیرا مقدار این خاصیت در بانک اطلاعاتی نباید تکراری باشد. نحوه‌ی محاسبه بدین شکل است که اگر رکوردی در بانک وجود نداشته باشد مقدار این خاصیت برابر ۱ می‌شود (خط ۷۱) و در غیر اینصورت مقدار خاصیت PersonID آخرین رکورد بانک اطلاعاتی را بدست می‌آوریم و یک واحد به آن اضافه می‌کنیم و در نهایت این مقدار را در جعبه متن مربوطه نمایش می‌دهیم. متن دکمه‌یaddButton را به Done تغییر می‌دهیم که نشان دهنده‌ی این است که برنامه منتظر این است که کاربر اطلاعات جدید را در جعبه‌های متن وارد کرده تا به بانک اطلاعاتی اضافه شوند. در نهایت مقدار true را در فیلد addPending قرار می‌دهیم تا وضعیت دوم دکمه فعال شود. وضعیت دوم دکمه زمانی فعال می‌شود که برنامه منتظر وارد کردن مقادیر توسط کاربر است. وقتی که کاربر برای بار دوم بر روی دکمه‌یaddButton کلیک می‌کند، رکورد جدید در بانک اطلاعاتی ذخیره می‌شود.

استفاده از متد SubmitChanges() ممکن است باعث بروز خطا یا استثناءهایی مانند FormatExceptions یا ChangeConflictExceptions شود، به همین خاطر دستورات اضافه کردن رکورد جدید را در بلاک try…catch قرار داده‌ایم (۱۰۴-۷۸). در خطوط ۸۵-۸۱ یک شیء جدید Person ایجاد کرده‌ایم و مقادیر خاصیت‌های آن را برابر جعبه‌های متن متناظر قرار داده‌ایم. شیء ایجاد شده را در خط ۸۸ به متد InsertOnSubmit() ارسال می‌کنیم. وظیفه اصلی این متد ثبت تغییرات در بانک اطلاعاتی می‌باشد. همانطور که از اسم متد مشخص است، تغییرات زمانی در بانک اطلاعاتی اعمال می‌شوند که متد SubmitChanges() فراخوانی شود. فراخوانی این متد (SubmitChanges()) را در خط ۸۹ باعث اعمال تغییرات (در اینجا اضافه شدن رکورد جدید) به جدول Persons می‌شود.

در خط ۹۰ مقدار فیلد maxIndex را یک واحد افزایش داده‌ایم. زیرا یک رکورد جدید ثبت شده و تعداد کل رکوردها یک واحد افزایش پیدا کرده است. بعد از ثبت رکورد جدید مقدار currentIndex را به فیلدmaxIndex اختصاص می‌دهیم. سپس متد DisableButtons() را فراخوانی کرده و یک پیغام مبنی بر ثبت موفقیت آمیز رکورد جدید در دیتابیس را به کاربر نمایش می‌دهیم. و در نهایت عنوان دکمه را به Add و مقدار فیلد addPending را به false تغییر می‌دهیم تا وضعیت اول دکمه فعال شود. در قسمت Catch به شکل ساده یک پیغام خطا را به کاربر مبنی بر عدم موفقیت یا مشکل در ثبت اطلاعات نشان می‌دهیم. کدهای اداره کننده رویداد کلیک دکمه‌ی deleteButton (خطوط ۱۳۱-۱۰۸) را در بلوک try…catch قرار می‌دهیم تا از استثناءهای رخ داده را مدیریت کنیم. در خطوط ۱۱۲ از متد DeleteOnSubmit() برای ثبت تغییرات (در اینجا حذف رکورد) در پایگاه داده استفاده کرده‌ایم. این متد به عنوان ورودی یک شیء را دریافت می‌کند و رکورد مرتبط با آن را از جدول پایگاه داده حذف می‌کند.

برای بدست آوردن شیء ای که می‌بایست حذف شود، از متد ElementAt() استفاده می‌کنیم و مقدار فیلد currentIndex را به آن ارسال می‌کنیم. در نهایت برای اعمال تغییرات در بانک اطلاعاتی باید متد SubmitChanges() را فراخوانی کنیم. دقت داشته باشید اگر این متد را فراخوانی نکنید، تغییرات در بانک اطلاعاتی اعمال نمی‌شود. سپس مقدار فیلد maxIndex را یک واحد کاهش می‌دهیم. زیرا یک رکورد از بانک

اطلاعاتی حذف شده است. همچنین مقدار مناسب را به فیلد currentIndex نسبت می‌دهیم. کاهش مقدار فیلد maxIndex باعث می‌شود که مقدار فیلد currentIndex از آن بیشتر شود. به همین خاطر، باید مقدار این فیلد (currentIndex) را نیز کاهش دهیم. در نتیجه رکورد قبل از رکورد حذف شده نمایش داده می‌شود. در اداره کننده رویداد کلیک دکمه updateButton برای انجام عمل ویرایش اطلاعات ابتدا یک شیء از کلاس Person ایجاد می‌کنیم و مقدار خاصیت‌های آنرا با مقادیر جدید پر می‌کنیم، سپس برای انجام عمل ویرایش این شیء را به متد UpdateOnSubmit() ارسال می‌کنیم و در نهایت برای اعمال تغییرات متد SubmitChanges() را فراخوانی می‌کنیم (خطوط ۱۵۱-۱۳۳).

متدهای بهم پیوسته (Aggregate Methods) در LINQ

LINQ دارای عملگرها یا متدهای به هم پیوسته‌ای است که به شما اجازه می‌دهند که عملیات ریاضی را بر روی مقادیر موجود در یک مجموعه انجام دهید. در ادامه در مورد برخی از این متدها توضیح می‌دهیم.

متدهای Count و LongCount

با استفاده از متدهای Count() و LongCount() می‌توان تعداد عناصر یک کلکسیون یا آرایه را به دست آورد. از متد Count() برای به دست آوردن تعداد عناصر یک کلکسیون و از متد LongCount() برای به دست آوردن تعداد عناصر یک کلکسیون بزرگ استفاده می‌شود. به عنوان مثال نحوه استفاده از عملگر Count در زیر آمده است:

```
int[] numbers = { 7, 2, 6, 1, 7, 4, 2, 5, 1, 2, 6 };
Console.WriteLine("numbers has {0} elements.", numbers.Count());
```

یکی از سربارگذاری‌های دو متد فوق به شما اجازه شمارش عناصری خاص را می‌دهد:

```
Console.Write("Number of even numbers in the collection: {0}", numbers.Count(n => n % 2 == 0));
```

عبارت لامبدا در سربارگذاری متد Count() در مثال بالا تعیین کننده شرایط عنصری است که ما می‌خواهیم مورد شمارش قرار بگیرد. که در این مثال اعداد زوج می‌باشد.

متدهای Min و Max

متدهای Min() و Max() کوچک‌ترین و بزرگ‌ترین مقادیر موجود در یک آرایه را مشخص می‌کنند. نسخه‌های بدون پارامتر این دو متد این کار را به سادگی انجام می‌دهند:

```
int[] numbers = { 7, 2, 6, 1, 7, 4, 2, 5, 1, 2, 6 };
Console.WriteLine("Min = {0}\nMax = {1}", numbers.Min(), numbers.Max());
```

برای اشیاء پیچیده تری که هیچ پیاده سازی برای IComparable ندارند، می‌توان از یکی از سربارگذاری‌های این متدها استفاده کرد که اشیاء را بر اساس یکی از خصوصیات آنها مقایسه می‌کنند:

```
List<Person> people = GetPersonList(); //Assume the method returns a collection of Person objects
```

```
Console.WriteLine("Youngest Age is {0}", people.Min( p=>p.Age ));
Console.WriteLine("Oldest Age is {0}", people.Max( p=>p.Age ));
```

همانطور که در مثال بالا مشاهده می‌کنید ما لیستی از اشخاص داریم که متدهای Min() و Max() آنها را بر اساس خاصیت Age شان مقایسه می‌کنند و کمترین و بالاترین سن آنها را مشخص می‌نمایند.

متد Average

متد Average() برای به دست آوردن میانگین مقادیر عددی به کار می‌رود. این متد حاصل جمع تمام مقادیر عناصر را بر تعداد عناصر تقسیم می‌کند:

```
int[] numbers = { 7, 2, 6, 1, 7, 4, 2, 5, 1, 2, 6 };

Console.WriteLine("Average = {0}", numbers.Average());
```

مانند متدهای Min() و Max() دارای یک سربارگذاری است که اشیاء را براساس یک کلید یا خصوصیت، مقایسه می‌کند.

```
List<Person> people = GetPersonList();

Console.WriteLine("Average age of all the persons");
Console.WriteLine("Average = {0}", people.Average( p => p.Age ));
```

همانطور که مشاهده می‌کنید در مثال بالا متد Average() متوسط سن افراد را بر می‌گرداند.

متد Sum

این متد مجموع همه مقادیر یک مجموعه را بر می‌گرداند.

```
int[] numbers = { 7, 2, 6, 1, 7, 4, 2, 5, 1, 2, 6 };

Console.WriteLine("Sum = {0}", numbers.Sum);
```

این متد نیز دارای یک سربارگذاری است که یک خاصیت از مجموعه اشیاء را با هم جمع می‌کند.

```
List<Person> people = GetPersonList();

Console.WriteLine("Total age of all the persons");
Console.WriteLine("Total = {0}", people.Sum(p => p.Age));
```

در مثال بالا خاصیت Age همه افراد با هم جمع می‌شوند.

متد Aggregate

این متد نسخه انعطاف پذیرتر متد Sum() است. به جای اینکه شما را محدود به جمع مقادیر یک مجموعه بکند، می‌توانید دو پارامتر برای آن تعریف کنید که اولین پارامتر، اولین عدد یا نتیجه قبل و دومین پارامتر عدد دومی باشد که قرار است در عملیات شرکت کند.

```
int[] numbers = { 1, 2, 3, 4, 5 };
Console.WriteLine( numbers.Aggregate( (n1, n2) => n1 + n2 ) );
```

این متد دارای دو پارامتر لامبدا در داخل بدنه خود است. ابتدا دو عدد از آرایه را به آن اضافه می‌کنیم. اولین عدد، عدد اول آرایه (۱) و دومین عدد، عدد دوم آرایه (۲) است. این دو مقدار با هم جمع می‌شوند و در n1 ذخیره می‌شوند. مقدار سوم (۳) در n2 قرار می‌گیرد و با n1 که قبلاً جمع دو عدد قبلی در آن قرار گرفته است (۳) جمع می‌شود. این کار تا آخرین عنصر آرایه ادامه می‌یابد. نتیجه نهایی همانند نتیجه‌ای است که متد Sum() برمی گرداند، یعنی جمع همه مقادیر. نکته مهمی که در مورد Aggregate() وجود دارد این است که شما می‌توانید عملگری را که می‌خواهید در عملیات مورد استفاده قرار دهید را انتخاب کنید. در مثال زیر عملگر را به عملگر ضرب تغییر می‌دهیم. نتیجه نهایی حاصلضرب مقادیر یعنی ۱۲۰ خواهد شد:

```
int[] numbers = { 1, 2, 3, 4, 5 };
Console.WriteLine( numbers.Aggregate( (n1, n2) => n1 * n2 ) );
```

متد را می‌توان به صورت زیر پیچیده تر هم کرد:

```
int[] numbers = { 1, 2, 3, 4, 5 };
Console.WriteLine( numbers.Aggregate( (n1, n2) => (n1 / 2) + (n2 / 2) ) );
```

متد بالا قبل از جمع مقادیر ابتدا آنها را بر دو تقسیم می‌کند. متد Aggregate() دارای سربارگذاری دیگری نیز هست که در آن یک مقدار ثابت را می‌توان به حاصل جمع مقادیر اضافه کرد:

```
int[] numbers = { 1, 2, 3, 4, 5 };
Console.WriteLine( numbers.Aggregate( 1, (n1 , n2) => n1 + n2 ));
```

حاصل جمع مقادیر آرایه ۱۵ می‌شود و چون ما مقدار ثابت ۱ را سربارگذاری متد Aggregate() قرار داده‌ایم جمع نهایی ۱۶ خواهد شد. سومین سربارگذاری متد Aggregate() آرگومان سومی را قبول می‌کند که برای قالب بندی نتیجه نهایی مورد استفاده قرار می‌گیرد:

```
Console.WriteLine( numbers.Aggregate( 1, (n1, n2) => n1 + n2, n => String.Format("{0:C}", n)));
```

استفاده از متدهای به هم پیوسته کمک زیادی به برنامه نویسان در کاهش مدت زمان برنامه‌نویسی می‌کند.

فصل پنجم

ADO.NET و دیتابیس ها

ADO.NET و دیتابیس‌ها

بیشتر برنامه‌های امروزی از روش‌های مختلفی برای ذخیره سازی داده‌ها استفاده می‌کنند. شاید برنامه‌هایی که در درس‌هایی قبل ایجاد شده‌اند جزو دسته برنامه‌هایی باشند که از روش‌های ذخیره سازی اطلاعات استفاده نمی‌کنند چون برنامه‌های بسیار ساده و کوچکی هستند. یک برنامه می‌تواند دارای انواع مختلفی از منابع داده مانند فایل text و فایل XML و همچنین یک database باشد. از Databaseها معمولاً برای ذخیره انواع داده مانند نام، آدرس، سن، جنس و شغل یک شخص، آهنگ، تصویر و بسیاری چیزهای دیگر استفاده می‌شود. یک دیتابیس مجموعه‌ای است از انواع مختلف داده‌های ساخت یافته در داخل جداولی که شامل فیلدها و رکوردها می‌باشند. بیشتر برنامه‌های امروزی از دیتابیس برای ذخیره اطلاعات استفاده می‌کنند. دیتابیس‌های رابطه ای، شامل داده‌های هستند که به صورت سازمان یافته با همدیگر در ارتباط هستند. این نوع دیتابیس‌ها شامل یک یا چند جدول به هم پیوسته هستند. جداول شامل سطر و ستون هستند. در دیتابیس‌ها یک سطر نشان دهنده یک رکورد است. در یک دیتابیس که شامل رکوردهای کارمند است، یک سطر نشان دهنده یک رکورد از یک کارمند است. ستون نشان دهنده فیلدها یا خواص و صفت می‌باشد. به عنوان مثال یک کارمند دارای یک فیلد مثلاً FirstName (نام) و یک فیلد LastName (نام خانوادگی) و یک فیلد Age (سن) است. می‌توانید بین چندین جدول ارتباط برقرار کنید. به عنوان مثال یک جدول کارمند می‌تواند دارای یک فیلد به نام City_ID باشد. سپس جدول دیگر به نام Cities می‌تواند شامل فیلدهای City_ID و CityName باشد. شما می‌توانید بین این دو جدول ارتباط برقرار کنید. Structured Query Language یا SQL محبوب‌ترین و شاید استانداردترین راه برقراری ارتباط با دیتابیس می‌شود. این زبان دارای دستوراتی است که شما به وسیله آنها می‌توانید داده‌های دیتابیس را بروز کرده، بازیابی، اضافه و حذف کنید. همچنین به شما اجازه ایجاد و تغییر دیتابیس و جداول و ایجاد ارتباط بین جداول مختلف را می‌دهد. Database Management Systems یا DBMS مانند SQL server به شما اجازه دسترسی سریع به داده‌های دیتابیس را می‌دهد و شامل ابزارهای مختلفی برای پرس و جو، ایجاد، حذف و آپدیت دیتابیس می‌باشد. بیشتر DBMSها یک محیط گرافیکی جهت انجام امور مختلف برای شما فراهم می‌آورند. مثال‌هایی از DBMS عبارت‌اند از Access، Oracle و MySQL.

ADO.NET قسمتی از تکنولوژی دات‌نت است که به شما اجازه دسترسی و تغییر داده‌های منابع داده مختلف را می‌دهد. این تکنولوژی از Microsoft SQL Server، MySQL، Oracle و Mircrosoft Access پشتیبانی می‌کند. دات‌نت فریم ورک تعدادی data providers برای استفاده در اختیار شما قرار می‌دهد. از این data providersها برای ارتباط با منابع داده، اجرای دستورات و به دست آوردن نتایج استفاده می‌شود. در درس‌های آینده پروژه‌هایی ارائه شده است که با مطالعه آنها نحوه اتصال دیتابیس به برنامه‌های ویندوزی را خواهید آموخت.

مبانی SQL

قبل از اینکه اقدام به دسترسی و تغییر دیتابیس‌ها کنید لازم است در مورد مبانی SQL اطلاعاتی داشته باشید. از این زبان برای پرس و جو در دیتابیس استفاده می‌شود. درک SQL بسیار ساده و آسان است. از SQL تنها برای پرس و جوی داده استفاده نمی‌شود، بلکه تقریباً از آن برای انجام هر کاری مثلاً ایجاد، بروزرسانی و حذف دیتابیس و جداول استفاده می‌شود. از آن جاییکه این کتاب در مورد SQL نیست فقط در مورد مباحثی از آن که مورد نیازمان است توضیح می‌دهیم.

انواع داده‌های SQL

SQL همانند اکثر زبآنها دارای انواع داده ای است که در جدول زیر به مهم‌ترین آنها اشاره شده است:

نوع داده	توضیح	مثال
int	داده‌های صحیح	-3, 1, 12, 57
char(n)	رشته ای با طول ثابت که در آن n نهایت طول رشته است	‘hello’, ‘goodbye’ ‘
varchar(n)	رشته ای با طول متغیر که در آن n نهایت طول رشته است	‘hello’, ‘goodbye’ ‘
datetime	برای ذخیره ساعت و روز	Jan 1, 2010 3:00PM
date	برای ذخیره فقط روز	Jan 1, 2010
time	برای ذخیره فقط ساعت	3:00PM
money	برای ذخیره نوع پول	123.45

به این نکته توجه کنید که char(n) و varchar(n) به نظر می‌رسد شبیه هم باشند. char(n) ثابت است و سایز آن یکبار توسط شما مشخص می‌شود و حافظه اشغال شده توسط آن، بستگی به همین سایز دارد. اگر سایزی که شما انتخاب کرده‌اید ۱۰۰ باشد و رشته ای که ذخیره می‌کنید دارای ۳ کاراکتر باشد در اینصورت char(n) باز هم مقدار حافظه ای معادل ۱۰۰ کاراکتر را اشغال می‌کند. varchar(n) کمی متفاوت است و فقط مقدار حافظه ای را که لازم دارد اشغال می‌کند. به این نکته هم توجه کنید که کاراکترهای رشته در داخل کوتیشن محصور هستند. برای مشاهده لیست کامل داده‌های SQL به لینک زیر مراجعه کنید:

```
http://www.w3schools.com/sql/sql_datatypes.asp
```

دستورات SQL

SQL دارای دستوراتی برای کار با بانک اطلاعاتی مانند ایجاد بانک، ایجاد و حذف جداول و کار با داده‌های ذخیره شده در بانک می‌باشد که در جدول زیر به مهم‌ترین آنها اشاره شده است:

کاربرد	نحوه استفاده	دستور
برای اضافه کردن، حذف کردن یا تغییر ستون‌ها در	ALTER TABLE table_name ADD column_name datatype or ALTER TABLE table_name DROP COLUMN column_name	ALTER TABLE

		جدول موجود استفاده می‌شود.
CREATE DATABASE	CREATE DATABASE database_name	یک دیتابیس جدید ایجاد می‌کند.
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, column_name3 data_type, …)	یک جدول جدید ایجاد می‌کند.
DELETE	DELETE FROM table_name WHERE some_column=some_value or DELETE FROM table_name (Note: Deletes the entire table!!) DELETE * FROM table_name (Note: Deletes the entire table!!)	برای حذف یک رکورد یا همه رکوردهای جدول به کار می‌رود.
DROP DATABASE	DROP DATABASE database_name	برای حذف دیتابیس به کار می‌رود.
DROP TABLE	DROP TABLE table_name	برای حذف جدول به کار می‌رود.
INSERT INTO	INSERT INTO table_name VALUES (value1, value2, value3,….) *or* INSERT INTO table_name (column1, column2, column3,…) VALUES (value1, value2, value3,….)	برای اضافه کردن یک یا چند رکورد به جدول به کار می‌رود.
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern	به همراه دستور WHERE برای پیدا کردن یک الگوی

		خاص در یک ستون استفاده می‌شود.
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]	برای مرتب کردن نتیجه پرس و جو بر اساس ستون مشخص شده به صورت صعودی یا نزولی به کار می‌رود.
SELECT	SELECT column_name(s) FROM table_name	برای استخراج داده‌های ستون‌های خاصی از جدول به کار می‌رود.
SELECT *	SELECT * FROM table_name	برای استخراج داده‌های یک جدول به کار می‌رود.
UPDATE	UPDATE table_name SET column1=value, column2=value,… WHERE some_column=some_value	برای به روز رسانی رکوردهای موجود در یک جدول استفاده می‌شود.
WHERE	SELECT column_name(s) FROM table_name WHERE column_name operator value	برای استخراج رکوردهایی که در شرط خاصی صدق می‌کنند کاربرد دارد.

SQL به تنهایی زبان پیشرفته و بزرگی است. بیشتر از این در مورد آن توضیح نمی‌دهیم. برای مشاهده لیست کامل دستورات SQL یه لینک زیر مراجعه کنید :

```
http://www.w3schools.com/sql/sql_quickref.asp
```

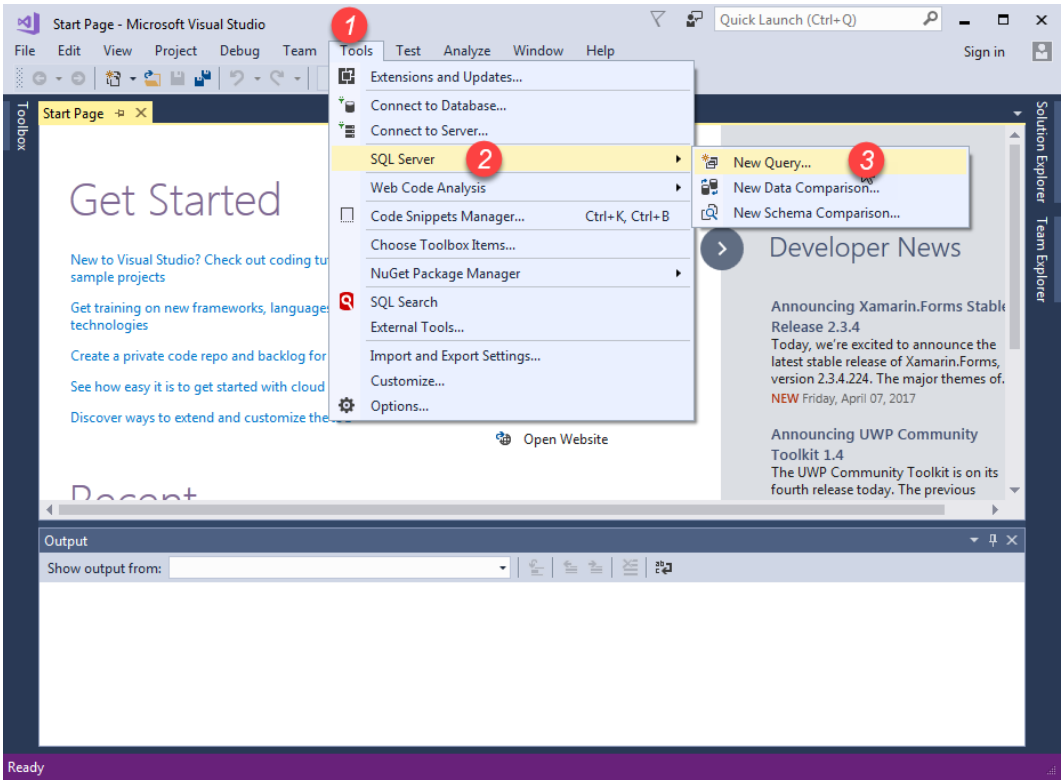
حال که با مفاهیم پایه ای آن آشنا شدید، در درس بعد با کاربرد این دستورات به صورت عملی آشنا می‌شوید.

ایجاد جدول و دیتابیس با استفاده از ویژوال استودیو

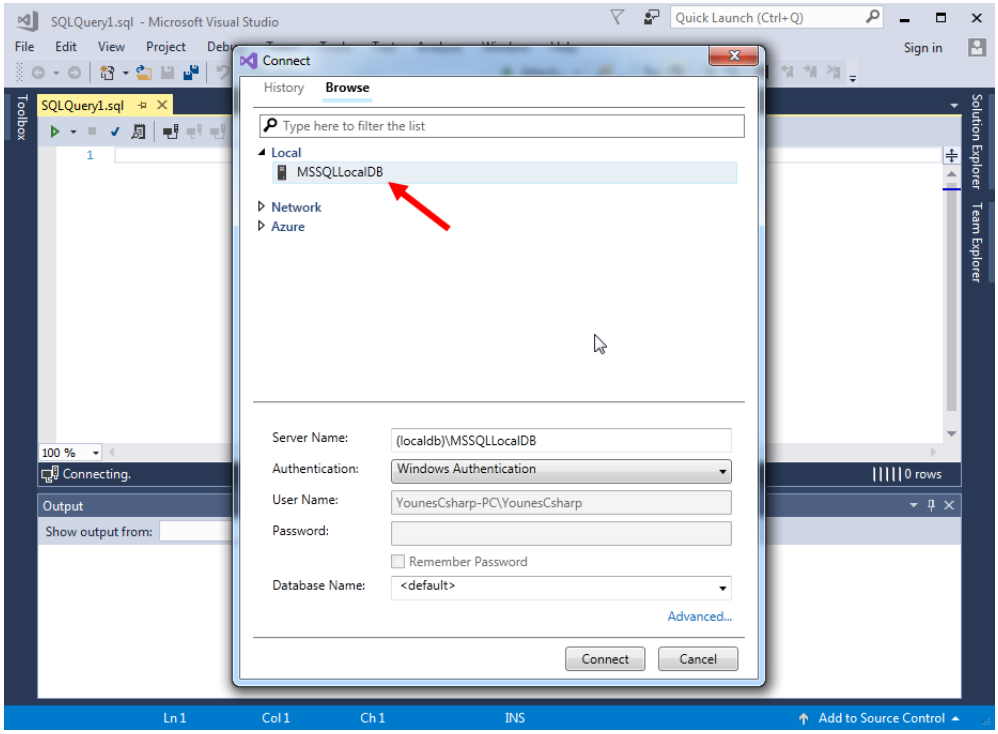
در این درس می خواهیم یک دیتابیس که شامل یک جدول و چندین رکورد است را ایجاد کنیم. قصد داریم که ایجاد دیتابیس و جدول را از دو را به شما آموزش دهیم. اولین روش استفاده از کدنویسی و دیگری استفاده از ابزارهای ویژوال استودیو.

ایجاد دیتابیس و جدول با استفاده از کدنویسی

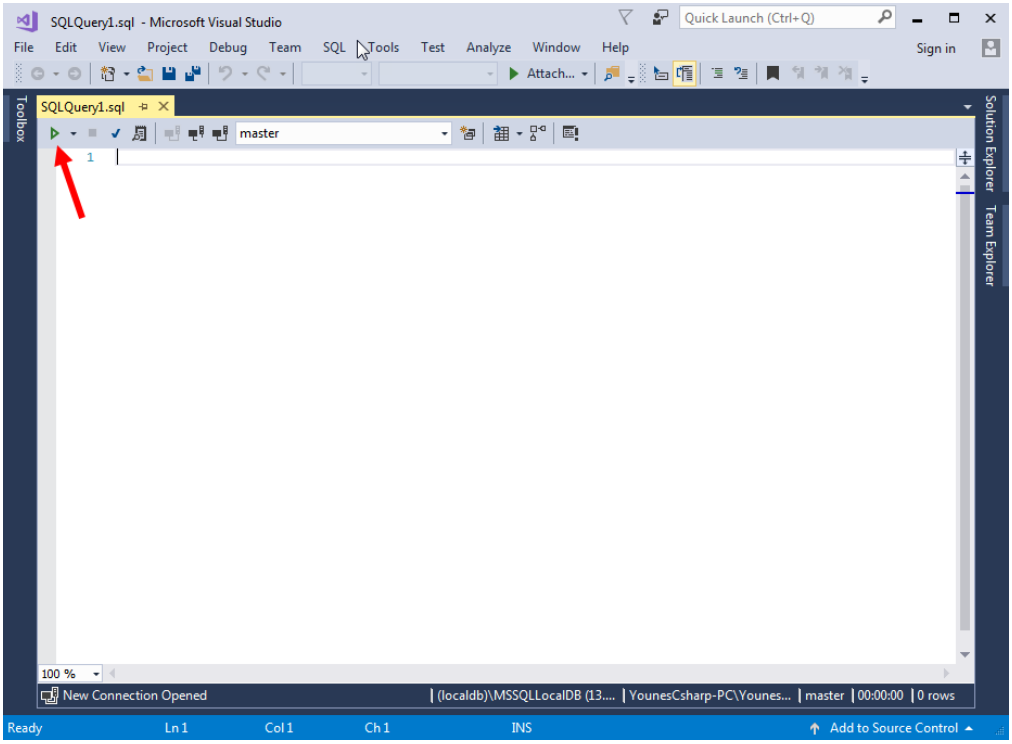
برای ایجاد دیتابیس و جدول به روش کدنویسی ابتدا ویژوال استودیو را باز کرده و سپس به مسیر زیر بروید :



با کلیک بر روی New Query صفحه ای به صورت زیر نمایش داده می شود که در آن شما باید سرور خود را انتخاب کنید :



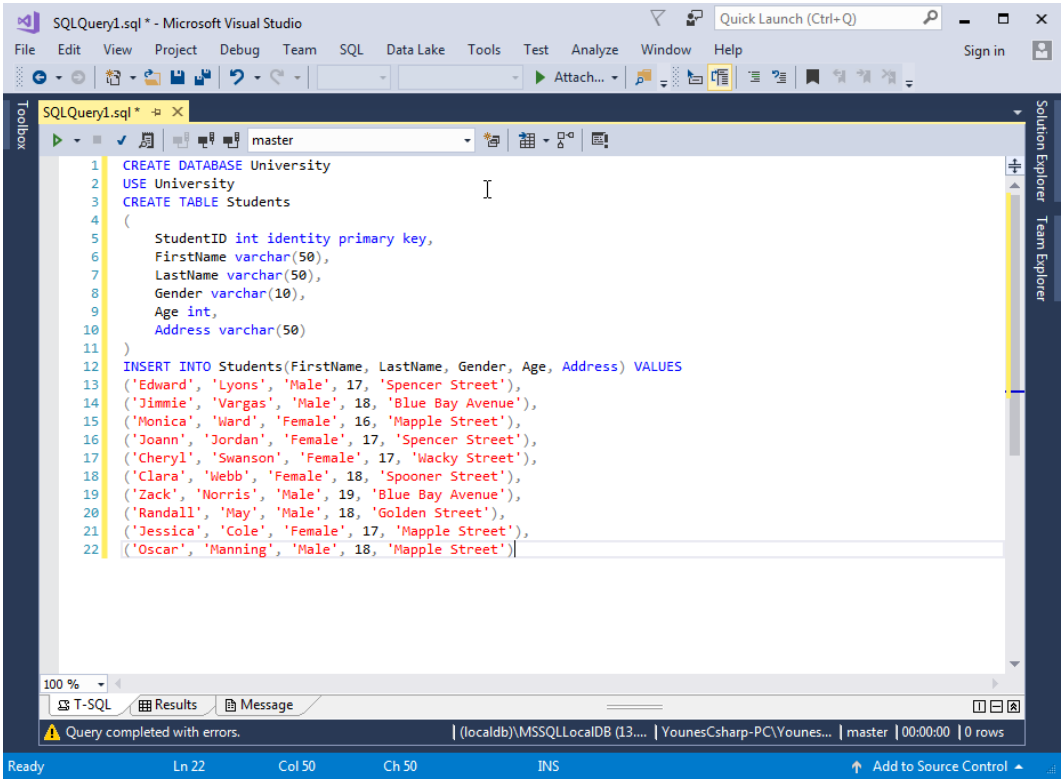
با کلیک بر روی دکمه `Connect` در شکل بالا صفحه ای که قرار است شما کدهای خود را در داخل آن بنویسید ظاهر می شود. توجه کنید که همه کدهای این درس در این بخش نوشته می شوند و شما بعد از نوشتن هر کد باید بر روی فلش سبز رنگ کلیک کنید :



حال می خواهیم یک دیتابیس با نام University که شامل جدول Students با ده رکورد است، را ایجاد کنیم. برای این کار دستورات SQL زیر را کپی:

```
CREATE DATABASE University
USE University
CREATE TABLE Students
(
   StudentID int identity primary key,
   FirstName varchar(50),
   LastName varchar(50),
   Gender varchar(10),
   Age int,
   Address varchar(50)
)
INSERT INTO Students(FirstName, LastName, Gender, Age, Address) VALUES
('Edward', 'Lyons', 'Male', 17, 'Spencer Street'),
('Jimmie', 'Vargas', 'Male', 18, 'Blue Bay Avenue'),
('Monica', 'Ward', 'Female', 16, 'Mapple Street'),
('Joann', 'Jordan', 'Female', 17, 'Spencer Street'),
('Cheryl', 'Swanson', 'Female', 17, 'Wacky Street'),
('Clara', 'Webb', 'Female', 18, 'Spooner Street'),
('Zack', 'Norris', 'Male', 19, 'Blue Bay Avenue'),
('Randall', 'May', 'Male', 18, 'Golden Street'),
('Jessica', 'Cole', 'Female', 17, 'Mapple Street'),
('Oscar', 'Manning', 'Male', 18, 'Mapple Street')
```

و در محیط ویژوال استودیو Paste کنید:



حال بهتر است در مورد کدهای بالا توضیحاتی ارائه دهیم. در خط ۱ با استفاده از دستور زیر یک دیتابیس به نام University ایجاد کرده ایم:

```
CREATE DATABASE University
```

برای ایجاد جدول student که شامل جزییاتی در مورد چندین دانش آموز است ابتدا باید از ایجاد دیتابیس مطمئن شوید. این کار ار در خط ۲ انجام داده ایم:

```
USE University
```

معنای دستور بالا این است که ما می خواهیم تغییراتی در دیتابیسی به نام Universityبدهیم.در خطوط ۱۱-۳ جدول Students را ایجاد کرده ایم:

```
CREATE TABLE Students
(
   StudentID int identity primary key,
   FirstName varchar(50),
   LastName varchar(50),
   Gender varchar(10),
   Age int,
   Address varchar(50)
)
```

جدول ایجاد شده دارای ۶ ستون می باشد. ستون اول StudentID است، که آن را به عنوان کلید اصلی و از نوع int تعریف کرده ایم. دیگر ستون های آن به ترتیب عبارتند از FirstName، LastName، Gender، Age و Address که نوع آنها مشخص شده است. با اجرای دستور فوق جدول ایجاد می شود. نکته ای که در کد بالا وجود دارد و بهتر است که در همین جا به آن اشاره کنیم مربوط به خط ۳ و کلمات identity و primary key می باشد.

- primary key، مقداری است که یک سطر یا رکورد را نشان می دهد. این مقدار باید برای هر رکورد، یکتا باشد و در رکورد دیگری عین همین مقدار نباید وجود داشته باشد. یکی از روش های ایجاد کلید اصلی استفاده از یک فیلد شمارشی به عنوان کلید اصلی است. به هر رکورد یک فیلد شمارشی اختصاص دهید که با اضافه شدن هر رکورد یک مقدار به آن اضافه شود. بدین معنا که اگر شما یک رکورد به دیتابیس اضافه کردید عدد ۱، به ازا دومین رکورد عدد ۲ و... به فیلد شمارشی اضافه شود.
- identity، فیلدی است که به طور خودکار هنگام اضافه شدن یک رکورد جدید، مقدار می گیرد(معمولا یک واحد به آن اضافه می شود). به عنوان مثال اگر یک رکورد جدید را اضافه کنیم، به طور خودکار مقدار ۱ به آن اختصاص داده می شود، و در هر بار اضافه شدن یک رکورد جدید این مقدار نیز یک واحد اضافه می شود. می توان یک کلید اصلی را به عنوان فیلد شناسه تعریف کرد.

حال که جدول مان را ایجاد کرده ایم اجازه دهید اطلاعات مربوط به چند دانش آموز را به آن اضافه کنیم.قرار است که جدول ما به صورت زیر باشد:

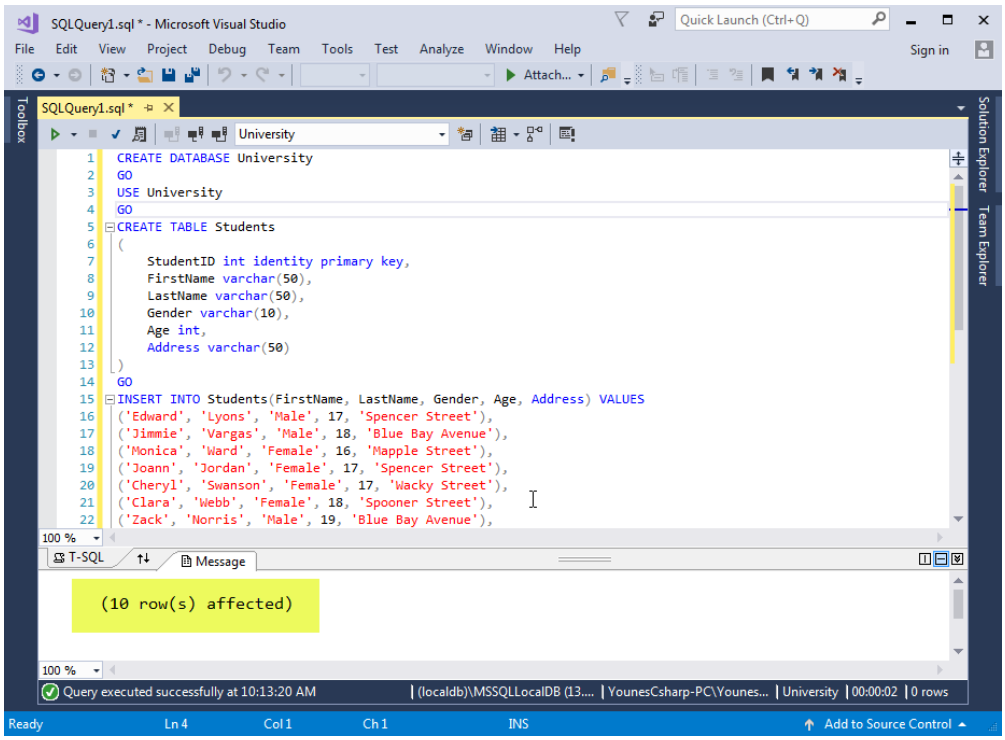
StudentID	FirstName	LastName	Gender	Age	Address
1	Edward	Lyons	Male	17	Spencer Street
2	Jimmie	Vargas	Male	18	Blue Bay Avenue
3	Monica	Ward	Female	16	Mapple Street

4	Joann	Jordan	Female	17	Spencer Street
5	Cheryl	Swanson	Female	17	Wacky Street
6	Clara	Webb	Female	18	Spooner Street
7	Zack	Norris	Male	19	Blue Bay Avenue
8	Randall	May	Male	18	Golden Street
9	Jessica	Cole	Female	17	Mapple Street
10	Oscar	Manning	Male	18	Mapple Street

در نتیجه از کد زیر برای وارد کرده اطلاعات در جدول استفاده کرده ایم:

```
INSERT INTO Students(FirstName, LastName, Gender, Age, Address) VALUES
('Edward', 'Lyons', 'Male', 17, 'Spencer Street'),
('Jimmie', 'Vargas', 'Male', 18, 'Blue Bay Avenue'),
('Monica', 'Ward', 'Female', 16, 'Mapple Street'),
('Joann', 'Jordan', 'Female', 17, 'Spencer Street'),
('Cheryl', 'Swanson', 'Female', 17, 'Wacky Street'),
('Clara', 'Webb', 'Female', 18, 'Spooner Street'),
('Zack', 'Norris', 'Male', 19, 'Blue Bay Avenue'),
('Randall', 'May', 'Male', 18, 'Golden Street'),
('Jessica', 'Cole', 'Female', 17, 'Mapple Street'),
('Oscar', 'Manning', 'Male', 18, 'Mapple Street')
```

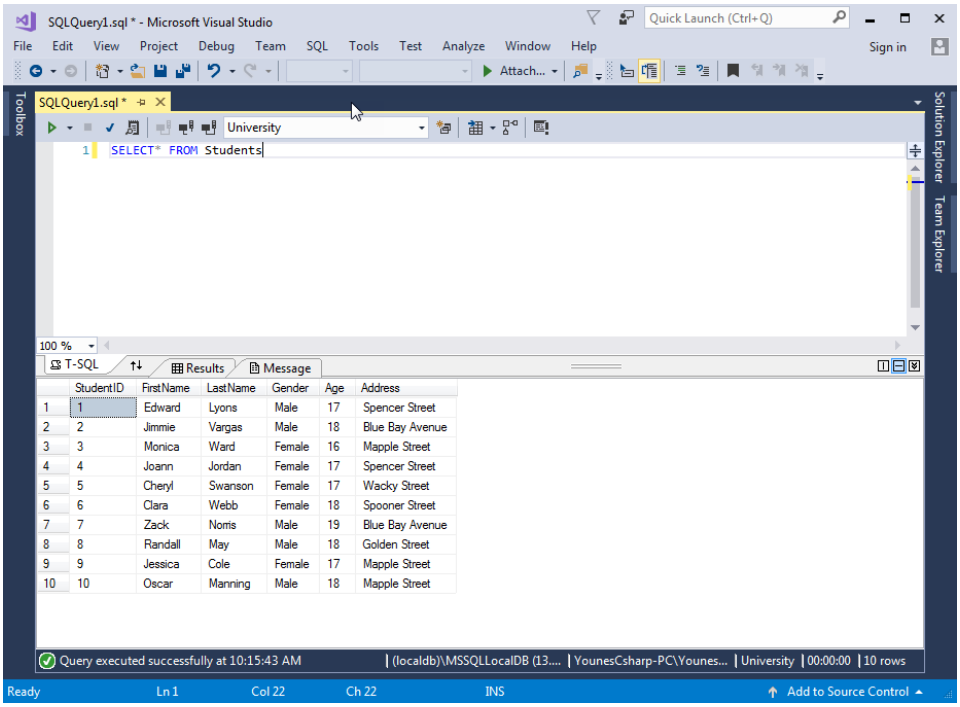
کد بالا یک نسخه اصلاح شده از دستور INSERT INTO است که به ما اجازه وارد کردن چندین رکورد را به صورت یکجا می دهد. هر رکورد در داخل پرانتز قرار دارد و رکوردها به وسیله کاما از هم جدا شده اند. بعد از اجرای دستورات SQL ، جدول Students باید دارای ۱۰ رکورد باشد. اگر همه چیز به درستی انجام شود، یک پیغام نشان داده می شود که نشان دهنده تعداد رکوردهایی است که دستکاری (اضافه) شده اند.



حال می خواهیم با توجه به مطالبی که در درس قبل یاد گرفتیم، عملیات مختلفی بر روی بانک انجام دهیم. دستور INSERT را که در بالا برای وارد کردن اطلاعات انجام داده ایم. حال برای نمایش اطلاعات می خواهیم از دستور SELECTاستفاده کنیم. کدهای قبلی را پاک کرده و کد زیر را نوشته و بر روی فلش سبز رنگ کلیک کنید:

```
SELECT * FROM Students
```

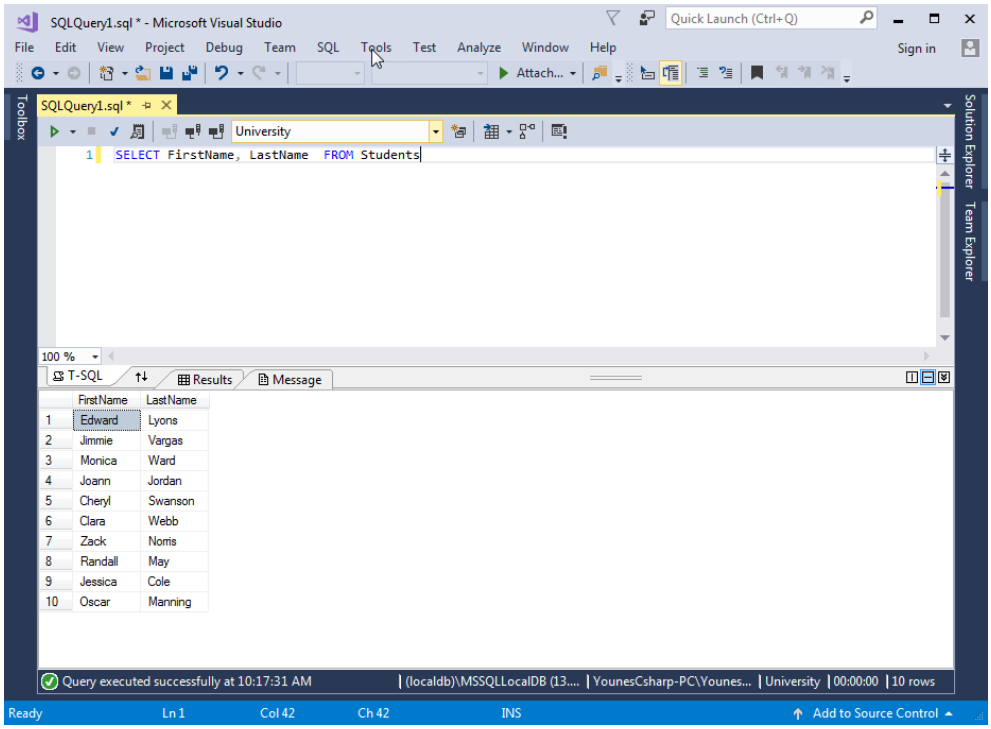
کد بالا تمامی داده های جدول Students را استخراج کرده و در خروجی نمایش می دهد:



اگر بخواهید اطلاعات یک یا چند ستون خاص را نمایش دهید کافیست از کد زیر استفاده کنید:

```
SELECT FirstName, LastName  FROM Students
```

کد بالا فقط اطلاعات مربوط به ستون های `FirstName` و `LastName` را نمایش می دهد:



سایر کدهای زیر را مانند روش بالا تست کنید. حال فرض کنید که می خواهیم نام خانوادگی و سن شخصی که اسمش Edward است را از جدول استخراج کنیم. برای این کار از دستور WHERE به صورت زیر استفاده می کنیم:

```
SELECT LastName, Age FROM Students WHERE FirstName = 'Edward';
```

حال می خواهیم اشخاصی که StudentID آنها کمتر از ۵ است را استخراج کرده و به صورت نزولی مرتب کنیم. برای این کار به صورت زیر عمل می کنیم:

```
SELECT StudentID, FirstName, LastName FROM Students WHERE StudentID < 5 ORDER BY studentID DESC;
```

کد بالا ۴ نفر اول را استخراج کرده و بر اساس ستون StudentID از بزرگ به کوچک مرتب می کند. فرض کنید می خواهید تمامی نام هایی که در داخل آنها ar وجود دارد را استخراج کنید. در این صورت از عبارت LIKE به صورت زیر استفاده کنید:

```
SELECT FirstName FROM Students WHERE FirstName LIKE '%ar%'
```

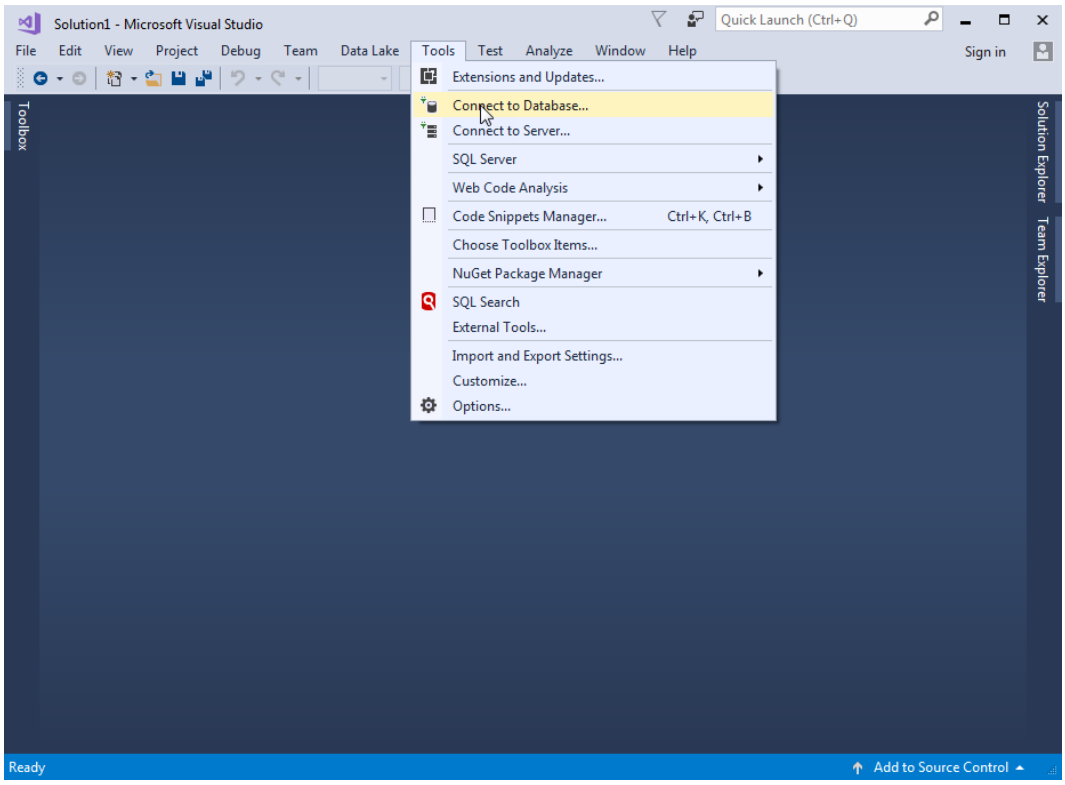
کد بالا تمامی نام هایی که در آنها دو حرف a و r پشت سر هم آمده اند را استخراجی می کند. اگر بخواهیم مثلا سن شخصی به نام Edward را از ۱۷ به ۲۰ تغییر دهیم می توانیم از دستور UPDATE استفاده کنیم:

```
UPDATE Students SET Age = 20 WHERE StudentID = 1
```

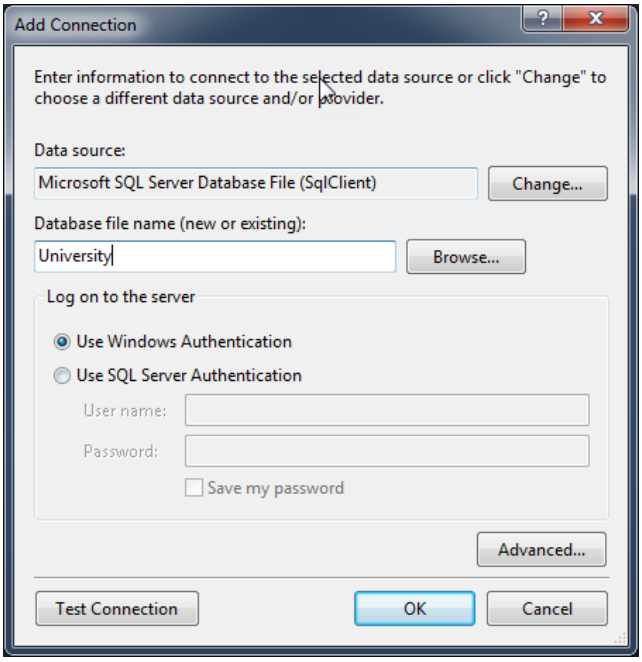
در دستور بالا گفته ایم که سن شخصی از جدول Students را که StudentID آن برابر ۱ است را به ۲۰ تغییر بده. حال اگر با استفاده از یک دستور SELECT همه داده ها را نمایش دهیم مشاهده می کنید که سن Edward به ۲۰ تغییر کرده است. همانطور که مشاهده می کنید استفاده از دستورات SQL بسیار آسان است. شما می توانید بقیه دستورات را که در درس قبل آموختید را به صورت بالا امتحان کنید.

ایجاد دیتابیس و جدول با استفاده از ابزارهای ویژوال استودیو

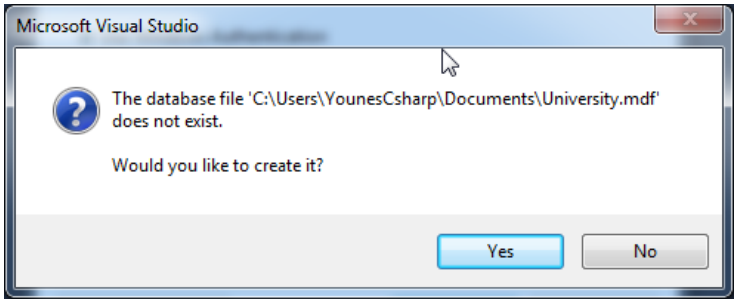
برای ایجاد دیتابیس بدون کدنویسی به مسیر زیر بروید:



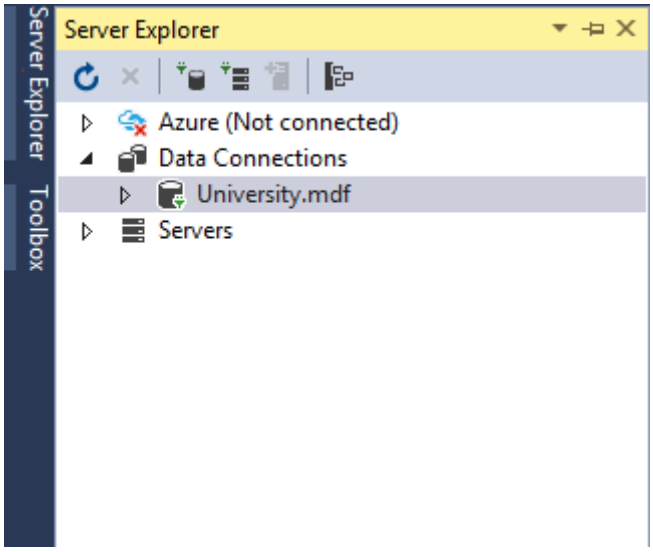
با کلیک بر روی گزینه `Connect to database` صفحه ای به صورت زیر باز می شود که از شما می خواهد مسیر دیتابیس تان را وارد کنید. اگر دیتابیسی از قبل وجود نداشته باشد و شما بخواهید آن را ایجاد کنید مستقیما در کادر `Databse file name` نام دیتابیسی که می خواهید ایجاد کنید را می نویسید:



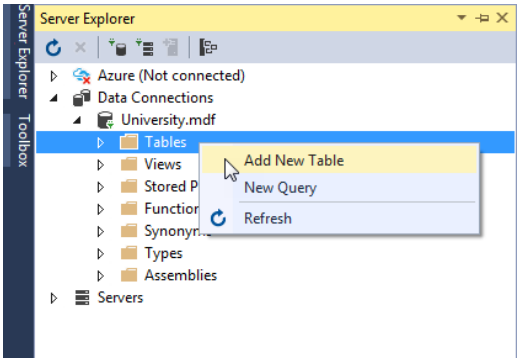
با کلیک بر روی OK صفحه ای باز می شود که می گوید دیتابیس از قبل وجود ندارد آیا می خواهید آن را ایجاد کنید :



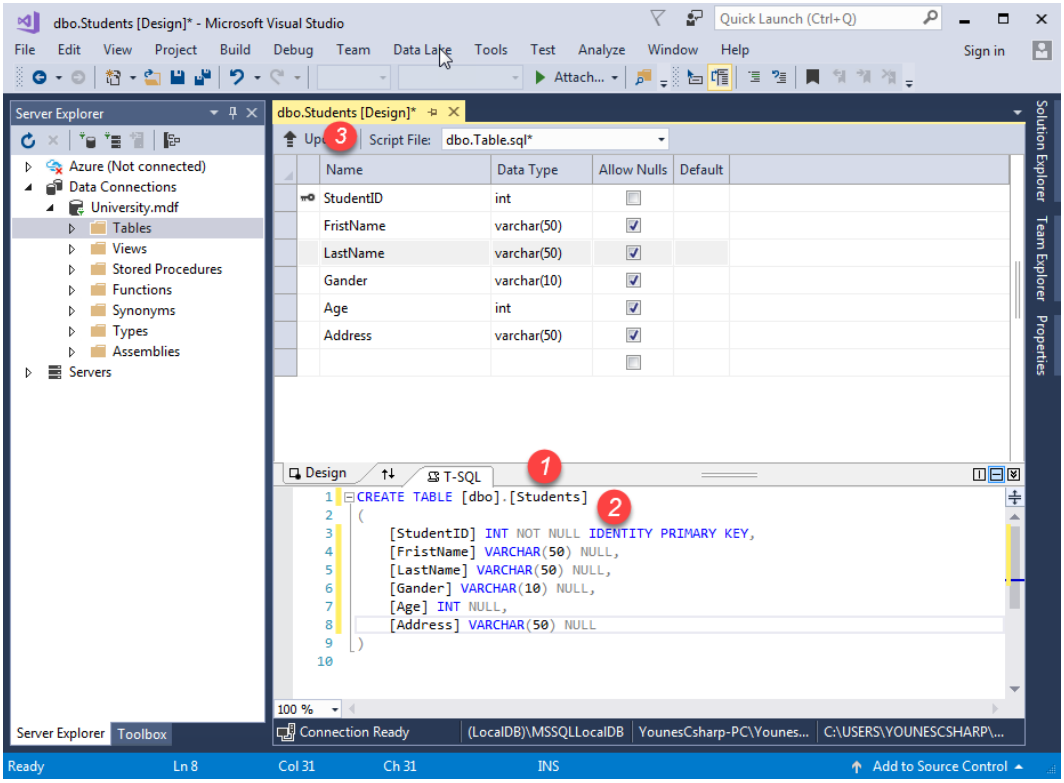
با کلیک بر روی دکمه Yes ، دیتابیس ایجاد می شود. حال اگر به قسمت Server Explorer نگاه کنید، مشاهده می کنید که دیتابیس ایجاد شده است :



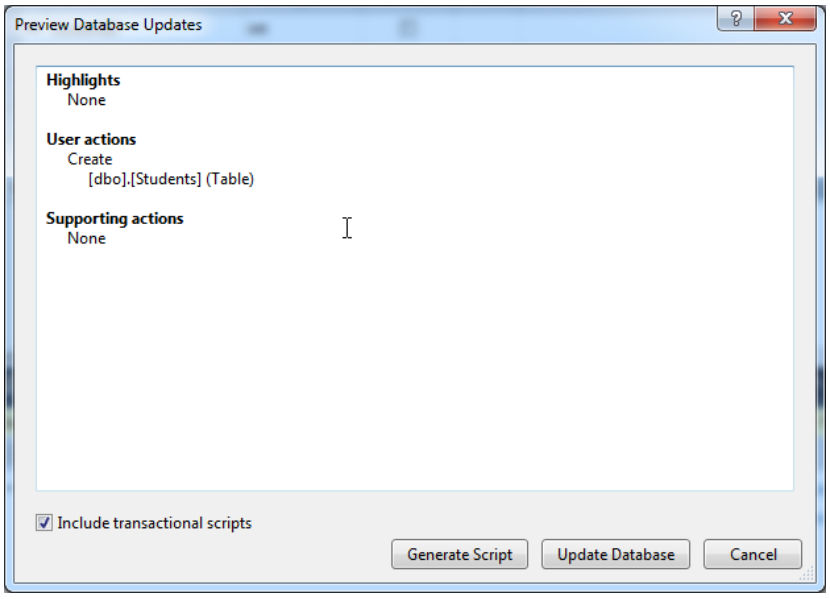
برای ایجاد جدول ابتدا بر روی فلش کوچک کنار نام دیتابیس کلیک کنید تا زیر پوشه های آن نمایان شوند. حال بر روی پوشه Tables راست کلیک کرده و گزینه Add New Table را بفشارید :



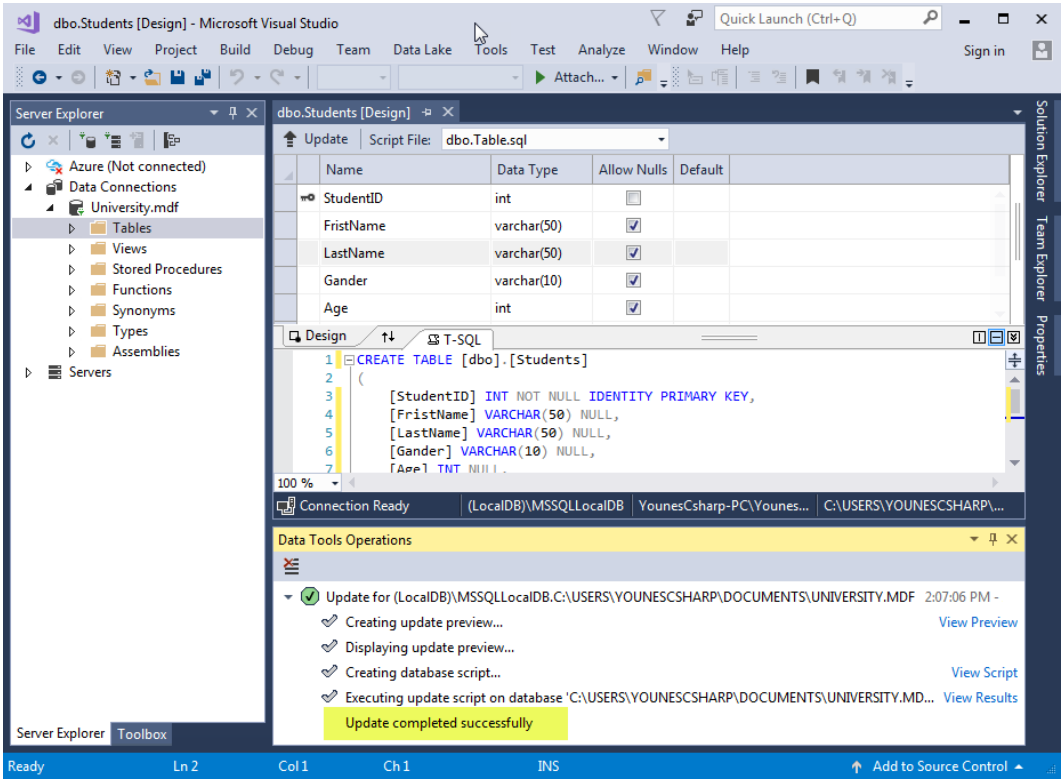
با کلیک بر روی این گزینه صفحه ای به صورت زیر باز می شود که همان صفحه ایجاد جدول است. در این صفحه تغییرات زیر را اعمال کنید :



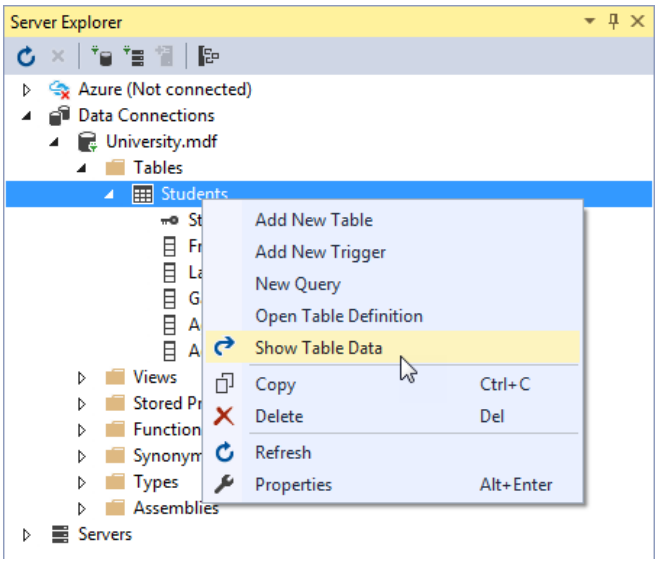
با کلیک بر روی گزینه `Update` (عدد ۳)، صفحه ای به صورت زیر باز می شود:



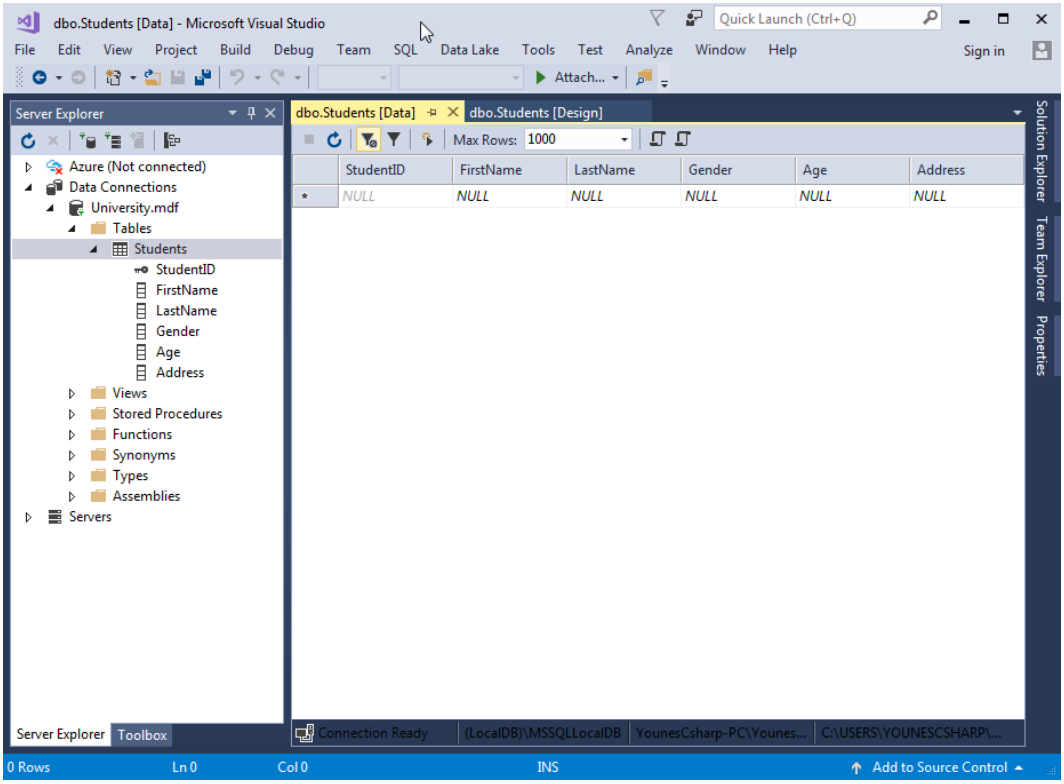
در صفحه بالا بر روی گزینه `Update Database` کلیک کنید. با کلیک بر روی این گزینه صفحه ای به صورت زیر باز می شود. اگر همه چی به خوبی پیش برود پیغام `Update Complated Successfully` نمایش داده می شود که نشان از ایجاد موفقیت آمیز جدول دارد:

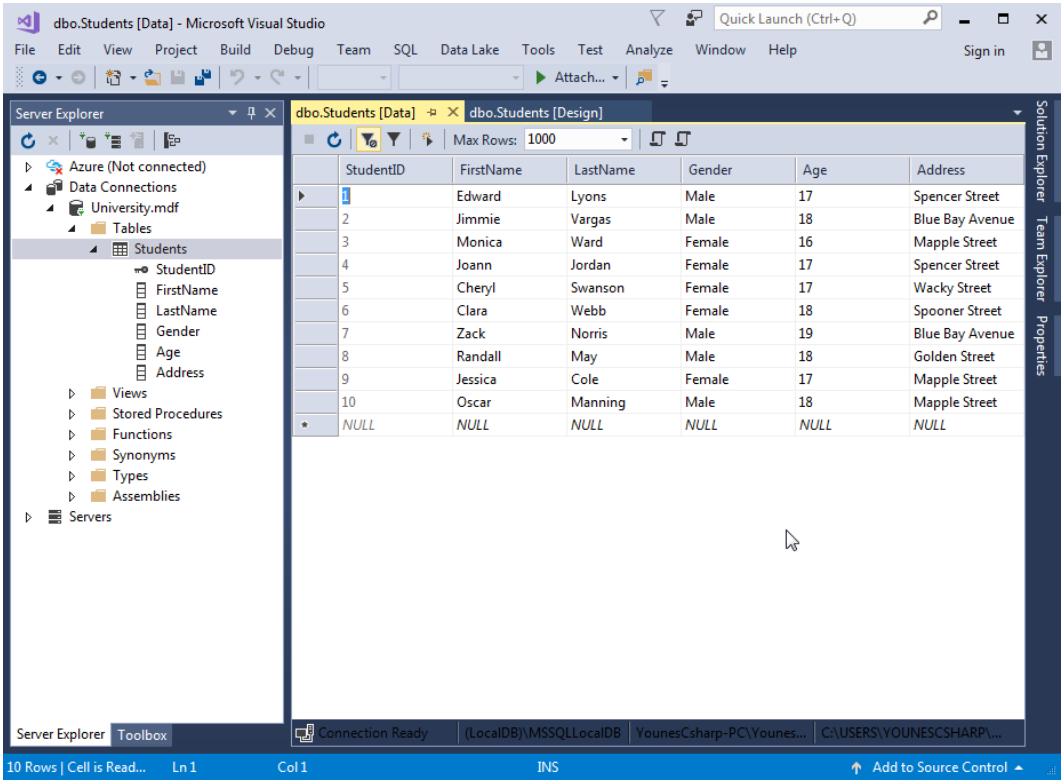


اگر به زیر پوشه های `Tables` نگاهی بیاندازید می بینید که جدول `Students` هم ایجاد شده است. حال نوبت به وارد کردن داده ها در جدول می رسد. برا این منظور بر روی نام جدول راست کلیک و سپس بر روی گزینه `Show Table Data` کلیک کنید:



با کلیک بر روی این گزینه صفحه زیر باز می شود که شما می توانید اطلاعات خود را بر اساس ستون ها وارد نمایید:


dbo.Students [Data] - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team SQL Data Lake Tools Test Analyze Window Help
Sign in
Attach...
Server Explorer
Azure (Not connected)
Data Connections
University.mdf
Tables
Students
StudentID
FirstName
LastName
Gender
Age
Address
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Servers
dbo.Students [Data]
dbo.Students [Design]
Max Rows: 1000
StudentID FirstName LastName Gender Age Address
NULL NULL NULL NULL NULL NULL
Server Explorer Toolbox
Connection Ready
(LocalDB)\MSSQLLocalDB
YounesCsharp-PC\Younes...
C:\USERS\YOUNESCSHARP\...
Solution Explorer
Team Explorer
Properties
0 Rows
Ln 0
Col 0
INS
Add to Source Control

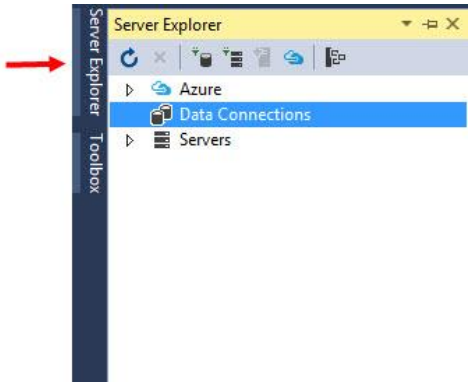

dbo.Students [Data] - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team SQL Data Lake Tools Test Analyze Window Help
Sign in
Attach...
Server Explorer
Azure (Not connected)
Data Connections
University.mdf
Tables
Students
StudentID
FirstName
LastName
Gender
Age
Address
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Servers
dbo.Students [Data]
dbo.Students [Design]
Max Rows: 1000
StudentID FirstName LastName Gender Age Address
1 Edward Lyons Male 17 Spencer Street
2 Jimmie Vargas Male 18 Blue Bay Avenue
3 Monica Ward Female 16 Mapple Street
4 Joann Jordan Female 17 Spencer Street
5 Cheryl Swanson Female 17 Wacky Street
6 Clara Webb Female 18 Spooner Street
7 Zack Norris Male 19 Blue Bay Avenue
8 Randall May Male 18 Golden Street
9 Jessica Cole Female 17 Mapple Street
10 Oscar Manning Male 18 Mapple Street
NULL NULL NULL NULL NULL NULL
Server Explorer Toolbox
Connection Ready
(LocalDB)\MSSQLLocalDB
YounesCsharp-PC\Younes...
C:\USERS\YOUNESCSHARP\...
Solution Explorer
Team Explorer
Properties
10 Rows | Cell is Read...
Ln 1
Col 1
INS
Add to Source Control

اتصال به دیتابیس با استفاده از ابزارهای ویژوال استودیو

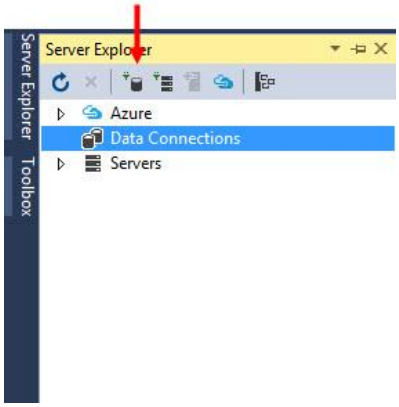
قبل از ورود به مبحث ADO.NET، اجازه بدهید ابتدا به برخی از ابزارهای ویژوال استودیو نگاهی بیندازیم. مثال زیر یک راه برای اتصال به دیتابیس بدون استفاده از کدنویسی را نشان می‌دهد.

ایجاد یک اتصال به دیتابیس

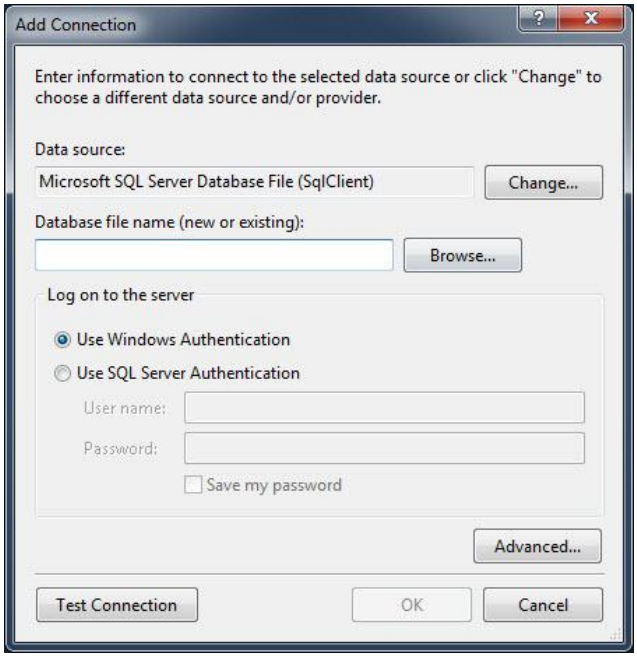
Visual Studio را باز و یک برنامه ویندوزی جدید ایجاد کنید. نام برنامه‌تان را DatabaseConnection بگذارید. سپس بر روی تب Server Explorer که در حالت پیشفرض در تب سمت چپ پنجره ویژوال استودیو (شکل زیر) قرار دارد کلیک کنید.



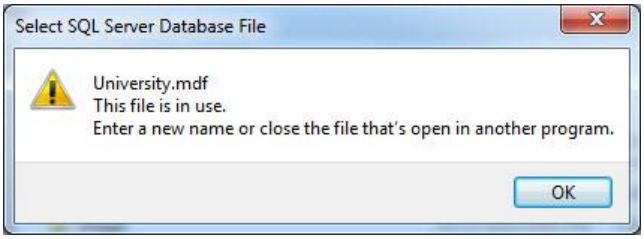
اگر Server Explorer را پیدا نکردید می‌توانید از مسیر View > Server Explorer برای یافتن آن اقدام کنید. بر روی آیکون Connect to Database واقع در Database/Server Explorer کلیک کنید (شکل زیر):



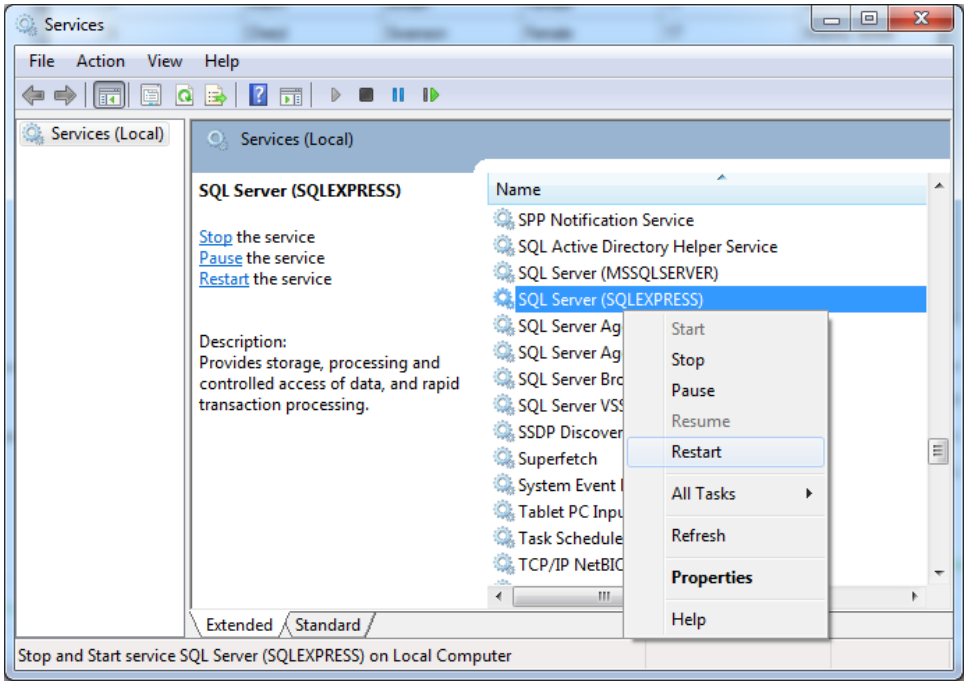
با کلیک بر روی این آیکون پنجره زیر نمایش داده می‌شود.



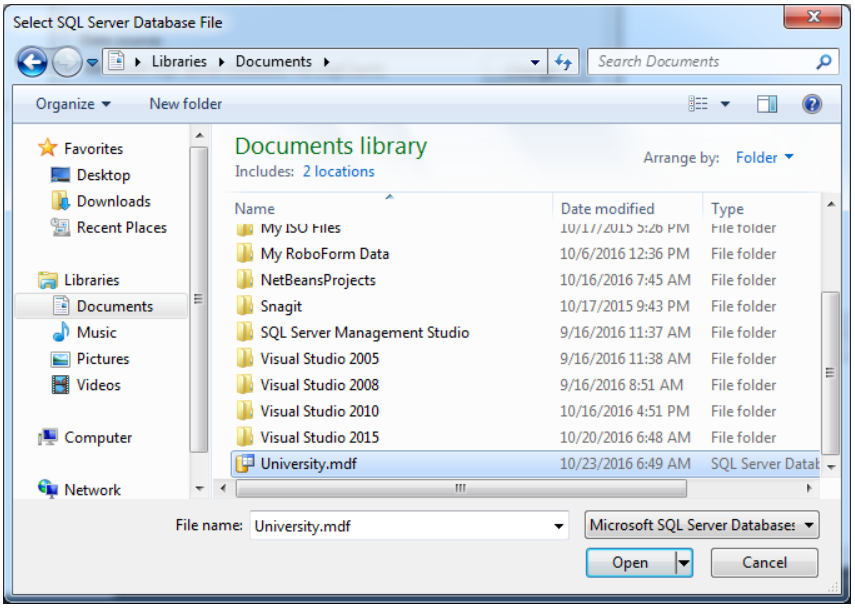
مطمئن شوید که در داخل کادر اول Data source جمله ی Microsoft SQL Server Database Fileنوشته شده است (شکل بالا). اگر نبود می توانید با کلیک بر روی دکمه Change منبع داده مناسب را انتخاب کنید. همچنین لازم است در قسمت Databae file nameنام دیتابیسی که قبلا ایجاد کرده اید را وارد نمایید .برای این کار بر روی دکمه Browse تا کادر محاوره ای open dialog ظاهر شود .در حالت پیشفرض دیتابیس های ایجاد شده توسط ویژوال استودیو در مسیر C:\Users\YourName\Documentsقرار دارند. فایل University.mdf (که در درس قبل ایجاد کرده اید) را یافته و آن را انتخاب کنید. اگر هم درس های قبل را انجام نداده اید و دیتابیس را در اختیار ندارید، بهتر است به درس قبل برگشته و آن را ایجاد کنید و سپس با استفاده از دکمه Browse ی که در بالا اشاره شد آن را انتخاب کرده و سپس بر روی دکمه Open کلیک کنید. اگر پیغام خطایی مشاهده کردید بدانید که فایل مذکور به وسیله یک برنامه دیگر در حال اجراست.



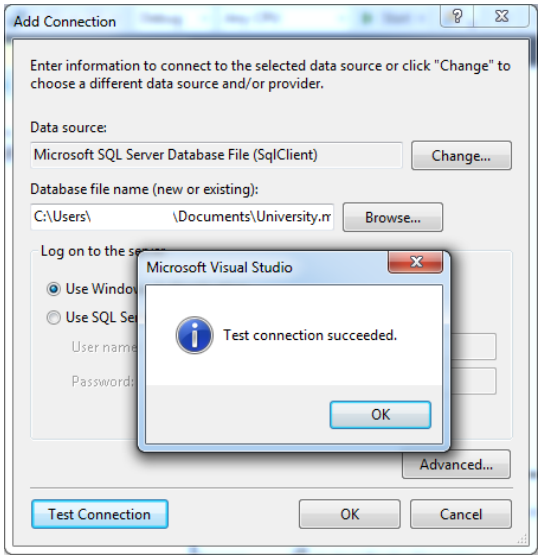
برنامه Services را به وسیله کلیک بر روی منوی Start و نوشتن کلمه Servises در جعبه جستجو (search) اجرا کنید. از گزینه‌های ظاهر شده بر روی view Local Services کلیک کنید تا پنجره ای ظاهر شود. از داخل این پنجره SQL Server (SQLEXPRESS) service را یافته و بر روی آن راست کلیک کنید. گزینه Restart را انتخاب کرده تا service مجددا راه اندازی شود.



بعد از ریستارت شدن آن، می‌توانیم به عقب برگردیم و از نو فایل `University.mdf` را انتخاب کرده و سپس بر روی دکمه `Open` کلیک کنیم :

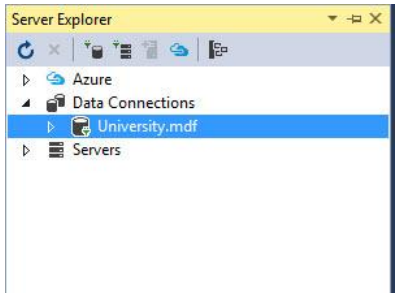


بعد از کلیک بر روی دکمه Open، دکمه `Test Connection` در پنجره `Add Connection` را کلیک کنید تا تست شود که آیا برنامه مان می‌تواند با دیتابیس ارتباط برقرار کند یا نه؟ اگر هیچ اشتباهی رخ ندهد، سپس یک پیغام موفقیت نمایش داده می‌شود.

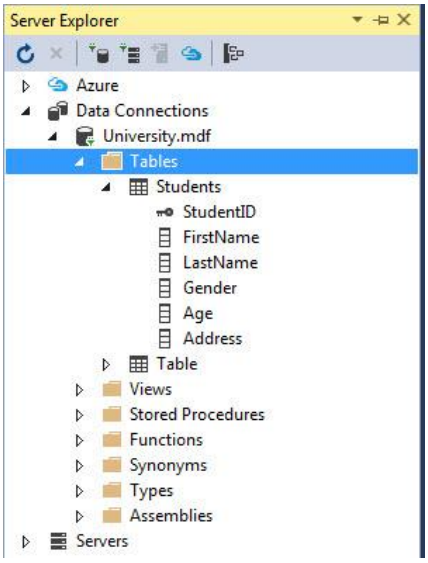


بر روی دکمه Ok کلیک کنید تا پنجره فوق بسته شود. شما همچنین می‌توانید از Authentication mode نیز استفاده کنید. می‌توانید از اعتبارسنجی ویندوز (Windows Authentication) یا اعتبارسنجی SQL Server (SQL Server Authentication) استفاده نمایید. اگر از اعتبار سنجی SQL Server (SQL Server Authentication) استفاده می‌کنید باید username و password در اختیار داشته باشید.

بر روی دکمه OK کلیک کرده تا پنجره Add Connection بسته شود و فایل دیتابیس به پنجره Server Explorer اضافه شود.

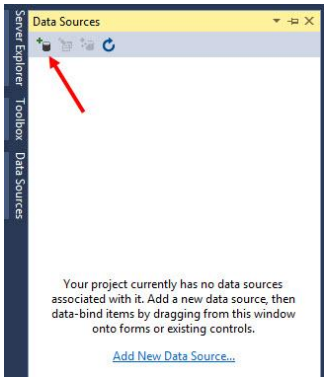


این پنجره (Server Explorer) به شما اجازه مشاهده محتویات دیتابیس را می‌دهد. اگر بر روی علامت فلش کنار نام University.mdf کلیک کنید، قسمت‌های دیگر مانند جداول و رویه‌های ذخیره شده (stored procedures) را مشاهده خواهید کرد. با کلیک بر روی علامت فلش کنار کلمه Tables جدول Students نمایش داده می‌شود و با کلیک بر روی فلش کنار این جدول، ستون‌های آن نمایش داده می‌شوند (شکل زیر).

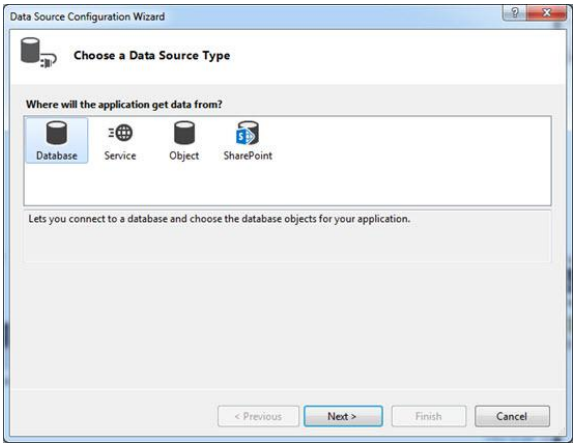


ایجاد DataSet

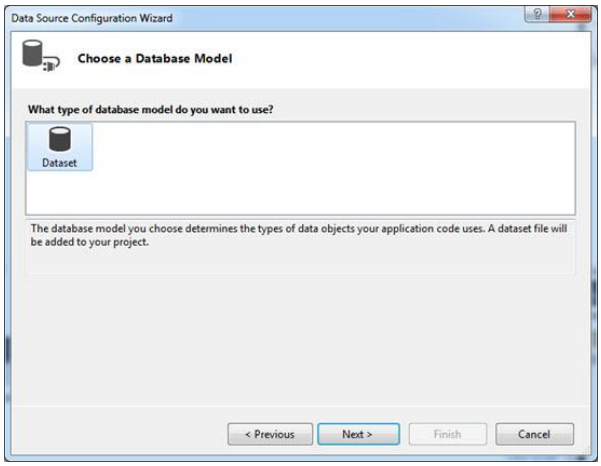
یک DataSet را می‌توان به عنوان یک دیتابیس کوچک که در حافظه رم کامپیوتر قرار دارد، در نظر گرفت. DataSet دارای یک یا چندین جدول همانند جداول موجود در پایگاه داده اصلی می‌باشد. DataSet اطلاعات لازم را از پایگاه داده اصلی می‌گیرد و آنها را در جداول خود ذخیره می‌کند. برای ایجاد یک Dataset که محتویات دیتابیسی که می‌خواهیم به آن وصل شویم را در خود نگهداری می‌کند، پنجره Data Sources استفاده می‌کنیم. اگر این پنجره را پیدا نکردید به مسیر Project > Add New Data Source بروید. در حالت پیشفرض این پنجره در سمت چپ محیط ویژوال استودیو قرار دارد.



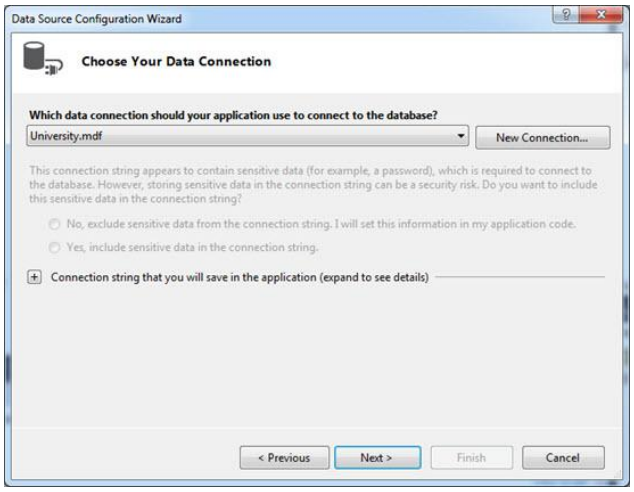
بر روی دکمه Add New Data Source کلیک کنید تا پنجره زیر با عنوان Data Source Configuration Wizard نمایش داده شود.



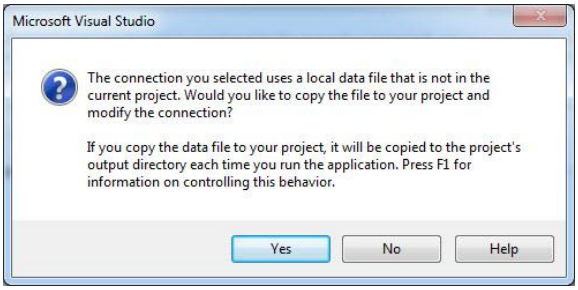
گزینه Database را انتخاب کرده و بر روی دکمه Next کلیک کنید.



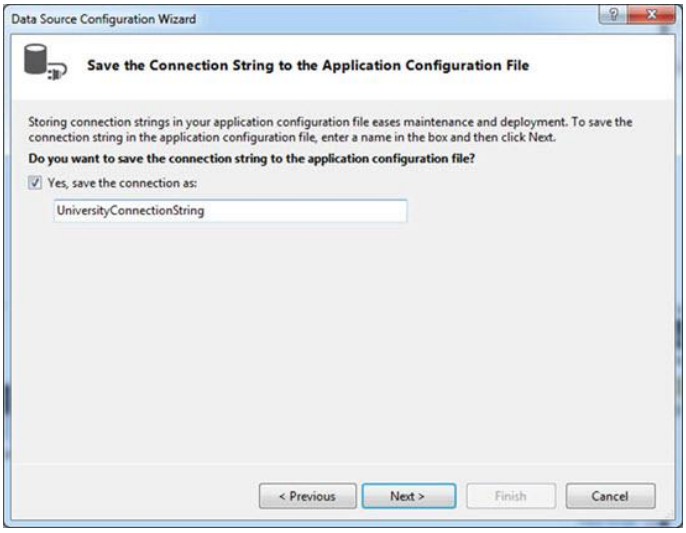
در پنجره بالا Dataset را انتخاب و بر روی دکمه Next کلیک کنید.



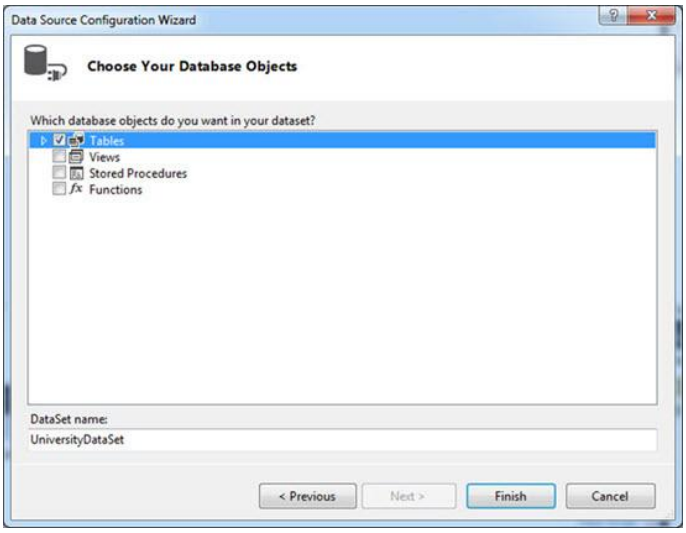
در جعبه باز شونده (combo box) پنجره Data Source Configuration Wizard، از انتخاب دیتابیس University.mdf که با استفاده از Server Explorer به آن متصل شده‌ایم مطمئن شوید. بر روی دکمه Next کلیک کنید.



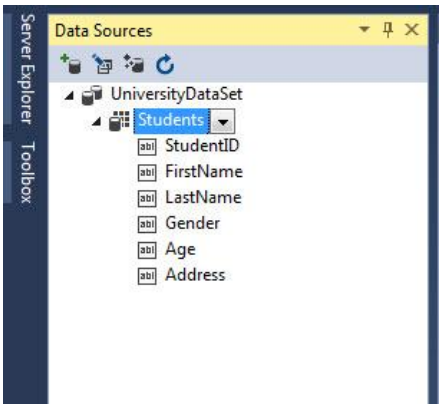
پس از زدن دکمه Next پنجره بالا نمایش داده می‌شود که دارای پیغامی است و به شما می‌گوید که لازم است فایل‌های دیتابیس در پوشه پروژه کپی شوند که با کلیک بر روی دکمه yes این کار انجام می‌شود. با نگاه کردن به پنجره Solution Explorer و مشاهده نام University.mdf متوجه می‌شوید که عمل کپی انجام شده است.



این پنجره نیز نشان دهنده رشته اتصالی (connection string) است که با استفاده از آن می‌توان به دیتابیس University وصل شویم. در باره رشته اتصال (connection string) در درس آینده توضیح می‌دهیم. با تنظیمات این پنجره کاری نداریم و بر روی دکمه Next کلیک می‌کنیم.



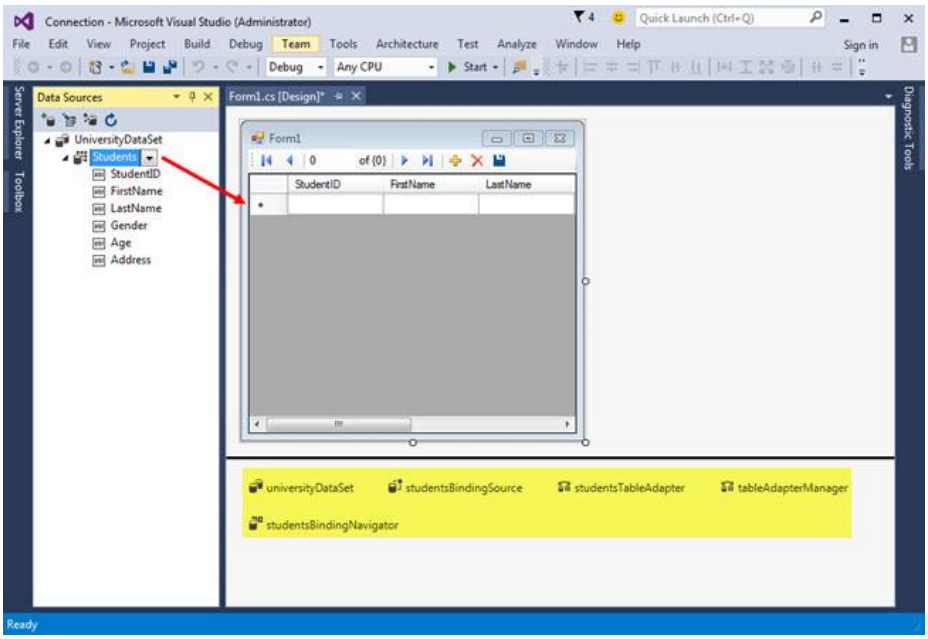
صبر کنید که محتویات دیتابیس بارگذاری شود. سپس از شما سؤال می‌شود که می‌خواهید کدام یک از قسمت‌های دیتابیس در داخل DataSet قرار بگیرند. از آنجاییکه ما فقط به جداول نیاز داریم گزینه Tables را تیک می‌زنیم. کادر DataSet name مشخص کننده نام DataSet ی است که قبلاً ایجاد کرده‌ایم. بر روی گزینه finish کلیک کرده تا DataSet ایجاد شود.



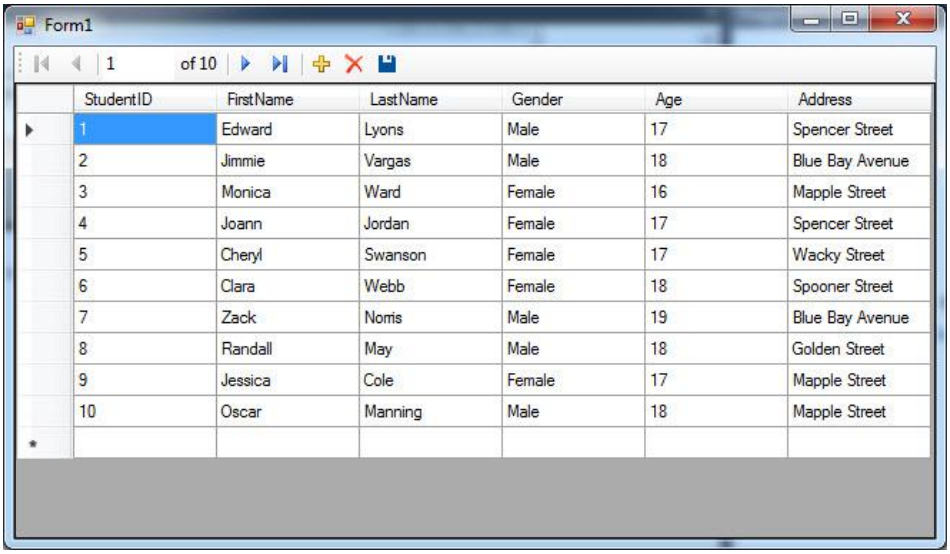
مشاهده می‌کنید که DataSet در پنجره Data Sources ایجاد می‌شود. وقتی که بر روی فلش کنار اسم DataSet کلیک کنیم، جداولی که در آن قرار دارند، نمایش داده می‌شوند. با باز کردن هر جدول هم فیلدها و ستون‌های آن نمایان می‌شوند.

نمایش داده‌های جدول با استفاده از کشیدن و انداختن(Drag and Drop)

اکنون وارد بخش هیجان انگیز آموزش می‌شویم. با DataSet مان که در پنجره Data Sources قرار دارد، می‌توانیم به راحتی یک جدول را بر روی فرم بکشیم. حتی می‌توان هر کدام از ستون‌های جدول را هم بر روی فرم کشید، ولی در حال حاضر ما همه جدول را به فرم انتقال می‌دهیم.



بعد از کشیدن جدول را بر روی فرم و رها کردن آن، ویژوال استودیو به طور خودکار یک کنترل `DataGridView` و تمام ابزارهای لازم (که با رنگ زرد در شکل بالا مشخص شده‌اند) جهت واکشی اطلاعات جدول `Student` را بر روی فرم قرار می‌دهد. `DataGridView` به شما اجازه مشاهده انواع داده‌های مختلف را در یک جدول (مثلاً یک جدول از دیتابیس یا مقادیر یک آرایه چند بعدی) می‌دهد. همانطور که مشاهده می‌کنید، تمام ستون‌های جدول `Students` به طور خودکار در `DataGridView` نمایش داده می‌شود. می‌توانید با اختصاص مقدار `Fill` به خاصیت `Dock` کنترل `DataGridView` کاری کنید که `DataGridView` تمام فضای فرم را در بر بگیرد. با استفاده از کنترل `BindingNavigator` (که شبیه نوار ابزار در بالای فرم قرار می‌گیرد) می‌توان در میان رکوردهای جدول حرکت کرده و آنها را ویرایش یا حذف کرده و یا یک رکورد جدید به جدول اضافه نمایید. با اجرای برنامه مشاهده می‌کنید که همه رکوردها در `DataGridView` نمایش داده می‌شوند. شما می‌توانید با استفاده از کنترل `BindingSourceNavigator` محتویات دیتابیس را ویرایش نمایید.



می‌توان با استفاده از دکمه‌های کنترل مذکور از در بین رکوردها حرکت کرد. علامت + به شما اجازه اضافه کردن یک رکورد جدید را می‌دهد. با اضافه شدن هر رکورد مقدار ستون StudentID به طور خودکار یک واحد اضافه می‌شود. با دو بار کلیک بر روی فیلدهای هر رکورد می‌توانید آنها را ویرایش نمایید. برای حذف یک رکورد، ابتدا آن را انتخاب کرده و سپس بر روی علامت ضربدر قرمز رنگ کلیک کنید. اگر در رکوردهای بالا تغییراتی اعمال کردید بر روی دکمه ذخیره کلیک کنید تا تغییرات به دیتابیس ارسال شود.

رشته اتصال (Connection Strings)

یک connection string یک سری از جفت‌های نام/مقدار (name/value) است که به وسیله سمیکالن از هم جدا شده‌اند و نشان دهنده تنظیماتی برای اتصال به دیتابیس می‌باشند. این تنظیمات شامل مکان دیتابیس، اعتبارسنجی، امنیت و نام دیتابیسی است که به آن متصل شده‌ایم. connection string برای منابع داده‌ای مختلف یکسان نیست. به عنوان مثال OLE DB و ODBC نیاز به تعیین یک درایور OLE DB و ODBC در رشته اتصالشان دارند. کلاس‌های کانکشن (Connection class) دارای یک خاصیت ConnectionString هستند که شما می‌توانید با استفاده از آن به دیتابیس وصل شوید. در زیر مثالی از یک رشته اتصال برای وصل شدن به یک منبع داده SQL Server Expres آمده است:

```
Data Source=.\SQLEXPRESS;Initial Catalog=University; Integrated Security=SSPI;
```

برای آگاهی بیشتر از انواع رشته اتصال به نشانی زیر مراجعه کنید:

```
http://connectionstrings.com
```

در مثال‌های زیر رشته‌های اتصالی نشان داده شده‌اند که از آنها برای اتصال به یک منبع داده SQL Server استفاده می‌شود. شما می‌توانید از طریق لینک http://connectionstrings.com برای یافتن منابع داده‌ای دیگر مانند Access اقدام نمایید. هنگام ایجاد رشته اتصال لازم است که منبع داده‌ای یا سروری که از آن استفاده می‌کنید را، مشخص کنید. برای این کار می‌توانید از پارامترهای Data Source یا Server استفاده کنید. رشته اتصال نشان داده شده در زیر، برای اتصال به یک منبع داده‌ای SQL Express مورد استفاده قرار می‌گیرد:

```
Data Source=.\SQLEXPRESS;Initial Catalog=University;Integrated Security=SSPI;
```

همانطور که می‌بینید در این مثال پارامتر Data Source به وسیله ".\SQLEXPRESS" تعیین شده است که در نتیجه ما می‌توانیم به سروری با نام SQLEXPRESS که در کامپیوترمان (localhost) قرار دارد متصل شویم. همچنین می‌توانید از کلمه localhost استفاده کنید که معادل "localhost\SQLEXPRESS" می‌باشد. می‌توان از نام یک سرور یا یک کامپیوتر به عنوان یک منبع داده استفاده نمایید. فرض کنید نام سروری که به آن متصل هستیم MyServer و نام سروری که می‌خواهیم به آن وصل شویم SQLEXPRESS می‌باشد، پس می‌توانیم از رشته اتصال زیر استفاده کنیم:

```
Data Source=MyServer\SQLEXPRESS;Initial Catalog=University; Integrated Security=SSPI;
```

همچنین به جای Data Source می‌توانید از پارامتر Server استفاده کنید:

```
"Server=.\SQLEXPRESS;Initial Catalog=University; Integrated Security=SSPI"
```

لازم است که دیتابیسی که می‌خواهیم از آن استفاده کنیم را هم مشخص کنیم. پارامترهای Initial Catalog یا Initial Database دیتابیسی که از آن استفاده می‌شود را، نشان می‌دهند. شما می‌توانید دیتابیس را در زمان اجرا به وسیله فراخوانی متد DbConnection.ChangeDatabase() یا اجرای دستور "USE DATABASE <Database Name> where <Database Name>" که از دستورات SQL است، تغییر دهید. که در این صورت نام دیتابیس جدید جایگزین نام دیتابیس قبلی می‌شود. در درس آینده چگونگی اجرای دستورات غیر پرس و جو گر SQL (non-query SQL commands) از قبیل CREATE، UPDATE، و DELETE را به شما آموزش می‌دهیم. به جای Initial Catalog یا Initial Database از پارامتر Database هم می‌توان استفاده کرد.

```
Data Source=.\SQLEXPRESS;Database=University;Integrated Security=SSPI;
```

رشته اتصال به اعتبارسنجی و اطلاعات امنیتی هم نیاز دارد. پارامتر Integrated Security که با SSPI مقداردهی شده است، نشان دهنده این است که، ما می‌خواهیم از اعتبارنامه کاربر ویندوز استفاده کنیم.

```
Data Source=.\SQLEXPRESS;Initial Catalog=University; Integrated Security=SSPI;
```

همچنین می‌توانید از پارامتر Trusted_Connection نیز استفاده کنیم و مقدار آن را برابر True قرار دهیم.

```
Data Source=.\SQLEXPRESS;Initial Catalog=University; Trusted_Connection=True;
```

یکی دیگر از راه‌های اتصال استفاده از کد کاربری و کلمه عبور است که برای این کار می‌توانیم از پارامترهای User Id و Password به صورت زیر استفاده کنیم:

```
Data Source=.\SQLEXPRESS;Initial Catalog=University;User Id=database_user;Password=the_password;
```

وقتی از Sql Server استفاده می‌کنیم، می‌توانیم یک فایل دیتابیس را به وسیله پارمتر AttachDbFileName به یک سرور محلی SQL پیوست کنیم.

```
Data Source=.\SQLEXPRESS;AttachDbFileName=C:\Data\University.mdf;
Database=University; Trusted_Connection=True;
```

ConnectionStringBuilder

دات نت دارای کلاسی به نام ConnectionStringBuilder برای هر یک از Data Providers می‌باشد. این کلاس از کلاس پایه DbConnectionStringBuilder ارث بری می‌کند. در جدول زیر انواع مختلف کلاس ConnectionStringBuilder نشان داده شده است:

ConnectionStringBuilder Class	Provider
SqlConnectionStringBuilder	SQL Server
OleDbConnectionStringBuilder	OLE DB
OdbcConnectionStringBuilder	ODBC

به عنوان مثال می‌خواهیم از SQL Server provider استفاده کنیم. می‌توانیم از کلاس SqlConnectionStringBuilder برای ایجاد رشته‌های اتصال SQL Server استفاده نماییم.

```
SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
builder.DataSource = @".\SQLEXPRESS";
builder.IntegratedSecurity = true;
builder.InitialCatalog = "Northwind";
```

این کلاس دارای خواص متناظر با پارامترهای مختلف رشته اتصال می‌باشد. از آنجاییکه منابع داده‌ای متفاوت هستند، کلاس‌های مختلف ConnectionStringBuilder نیز دارای مجموعه‌ای از خواص مختلف برای رشته اتصال می‌باشد. برای دسترسی به رشته اتصال ایجاد شده توسط کلاس ConnectionStringBuilder می‌توان از خاصیت ConnectionString استفاده نمود.

```
string connectionString = builder.ConnectionString;
```

پارامترهای پیشرفته زیادی وجود دارد که شما می‌توانید از آنها در ساخت رشته اتصال استفاده کنید. فقط ابتدایی‌ترین پارامترها را در این درس به شما آموزش دادیم، تا با مفاهیم ابتدایی ساخت رشته اتصال آشنا شوید.

Data Provider

یک data provider مجموعه‌ای از کلاس‌هایی است که اتصال به یک منبع داده مانند SQL Server را اداره و مدیریت می‌کنند. دات‌نت نه تنها برای SQL Server، بلکه برای دیگر منابع داده‌ای دارای data providerهای متفاوتی می‌باشد. به عنوان مثال، می‌توانید از data providerهای OLE DB و ODBC برای اتصال به منابع داده‌ای Microsoft Access، MySql و حتی Oracle استفاده نمایید. در زیر تعدادی از data providerهای پیشنهادی دات‌نت ذکر شده‌اند:

Provider	توضیحات
Sql Server Provider	دسترسی به یک دیتابیس SQL Server را ممکن می‌سازد.
OLE DB Provider	دسترسی به هر منبع داده‌ای که دارای راه انداز OLE DB است را ممکن می‌سازد. مانند MySQL
ODBC Provider	دسترسی به هر منبع داده‌ای که دارای راه انداز ODBC است را ممکن می‌سازد.

به این نکته توجه کنید که provider دیگر، Oracle Provider می‌باشد که مایکروسافت توصیه می‌کند به جای آن از ODP.NET استفاده شود. هر Data Provider دارای کلاس‌های متفاوتی است که شما می‌توانید با استفاده از آنها به دیتابیستان دسترسی یابید. به عنوان مثال Data Provider مربوط به SQL Server دارای کلاس‌هایی مانند SqlConnection، SqlCommand، SqlDataReader و SqlDataAdapter می‌باشد.

data provider مربوط به OLE DB شامل کلاس‌های OleDbConnection، OleDbCommand، OleDbDataReader و OleDbDataAdapter می‌باشد. اجزاء data providerهای مختلف در جدول زیر نشان داده شده‌اند:

اجزای Data Provider	Sql Server	OLE DB	ODBC
Connection	SqlConnection	OleDbConnection	OdbcConnection

Command	SqlCommand	OleDbCommand	OdbcCommand
DataReader	SqlDataReader	OleDbDataReader	OdbcDataReader
DataAdapter	SqlDataAdapter	OleDbDataAdapter	OdbcDataAdapter

هر Data Provider در فضای نام مربوط به خود قرار دارد. به عنوان مثال Sql Provider در داخل فضای نام System.Data.SqlClient و OLE DB provider در داخل فضای نام System.Data.OleDb قرار دارد.

Data Provider	فضای نام
Sql Server	System.Data.SqlClient
OLE DB	System.Data.OleDb
ODBC	System.Data.Odbc

کلاس‌های موجود در این فضاهای نامی یک رابط عمومی را پیاده سازی می‌کنند. مثلاً SqlConnection و OleDbConnection هر دو رابط IDbConnection را پیاده سازی می‌کنند. اگر به این رابط توجه کنید، مشاهده خواهید کرد که دارای خواص و متدهای عمومی کلاس Connection برای همه data providerها می‌باشد.

در درس آینده با مفاهیم بیشتری که با کدهای کوچک توضیح داده شده‌اند آشنا می‌شوید. همچنین بعد از توضیح این مفاهیم و کلاس‌ها در باره چگونگی استفاده از آنها در یک برنامه بحث می‌کنیم. لازم نیست که همه کلاس‌ها را حفظ کنید چون که می‌توانید حتی بخش آموزش کلاس‌ها را رد کرده و وارد بخش ایجاد یک برنامه ADO.NET شوید، سپس عملکرد هر کدام از کلاس‌ها را که متوجه نشدید به عقب برگشته و آن را یاد بگیرید.

کلاس Connection

هر Data Provider در ADO.Net شامل یک کلاس Connection است که از کلاس system.data.common.DbConnection ارث بری می‌کند. برای همه کلاس‌های connection مربوط به data providerهای مختلف DbConnection به عنوان کلاس پایه به حساب می‌آید. جدول زیر کلاس‌های Connection برای هر Data Provider را نشان می‌دهد.

Data provider	Connection class
SQL Server	sqlConnection
OLE DB	OleDbConnection
ODBC	OdbcConnection

کلاس DbConnection رابط IDbConnection را پیاده سازی می‌کند، بنابراین متدها و خصوصیات (properties) مربوط به آن را دارد و از همآنها برای تعریف یک connection یا باز کردن یک connection به یک منبع داده استفاده می‌کند. باز کردن یک connection باعث اشغال حافظه کامپیوتر می‌شود، مثلاً ممکن است برای انتقال اطلاعات از بافر کارت شبکه استفاده کند، هم چنین بدیهی است که برای استفاده از آن سایر سخت افزارها مثل RAM و CPU درگیر هستند، بنابراین بعد از این که استفاده ما از Connection تمام شد باید آن را ببندیم. برای وصل

شدن به یک دیتابیس ما باید آدرس و تنظیمات و سایر چیزهای لازم برای وصل شدن به دیتابیس مثل آدرس آن را به برنامه بدهیم، رشته اتصال (Connection String) برای ما این کار را انجام می‌دهد. در زیر متدها و خصوصیات رابط IDbConnection که در همه کلاس‌های Connection وجود دارند را ذکر کرده‌ایم:

خاصیت	توضیح
ConnectionString	رشته ای است که از آن برای برقراری ارتباط با بانک اطلاعاتی استفاده می‌شود.
ConnectionTimeout	مدت زمانی که باید منتظر برقراری ارتباط با بانک بود را مشخص می‌کند.
Database	نام بانک اطلاعاتی که قرار است ارتباط با آن برقرار شود را مشخص می‌کند.
State	وضعیت فعلی ارتباط را مشخص می‌کند.

متد	توضیح
BeginTransaction()	تراکنش با بانک را شروع می‌کند.
ChangeDatabase()	ارتباط را از یک دیتابیس به دیتابیس دیگر تغییر می‌دهد.
Close()	ارتباط با بانک را قطع می‌کند.
CreateCommand()	یک شیء SqlCommand ازروی دستور SQL تعیین شده ایجاد می‌کند.
Open()	ارتباط با بانک را برقرار می‌کند.

باید به این نکته اشاره کرد که هر کدام از کلاس‌های connection مربوط به هر Data Provider، می‌تواند بیشتر از این دو جدول متد و property داشته باشد، ولی در این صورت برای خود آن کلاس یکتاست، که ما در این جدول‌ها ذکر نکردیم. برای مثال‌های این مقاله ما از SQL Sever به عنوان Data Provider استفاده می‌کنیم. بنابراین ما از کلاس SqlConnection به عنوان کلاس connection ای که به SQL Server که یک Data Provider است، استفاده می‌کنیم. برای ساختن یک شیء از این کلاس، از خط کد زیر استفاده می‌کنیم (در اینجا ما از یک سازنده که هیچ پارامتری نمی‌گیرد استفاده کردیم):

```
SqlConnection connection = new SqlConnection();
```

حالا برای این که این شیء بداند که باید به کدام دیتابیس و چه نوع دیتابیسی وصل شود باید Connection String را مشخص کنیم. این شیء connection ی که ساخته‌ایم یک property به اسم connectionString دارد که می‌توان با استفاده از آن این کار را انجام دهیم. در خط زیر به برنامه می‌گوییم:

```
connection.ConnectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=University;" + "Integrated
Security=SSPI";
```

به جای این دو خط کد می‌توانیم در هنگام ساختن شئ، connectionString را هم مستقیماً به آن بدهیم (به این حالت کلاس که پارامتر هم می‌گیرد overloaded constructor یا سربارگذاری سازنده می‌گویند).

```
SqlConnection connection = new SqlConnection(@"Data Source=.\SQLEXPRESS;" +
                        "Initial Catalog=University;Integrated Security=SSPI");
```

این connection string می‌گوید که از دیتابیس university استفاده کن. فقط وقتی که connection بسته است می‌توان connection string را مقداردهی کرد یا تغییر داد. کلاس connection به ما این امکان را می‌دهد که به قسمت‌های مختلف connection string هم دسترسی داشته باشیم. این کار را با استفاده از Propertyهای آن می‌توانیم انجام دهیم.

یک property به اسم Database داریم که به برنامه می‌گوید که از کدام دیتابیس استفاده کن. این خاصیت در connection string با نام Initial Catalog مشخص شده است. با استفاده از کلاس connection و متد ChangeDatabase می‌توان دیتابیس مورد استفاده در برنامه را تغییر داد (البته همان طور که گفتیم connection باید بسته باشد).

```
connection.ChangeDatabase("AnotherDatabase");
```

این خط کد، دیتابیس مورد استفاده را از University به AnotherDatabase یا دیتابیسی دیگر عوض می‌کند. یک Property دیگر connectionTimeout داریم که مدت زمانی که connection باز است را به ما می‌گوید که اگر آن را مقدار دهی نکرده‌اید به صورت پیش‌فرض مقدار ۱۵ برای آن مقداردهی می‌شود. بعد از این زمان یک exception داده می‌شود که زمان تمام شده است.

برای باز کردن یک connection ما از متد ()open استفاده می‌کنیم. قبل از صدا زدن این متد باید connection string را مقداردهی کرده باشیم. بعد از این که از این connection استفاده کردیم آن را باید حتماً ببندیم، که این کار را هم با متد ()close انجام می‌دهیم. وقتی که از using استفاده می‌کنیم، بلافاصله که کار ما تمام شد، به صورت اتوماتیک connection بسته می‌شود.

```
using (SqlConnection connection = new SqlConnection())
{
   // some code....
   connection.Open();
   // some code....
}
```

تعریف و مقداردهی را در داخل پرانتز using قرار می‌دهیم. این به ما می‌گوید که connection ای که در داخل پرانتز مربوط به using قرار دارد، فقط در داخل بلاک آن قابل استفاده است. بعد از این که بلاک تمام شد connection بسته خواهد شد. اگر شما از متد ()close استفاده می‌کنید، باید آن را در قسمت finaly فراخوانی کنید.

```
try
{
   connection.Open();
}
catch(SqlException)
```

```
{

}
finally
{
   connection.Close();
}
```

این کار برای این است که اگر ما آن را در بلاک try قرار دهیم و یک exception رخ دهد در این صورت آن کد اجرا نمی‌شود و در نتیجه connectionباز می‌ماند ولی در این حالت حتی با وجود رخداد exception هم بلاک finaly اجرا می‌شود و به مشکلی بر نمی‌خوریم. با استفاده از خاصیتی به نام state که مربوط به connection است می‌توان از وضعیت connection با خبر شد که آیا باز است یا بسته. State مقدار خود را از system.data.connectionStateEnumeration می‌گیرد. در جدول زیر مقادیر نوع شمارشی ConnectionState ذکر شده است:

مقدار	توضیح
Broken	مشخص می‌کند که ارتباط با بانک با شکست مواجه شده است.
Closed	مشخص می‌کند که ارتباط بسته است.
Connecting	مشخص می‌کند که ارتباط با منبع داده برقرار است.
Executing	مشخص می‌کند که ارتباط در حال اجرای یک دستور است.
Fetching	مشخص می‌کند که ارتباط در حال دریافت داده است.
Open	مشخص می‌کند که ارتباط باز است.

کلاس connection دو رویداد (event) هم در اختیار شما قرار می‌دهد، که شما می‌توانید از آنها استفاده کنید. اولی InfoMessage است که از آن می‌توان برای گرفتن پیام‌های اطلاعاتی استفاده کرد. و هر وقت که وضعیت مربوط به connection عوض شود رویداد StateChange رخ می‌دهد (trigger می‌شود).

```
static void con_StateChange(object sender, System.Data.StateChangeEventArgs e)
{
    Console.WriteLine("State has been changed from {0} to {1}.",
            e.OriginalState.ToString(), e.CurrentState.ToString());
}

static void Main()
{
    SqlConnection con = new SqlConnection();
    con.ConnectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;" +
                           "Integrated Security=SSPI";
    con.StateChange += new System.Data.StateChangeEventHandler(con_StateChange);
    con.Open(); // Opens a connection
    con.Close(); // Closes a connection
}
```

```
State has been changed from Closed to Open.
```

```
State has been changed from Open to Closed.
```

Con_StateChange را برای کنترل کردن رخداد عوض شدن وضعیت داریم و این تابع را به property کانکشن (StateChange) انتساب می‌دهیم تا هر وقت وضعیت کانکشن عوض شد از این استفاده کند و کارهای کنترلی را انجام دهد که در اینجا ما به خاطر آموزش فقط یک جمله پرینت می‌کنیم. وقتی که ما متد ()open را صدا می‌زنیم وضعیت کانکشن از close به open تغییر می‌کند و کنترل کننده رویداد (event handler) را صدا می‌زند و خط اول را چاپ می‌کند و وقتی که متد ()close را صدا می‌زنیم همین اتفاق می‌افتد و خط دوم را چاپ می‌کند.

کلاس command

هر Data Provider کلاس Command مربوط به خودش را دارد که برای اجرای دستورات SQL (دستوراتی که معمولاً هم لفظ execute در آنها دیده می‌شود و جنبه اجرایی دارند مانند sqlCommand.ExecuteScalar) یا ذخیره سازی یک رویه (procedure) در دیتابیس، استفاده می‌شوند. هر کلاس command از یک کلاس Systam.Data.Common.DbCommand ارث بری می‌کند. جدول زیر کلاس‌های command مختلف را برای Data Providerهای مختلف را نشان می‌دهد.

کلاس Command	Data Provider
SqlCommand	Sql Server
OleDbCommand	OLE DB
OdbcCommand	ODBC

کلاس DbCommand رابط IDbCommand را پیاده سازی می‌کند یعنی از متدها و خاصیت‌های آن استفاده می‌کند، که در جدول زیر متدها و خاصیت‌هایی را که استفاده می‌کند را می‌بینید.

خاصیت	توضیحات
CommandText	دستور Sql یا رویه ذخیره شده در دیتابیس (stored procedure) یا اسم جدول را مشخص می‌کند
CommandTimeout	زمان مورد نیازی را به ما نشان می‌دهد که برای اجرای یک دستور لازم است. اگر اجرای یک دستور از این بیشتر طول بکشد یک استثناء رخ خواهد داد. این زمان به صورت پیش‌فرض ۳۰ ثانیه است.
CommandType	نوع دستوری را که در commandText مشخص شده است به ما نشان می‌دهد که مقدار خود را از System.Data.CommandType می‌گیرد. که سه مقدار می‌تواند بگیرد. Text که از آن برای دادن دستور Sql، stroedPeocedure برای رویه‌های ذخیره شده، TableDirect که از آن برای گرفتن تمام سطر و ستون‌ها از چند جدول استفاده می‌شود. به صورت پیش‌فرض text مورد استفاده قرار می‌گیرد.

Connection	کانکشنی را مشخص می‌کند که command با آن کار می‌کند. کلاس command باید با به یک کانکشن باز وصل شود که در واقع این دستور بر روی آن کانکشن اجرا می‌شود.
Parameters	مجموعه‌ای از پارامترها که در CommandText تعریف شده است

متدها	توضیحات
Cancel	دستوری که در حال اجراست را لغو می‌کند
CreateParameter	یک شیئ جدید از پارامتر می‌سازد که می‌تواند به مجموعه command.parameter اضافه شود
ExecuteReader	دستور را اجرا می‌کند و یک سری اطلاعات به فرم DateReader برمی گرداند که فقط هم قابل خواندن است. مثلاً تنها برای دیدن حاصل یک query.
ExecuteNonQuery	یک دستور اجرا می‌کند و ردیف‌هایی که تحت تأثیر این دستور تغییر کردند را بر می‌گرداند. مثلاً برای اجرای update، delete، Insert.
ExecuteScalar	یک دستور را اجرا می‌کند و تنها یک مقدار را برمی گرداند. مثلاً برای برگرداندن اولین خانه از اولین ردیفی که نتیجه ماست استفاده می‌شود.

برای مثال‌ها ما از بین data provider‌ها از SQL Server استفاده می‌کنیم. برای ساختن یک شیء command از سازنده بدون پارامتر استفاده می‌کنیم.

```
SqlCommand command = new SqlCommand();
```

این شیء تا زمانی که پرس و جوی مورد نظر را با استفاده از CommandText به آن ندهیم بدرد نمی‌خورد.

```
command.CommandText = "SELECT * FROM Students";
```

دو دستور بالا را به صورت زیر هم می‌توان ادغام کرد:

```
SqlCommand command = new SqlCommand("SELECT * FROM Students");
```

هم چنین باید به دستور بفهمانیم که روی کدام کانکشن باید این پرس و جو اجرا شود.

```
SqlConnection connection = new SqlConnection();
SqlCommand command = new SqlCommand();
command.Connection = connection;
```

سازنده دیگر برای ساختن شیء command (روش دیگری برای ساختن command) این است که می‌توان مستقیماً CommandText و Connection را به سازنده داد.

```
SqlCommand command = new SqlCommand("SELECT * FROM Students", connection);
```

متد دیگری که در کلاس DbConnection وجود دارد CreateCommand() است. این متد یک DbCommand برمی گرداند که connection هم به آن متصل است که می‌توان commandText را به آن داد.

```
SqlCommand command = connection.CreateCommand();
command.CommandText = "SELECT * FROM Students";
```

برای اجرای دستور اول باید یک connection باز کرد. برای استفاده از دیتابیس باید از متد ExecuteReader() استفاده کرد که یک شیء dataReader برمی گرداند که با آن می‌توان به ردیف‌های یک جدول از دیتابیس دسترسی داشت. برای اجرای دستورات update، Insert، delete باید از متد ExecuteNonQuery() استفاده کرد. این متدها در درس‌های بعد توضیح داده خواهد شد.

کلاس Parameter

ما می‌توانیم هنگام تعریف کردن عبارت پرس و جو در commandText که یک خصوصیت از کلاس DbComnmand است به جای این که کل عبارت پرس و جو را به صورت رشته تعریف کنیم در آن از پارامترها هم استفاده کنیم. در صورت عدم استفاده از پارامترها، مجبوریم به صورت دستی مقادیر را در عبارت پرس و جو بنویسیم. مثلاً می‌خواهیم تمام دانشجوهایی که دارای شماره دانشجویی خاصی هستند را، انتخاب کنیم. برای این کار شماره دانشجویی را از کاربر می‌گیریم و دستور را تولید می‌کنیم.

```
int studentId = Int32.Parse(Console.ReadLine());
SqlCommand command = new SqlCommand();
command.CommandText = "SELECT * FROM Students WHERE StudentID = " + StudentID;
```

در مثال بالا از کاربر خواسته می‌شود که شماره دانشجویی را بدهد. این کار را با Console.ReadLine() انجام، سپس در خط سوم آن را به commandText می‌دهیم. این روش برای کد نویسی راحت است ولی از نظر امنیتی مشکل دارد چرا که کاربر هر چیزی را در query می‌تواند اضافه کند. این روش برای کار کردن با پارامترها روش درستی نیست به خاطر همین C# برای کار کردن با پارامترها، کلاس‌هایی را در اختیار ما قرار داده است که در زیر می‌بینیم. همه این کلاس‌ها از کلاس پایه DbParameter ارث بری می‌کنند.

کلاس Parameter	Provider
SqlParameter	SQL Server
OleDbParameter	OLE DB
OdbcParameter	ODBC

قبل از کار با کلاس‌های پارامتر باید در داخل CommandText مشخص کنیم که چه چیزی پارامتر است. برای این کار قبل از اسم پارامتر باید علامت @ قرار دهیم به مثال زیر.

```
SqlCommand command = new SqlCommand();
command.CommandText = "SELECT * FROM Students WHERE Age=@Age";
```

در مثال بالا به @Age در داخل عبارت SQL توجه کنید. در عبارت پرس و جو می‌بینیم تمام دانشجویانی که سن آنها برابر با @Age است را عبارت پرس و جو می‌گیرد که Age یک متغیر و پارامتری است که ما مشخص کرده‌ایم. وقتی C# این پرس و جو را می‌سازد، به جای @Age مقداری را قرار می‌دهد. برای استفاده از یک پارامتر باید اسم پارامتر و نوعی که این پارامتر در خود دارد را مشخص کنیم. ADO.Net خود انواع داده‌ای احتمالی را که ممکن است که ما بخواهیم استفاده کنیم را در اختیار ما قرار داده و می‌توانیم از آنها استفاده کنیم. برای استفاده از آنها از انواع شمارشی System.Data.SqlDbType استفاده می‌کنیم یا اگر از دیتابیس OLE DB استفاده می‌کنیم، باید از System.Data.OleDbType استفاده کنیم. کد زیر مثالی است که به ما می‌گوید که چگونه می‌شود از پارامترها استفاده کرد.

```
SqlParameter parameter1 = new SqlParameter("@Age", SqlDbType.Int);
```

سازنده مربوط به DbParameter در کد بالا که می‌بینیم اسم پارامتر و نوعی که این پارامتر می‌پذیرد را می‌گیرد. اگر هم بخواهیم از رشته‌ای با طول متغیر یعنی nvarchar استفاده کنیم می‌توانیم از سازنده زیر برای آن استفاده کنیم.

```
SqlParameter parameter2 = new SqlParameter("@FirstName", SqlDbType.VarChar, 100);
```

سومین پارامتر در بالا، سایز نوع داده را مشخص می‌کند. در نهایت هم می‌توان مقداری که باید جایگزین آن پارامتر شود را، مشخص کرد. مثل کد زیر:

```
parameter1.Value = 18;
parameter2.Value = "Mandy";
```

حالا می‌توانیم شیء پارامتر تولید شده را به خاصیت Parameters شیء DbCommand اضافه کنیم.

```
command.Parameters.Add(parameter1);
```

اگر چند پارامتر داشتیم همه آنها را اضافه می‌کردیم و ترتیب اضافه کردنشان اهمیتی نداشت. اما در هنگام استفاده از OLE DB باید آنها را به همان ترتیبی که در پرس و جو مشخص شده است، اضافه کنید. روش دیگر اضافه کردن پارامترها روش ()addWithValue است که دیگر لازم نیست که پارامترها را بسازیم و نوع آنها را معلوم کنیم.

```
command.CommandText = "UPDATE Students SET FirstName=@FirstName, LastName=@LastName " + "WHERE StudentId=@StudentId";

command.Parameters.AddWithValue("@FirstName", "John");
command.Parameters.AddWithValue("@LastName", "Smith");
command.Parameters.AddWithValue("@StudentId", 1);
```

آرگومان اول پارامتر مربوطه است و آرگومان دوم مقداری است که جایگزین این پارامتر می‌شود. پارامتر دوم به صورت اتوماتیک به نوعی که در دیتابیس باید وارد شود، تبدیل می‌شود. دوباره باید بگوییم که اگر شما از OLE DB استفاده می‌کنید باید پارامترها را به همان ترتیبی که در دستور دیده می‌شود وارد کنید.

```
command.CommandText = "UPDATE Students SET FirstName=@FirstName, LastName=@LastName " +
                      "WHERE StudentId=@StudentId";

command.Parameters.AddWithValue("@LastName", "John");
command.Parameters.AddWithValue("@FirstName", "Smith");
```

```
command.Parameters.AddWithValue("@EmployeeId", 1);
```

مثلاً در مثال بالا باید اول @FirstName و سپس @LastName اضافه شود، ولی برعکس اضافه شده که مشکل ساز خواهد شد. بنابراین هنگام استفاده از OLE DB مراقب باشید.

کلاس DataReader

یک شیء dataReader دسترسی به دیتابیس را برای ما فراهم می‌کند. این دسترسی فقط برای دیدن اطلاعات است و نمی‌توان اطلاعات را تغییر داد (read-only). استفاده از این کلاس یک روش متصل (connected) است یعنی آفلاین نمی‌توان از آن استفاده کرد و همیشه یک connection باز برای ما لازم است. هر data provider، dataReader خودش را دارد که همه آنها از کلاس پایه system.data.common.DbDatareader ارث بری می‌کنند.

کلاس DataReader	provider
SqlDataReader	SQL Server
OleDbDataReader	OLE DB
OdbcDataReader	ODBC

کلاس DbDataReader شامل متدها و خاصیت‌هایی برای خواندن از دیتابیس هستند. تعدادی از آنها را در جدول‌های زیر آورده‌ام.

خاصیت	توضیحات
FieldCount	تعداد ستون‌های ردیف (فعلی) را به ما می‌گوید
HasRows	به ما می‌گوید که این query حداقل یک ردیف دارد یا نه
IsClosed	مشخص می‌کند که Dbdatareader بسته است یا نه
RecordsAffected	تعداد ستون‌هایی که با اجرای دستور SQL تغییر یافته‌اند را بر می‌گرداند (ستون‌هایی که insert،update یا delete شده‌اند)

متد	توضیحات
GetBoolean	مقدار یک ستون را به صورت Boolean بر می‌گرداند
GetChar	مقدار یک ستون را به صورت char برمی گرداند
GetDataTypeName	اسم dataType (نوع داده) ستون فعلی را برمی گرداند

GetDateTime	مقدار ستون را به عنوان dateTime (شیئی که تاریخ و زمان را می‌دهد) برمی گرداند
GetDecimal	مقدار دهدهی ستون را برمی گرداند
GetDouble	مقدار اعشار ستون را بر می‌گرداند
GetFieldType	نوع فیلد ستون مورد نظر را بر می‌گرداند
GetInt32	مقدار ستون مورد نظر را به عنوان interger برمی گرداند
GetName	اسم ستون را بر می‌گرداند
GetString	مقدار ستون را به صورت string می‌دهد
GetValue	مقدار ستون را به صورت یک شیء برمی گرداند
GetValues	تمام ستون‌های یک ردیف را به صورت آرایه ای از اشیاء بر می‌گرداند
NextResult	وقتی در حال خواندن دسته‌ای از نتایج هستیم این متد به ما نتیجه بعدی را می‌دهد
Read	Reader (برای خواندن استفاده می‌شود) را به رکورد بعدی می‌برد

dataReader با هر بار خواندن تنها یک رکورد را به حافظه می‌آورد. این کار برای استفاده بهینه از حافظه مفید است. Datareader احتیاج دارد که connection باز باشد، پس برای استفاده از آن باید کانکشن را اول باز کرد و به محض اتمام کار آن را بست. برای یک دستور SELECT در SQL باید از تابع ExecuteReader() استفاده کنیم، تا به ما یک شیء DbDataReader برگرداند.

```
SqlCommand command = new SqlCommand("SELECT * FROM Students", connection);
SqlDataReader reader;

connection.Open();
reader = command.ExecuteReader();
```

می‌بینیم قبل از این که از متد ExecuteReader() استفاده کنیم باید کانکشن را باز کنیم. این کار را با استفاده از متد DbConnection.Open() انجام می‌دهیم. نسخه دیگر ExecuteReader() پارامتر می‌گیرد (constructor overloader) که مجموعه این پارامترها را می‌توانیم در System.Data.CommandBehavior ببینیم. در جدول زیر یک سری از این مقادیر را می‌بینیم.

مقدار	توضیحات
CloseConnection	وقتی متد close() از dataReader صدا زده می‌شود، بلافاصله کانکشن را می‌بندد
Default	رفتار پیش‌فرض DataReader را مشخص می‌کند
SingleResult	پرس و جو، یک مقدار واحد بر می‌گرداند

SingleRow	انتظار می‌رود که پرس و جو یک مقدار واحد را برگرداند

به عنوان مثال وقتی شما می‌خواهید که فقط یک ردیف خوانده شود باید SingleRow را به عنوان پارامتر به آن ارسال کنید مانند کد زیر:

```
reader = command.ExecuteReader(CommandBehavior.SingleRow);
reader = command.ExecuteReader(CommandBehavior.SingleRow);
```

با استفاده از OR بیتی می‌توانیم این رفتارها را با هم ترکیب کنیم.

```
reader=command.ExecuteReader(CommandBehavior.SingleRow|CommandBehavior.CloseConnection);
```

وقتی که ExecuteReader() اجرا شد و یک نمونه (Instance) از DbDataReader را برای ما برگرداند آن را در یک متغیر قرار می‌دهیم. حال با استفاده از حلقه در میان نتایجی که به وسیله دستور SELECT برگردانده شده‌اند حرکت کنیم. به کد زیر توجه کنید:

```
while (reader.Read())
{
   MessageBox.Show(reader["FirstName"].ToString());
}
```

در قسمت شرط حلقه ما متد Read() را از DataReader فراخوانی کرده‌ایم که ردیف اول را که از پرس و جو به دست آمده است به ما بر می‌گرداند. اگر عملیات خواندن یک رکورد (سطر) موفقیت آمیز باشد در این صورت true را برمی گرداند و در نتیجه حلقه ادامه پیدا می‌کند در غیر این صورت false برمی گرداند و از حلقه خارج می‌شود. وقتی که به صورت موفقیت آمیز یک ردیف را برگردانید حالا می‌توانیم با استفاده از نام ستون مقدار یک خانه از جدول را استفاده کنیم همانطور که در کد بالا آن را چاپ می‌کند. بعد از این که بلاک تمام شد دوباره شرط حلقه اجرا می‌شود پس متد Read() بار دیگر اجرا می‌شود و این بار ردیف بعدی را برمی گرداند. به جای این روش ما می‌توانیم از متد Get() مربوط به DataReader استفاده کنیم، که ایندکس مربوط به ستونی که می‌خواهیم بخوانیم را، دریافت می‌کند. به عنوان مثال برای برگرداندن مقدار ستون اول از کد زیر استفاده می‌کنیم:

```
int studentID = reader.GetInt32(0);
```

همانطور که مشاهده می‌کنید مقدار ستون اول (اندیس ۰) به عنوان یک عدد صحیح برگشت داده می‌شود. بعد از استفاده کردن از DataReader باید آن و همچنین connection را بست تا بی خود از منابع استفاده نکند.

```
reader.Close();
connection.Close();
```

در درس بعد ما از کلاس‌های DateReader برای محتویات داخل یک دیتابیس در حالت متصل صحبت می‌کنیم.

کلاس DataAdapter

DataAdaper را می‌توان به عنوان یک پل بین منبع اطلاعاتی واقعی (data source) و برنامه خودمان در نظر بگیریم که معمولاً همراه DataSet مورد استفاده قرار می‌گیرد. استفاده DataAdapter و DataSet همراه با هم، یک روش غیر متصل (disconnected) برای استفاده از منبع اطلاعاتی یا همان Data Source است (از این به بعد به جای منبع اطلاعاتی از لفظ Data source استفاده می‌کنیم چرا که قابل فهم تر است).

یعنی دیگر لازم نیست یک connection برای استفاده از آنها باز کنیم. DataAdapter به ما این اجازه را می‌دهد که DataSet را با استفاده از مقادیری که در Data Source وجود دارد، پر کنیم. کلاس DataAdapter از کلاس پایه System.Data.Common.DbDataAdapter به ارث برده می‌شود و هر Data provider، dataAdapter مختص به خود را دارد.

کلاس dataAdapter	Data provider
SqlDataAdapter	SQL Server
OleDbDataAdapter	OLE DB
OdbcDataAdapter	ODBC

کلاس DbDataAdapter متدها و property های جدول زیر را در اختیار ما قرار می‌دهد.

خاصیت	توضیحات
DeleteCommand	شیء command را برای پاک کردن یک رکورد در اختیار ما قرار می‌دهد
FillCommandBehavior	رفتار command را که برای پر کردن یک dataAdapter مورد استفاده قرار می‌گیرد را برای ما مشخص می‌کند
InsertCommand	شیء command را برای پر کردن یک رکورد در اختیار ما قرار می‌دهد
SelectCommand	شیء command را برای انتخاب کردن (Select) یک رکورد در اختیار ما قرار می‌دهد
UpdateBatchSize	تعداد command هایی را که می‌توان به عنوان یک دسته اجرا کرد، به ما می‌گوید
UpdateCommand	شیء command را برای update کردن یک رکورد در اختیار ما قرار می‌دهد

متد	توضیحات
AddToBatch	یک شیء command را به دسته‌ای که قرار است اجرا شود اضافه می‌کند
ClearBatch	تمام اشیاء command ای که قرار است اجرا شود را از دسته پاک می‌کند
ExecuteBatch	دسته commandها را اجرا می‌کند
Fill	خاصیت Selectcommand را اجرا می‌کند و نتیجه را در dataset مربوطه قرار می‌دهد
InitializeBatching	عملیات دسته بندی را مقداردهی اولیه می‌کند.
GetFillParameters	پارامترهای selectCommand را می‌گیرد

TerminateBatching	عملیات دسته بندی را پایان می‌بخشد
Update	دستورهای update، Insert و delete مربوط به این dataAdapter را صدا میزند و با استفاده از آن دستورها data source را update می‌کند.

برای مثال‌های بعد ما از SQL Server به عنوان data provider استفاده می‌کنیم. برای ساختن یک dataAdapter شما می‌توانید از نوع بدون پارامتر آن استفاده کنید.

```
SqlDataAdapter adapter = new SqlDataAdapter();
```

dataApadpter دستور را در data source اجرا می‌کند و یک خاصیت به اسم selectCommand دارد که شیء DbCommand را (که این شیء select را مشخص می‌کند و برای استخراج از دیتابیس است) به آن می‌دهیم. مانند مثال زیر:

```
SqlCommand selectCommand = new SqlCommand("SELECT * FROM Students", connection);
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = selectCommand;
```

برای اجرای دستوری که در selectCommand مشخص کردیم باید از تابع ()fill در dataAdaper استفاده کنیم. برای استفاده از این تابع ما باید از یک کلاس DataTable یا DataSet استفاده کنیم. در مثال زیر یک شیء از DataTable به وسیله مقادیر به دست آمده از منبع داده پر شده است:

```
DataTable myTable = new DataTable("TableName");

adapter.Fill(myTable);
```

این کد در واقع دستوری که در selectCommand مشخص کرده‌ایم را، اجرا می‌کند و نتیجه را در DataTable ی که مشخص کرده‌ایم، به اجرا در می‌آورد. شکل دیگر متد ()Fill یک DataSet و نام DataTable ایجاد شده را قبول می‌کند:

```
DataSet myDataSet = new DataSet();

adapter.Fill(myDataSet, "TableName");
```

بعد از اجرای کد بالا یک DataTable جدید با نام TableName به خاصیت Tables، DataSet اضافه می‌شود. حال می‌توان با استفاده از خاصیت DataTable، Row به داده‌ها دسترسی یافت. dataAdapter، خاصیت‌های دیگری هم دارد که برای UpdataCommand و InsertCommand و DeleteCommand به کار می‌رود که این دستورها را با commandBuilder می‌سازیم که در درس‌های آینده به آنها می‌پردازیم.

کلاس DataSet

کلاس DataSet اطلاعاتی که از دیتابیس رسیده را در خود ذخیره می‌کند. کلاس DataSet به شما این امکان را می‌دهد که داده‌ها را به صورت غیر متصل (disconnected) نگهداری و دستکاری کنید. یعنی یک بار از دیتابیس اطلاعات را می‌کشیم و در این کلاس نگه می‌داریم و هر کاری

داشته باشیم با آن انجام می‌دهیم. این که می گوییم غیر متصل یعنی ما با اطلاعاتی که در دیتابیس وجود دارد کار می‌کنیم در حالی که از دیتابیس واقعی جدا هستیم و به آن دسترسی نداریم. نکات مهمی که در مورد DataSet وجود دارد به شرح زیر است:

- DataSet یک دیتابیس کوچک است که داخل حافظه قرار دارد.
- DataSet همانند دیتابیس واقعی می‌تواند دارای یک یا چند جدول باشد. به هر کدام از این جداول یک DataTable می گویند.
- DataTable دارای سطر و ستون می‌باشد. به هر سطر آن DataRow و به هر ستون آن DataColumn می گویند.

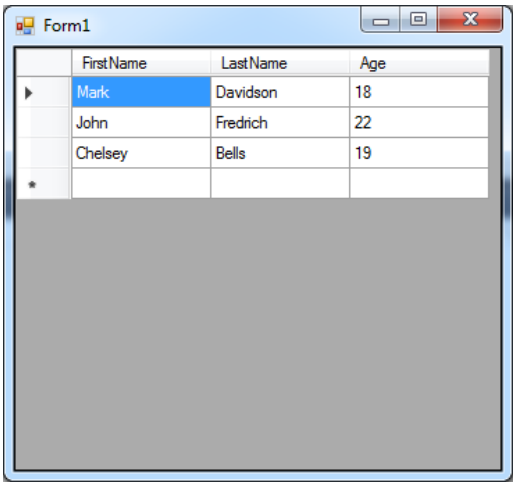
نکته مثبت در مورد dataTable و DataColumn این است که می‌توان آنها را با استفاده از اسم مقداردهی کرد. این کار را می‌توان با استفاده از دو خاصیت به نام‌های TableName و ColumnName و سازنده آن انجام داد. حالا یک DataTable ساده ساخته و آن را به DataSet اضافه می‌کنیم. یک ویندوز فرم معمولی درست می‌کنیم و اسم آن را DataSetDemo می‌گذاریم سپس یک dataGridView داخل آن فرم قرار می‌دهیم و آن را Dock می‌کنیم تا کل فرم را پر کند. بر روی فرم دابل کلیک می‌کنیم تا کنترل کننده رویداد (event handler) مربوط به رویداد Load فرم را بسازد. کد زیر را به آن اضافه می‌کنیم. حواستان باشد که System.data را به آن به عنوان فضای نام، اضافه کرده باشد. این کار را با کد using System.Data; باید انجام دهیم.

```
1  private void Form1_Load(object sender, EventArgs e)
2  {
3      DataSet personsDataSet = new DataSet();
4      DataTable personsTable = new DataTable("Persons");
5      personsDataSet.Tables.Add(personsTable);
6
7      DataColumn firstNameColumn = new DataColumn("FirstName");
8      DataColumn lastNameColumn  = new DataColumn("LastName");
9      DataColumn ageColumn       = new DataColumn("Age");
10
11     personsTable.Columns.AddRange(new DataColumn[] { firstNameColumn, lastNameColumn, ageColumn });
12
13     DataRow firstPerson  = personsTable.NewRow();
14     DataRow secondPerson = personsTable.NewRow();
15     DataRow thirdPerson  = personsTable.NewRow();
16
17     firstPerson["FirstName"]  = "Mark";
18     firstPerson["LastName"]   = "Davidson";
19     firstPerson["Age"]        = 18;
20
21     secondPerson["FirstName"] = "John";
22     secondPerson["LastName"]  = "Fredrich";
23     secondPerson["Age"]       = 22;
24
25     thirdPerson["FirstName"]  = "Chelsey";
26     thirdPerson["LastName"]   = "Bells";
27     thirdPerson["Age"]        = 19;
28
29     personsTable.Rows.Add(firstPerson);
30     personsTable.Rows.Add(secondPerson);
31     personsTable.Rows.Add(thirdPerson);
32
33     dataGridView1.DataSource = personsDataSet.Tables["Persons"];
34 }
```

در کد بالا و در خط ۳ ابتدا یک DataSet، سپس در خط ۴ یک جدول یا همان DataTable با نام Persons ایجاد کرده‌ایم. DataSet دارای خاصیتی به نام Tables می‌باشد. این خاصیت شامل مجموعه DataTable های DataSet می‌باشد و دارای متدی به نام Add() است که با استفاده از آن می‌توان یک DataTable جدید به DataSet اضافه کرد.

در خط ۵ DataTable ایجاد شده در خط ۴ را با استفاده از متد ()Add به DataSet اضافه می‌کنیم. تا اینجای کار DataSet ایجاد و DataTable به آن اضافه شد. حال نوبت به اضافه کردن سطرها و ستون‌ها به DataTable است. در خطوط ۹-۷ سه ستون با استفاده از کلاس DataColumn و به نام‌های FirstName، LastName و Age ایجاد و در خط ۱۱ آنها را با استفاده از متد ()AddRange به DataTable اضافه می‌کنیم.

این متد یک آرایه از ستون‌ها را می‌گیرد و آنها را به DataTable اضافه می‌کند. برای ایجاد سطرها هم از کلاس DataRow در خطوط ۱۵-۱۳ استفاده کرده‌ایم. در همین خطوط هم با استفاده از متد ()NewRow اعلام می‌کنیم که این سطرها مربوط به جدول personsTable است. در خطوط ۲۷-۱۷ مقادیر را در داخل این سطرها می‌ریزیم. سپس در خطوط ۳۱-۲۹ این سه سطر را به DataTable اضافه می‌کنیم. برای نمایش اطلاعات داخل این DataTable در داخل DataGridView هم در خط ۳۳ می گوییم که منبع داده DataGridView یا همان DataSource آن همان جدول موجود در DataSet به نام Persons است:



اتصال به دیتابیس با کد

اولین مثالی که در اتصال به دیتابیس مطرح کردیم از قابلیت‌های ویژوال استودیو استفاده کردیم و اصلاً نیاز به کد زدن نداشت. الان یک سری از درس‌ها را شروع می‌کنیم که در آنها تاکید بیشتر بر روی اتصال به منابع داده‌ای مختلف با استفاده از کلاس‌های ADO.NET است.

چیزی که در درس بعد توضیح می‌دهیم این است که چگونه Insert، delete، Update کنیم. این کار را به دو روش انجام می‌دهیم. یک روش متصل (connected) و روش دیگر غیر متصل (disconnected). اگر چه این دو روش هر دو نهایت یک کار را انجام می‌دهد ولی تفاوت‌هایی کوچکی دارند که باید در شرایط مختلف انتخاب کنیم که از کدام یک از آنها، استفاده کنیم.

اگر می‌خواهیم پرس و جوهای سریع از دیتابیس‌های بزرگ بگیریم بهتر است که از روش متصل استفاده کنیم. چرا که اگر بخواهیم از روش غیر متصل استفاده کنیم، باید قسمتی از این دیتابیس را اول به حافظه ببریم، که برای دیتابیس‌های بزرگ غیر منطقی است. نقطه ضعف این روش

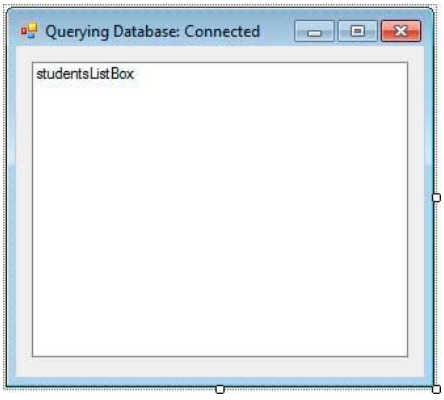
این است که باید یک کانکشن باز کنید و دیگر کانکشن‌ها نمی‌توانند از آن استفاده کنند تا زمانی که آن را ببندید. یعنی برای هر بار استفاده باید یک کانکشن باز کرد و بلافاصله بست.

اگر می‌خواهید با قسمت کوچکی از دیتابیس کار کنید روش غیر متصل بهتر است نکته قوت این روش این است که دیگر کانکشن نمی‌سازیم و تنها نتایج را در داخل DataSet ذخیره می‌کنیم. سپس می‌توانیم تغییرات مورد نظرمان را روی DataSet انجام دهیم و بعداً تغییرات را به دیتابیس ارسال کنیم.

اگر تا به حال از ابزارهای ویژوال استودیو برای وصل شدن به دیتابیس استفاده کرده‌اید از الان به بعد برای کوچک‌ترین مرحله از پروسه وصل شدن به دیتابیس باید کد بزنیم. حالا اصلاً چرا باید کد بزنیم، وقتی که چنین ابزارهایی وجود دارند؟ اگر این کار را به ویژوال استودیو بسپاریم صدها خط کد تولید می‌کند، این کدها برای بهینه سازی‌ها و مسائل پیشرفته دیتابیسی است که مثلاً در شرایط سخت (مثلاً حجم داده‌ای زیاد) استفاده می‌شود و برای یک سری کار ساده دیتابیسی اصلاً به آنها احتیاج نداریم. تعداد خط‌های کدی که می‌زنیم زیاد نیست ولی به مرور که جلو می‌رویم وارد مسائل پیشرفته تر می‌شویم. به عنوان منبع داده از SQL Server 2008 Express استفاده می‌کنیم و مثال‌های ما با این فرض جلو می‌رود که آن را نصب کرده‌اید و جدول‌ها و دیتابیس مثال را هم ساخته‌اید.

پرس و جو در دیتابیس: روش متصل (Connected)

Query زدن یا پرس و جو یعنی رویه استخراج اطلاعات از یک منبع اطلاعاتی مانند دیتابیس. مثلاً به یک دستور select از زبان SQL که از دیتابیس چند ردیف اطلاعات دریافت می‌کند Query می گویند و به آن عمل Query زدن می گویند. در ادامه یاد می‌گیرید که چگونه با استفاده از کلاس‌های ADO.NET برای پرس و جوی داده‌ها استفاده کنیم. در ادامه ما روش متصل برای استخراج یا واکشی داده‌ها از دیتابیس را توضیح می‌دهیم برای این کار ابتدا یک برنامه ویندوزی با نام QueryingDataBaseConnected ایجاد می‌کنیم. یک listBox به روی فرم قرار داده و اسم آن را studentsListBox می‌گذاریم:



دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم. حالا بر روی فرم دابل کلیک می‌کنیم تا کنترل کننده رویداد (که از این به بعد به آن event handler می گوییم) برای رویداد load فرم تولید شود .حواستان باشد که فضای نام لازم را به ابتدای برنامه اضافه کنید این کار را با کد زیر انجام می‌دهیم.

```
using System.Data.SqlClient;
```

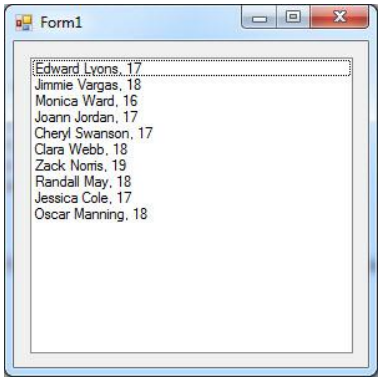
در رویداد Load فرم کد زیر را می‌نویسیم:

```
1  private void Form1_Load(object sender, EventArgs e)
2  {
3      SqlConnection connection = new SqlConnection();
4      connection.ConnectionString =   @"Data Source=(LocalDB)\MSSQLLocalDB;"
5                                    + @"AttachDbFilename=|DataDirectory|\University.mdf;"
6                                    +  "Integrated Security=True;"
7                                    +  "Connect Timeout=30";
8
9
10     SqlCommand command  = new SqlCommand();
11     command.CommandText = "SELECT FirstName, LastName, Age FROM Students";
12     command.Connection  = connection;
13
14     connection.Open();
15
16     SqlDataReader reader;
17     reader = command.ExecuteReader();
18
19     while (reader.Read())
20     {
21         string firstName = reader["FirstName"].ToString();
22         string lastName  = reader["LastName"].ToString();
23         int    age       = Convert.ToInt32(reader["Age"]);
24
25         studentsListBox.Items.Add(String.Format("{0} {1}, {2}", firstName, lastName, age));
26     }
27
28     reader.Close();
29
30     connection.Close();
31 }
```

همانطور که در کد بالا مشاهده می‌کنید، ابتدا لازم است که مسیر دیتابیس را مشخص کرده و با آن ارتباط برقرار کنیم. این کار را در خطوط ۳-۷ انجام داده‌ایم. ابتدا یک شیء از کلاس SqlConnection ایجاد کرده و سپس در خط ۴-۷ با استفاده از خاصیت ConnectionString مسیر بانک که همان رشته اتصال است را، به برنامه معرفی می‌کنیم. بعد از این کار، لازم است که دستوراتی را که می‌خواهیم بر روی بانک انجام دهیم را مشخص کنیم. این کار را با ستفاده از کلاس SqlCommand انجام می‌دهیم. ابتدا در خط ۱۰ یک شیء از این کلاس ایجاد می‌کنیم و سپس در خط ۱۱ و با استفاده از خاصیت CommandText این کلاس دستوری را که می‌خواهیم بر روی بانک انجام دهیم را مشخص می‌کنیم. در خط ۱۲ هم با استفاده از Connection مربوط به کلاس SqlCommand مشخص می‌کنیم که این کدها قرار است بر روی کدام بانک انجام شوند.

تا کنون کار خاصی انجام نداده‌ایم و فقط مسیر بانک و دستوراتی که قرار است بر روی بانک انجام شوند را، مشخص کرده‌ایم. حال برای اعمال دستورات بر روی بانک لازم است که ارتباط را برقرار (باز) کنیم. این کار را در خط ۱۴ و با استفاده از متد Open() انجام داده‌ایم. برای خواندن اطلاعات هم از کلاس SqlDataReader استفاده می‌کنیم. ابتدا یک شیء از این کلاس ایجاد می‌کنیم (خط ۱۶). در خط ۱۷ با فراخوانی متد ExecuteReader() از کلاس SqlCommand دستورات خط ۱۱ را بر روی بانک انجام داده و سپس نتایج به دست آمده را در شیء ایجاد شده از کلاس SqlDataReader قرار می‌دهیم. حال که نتایج در داخل شیء reader قرار گرفت، با استفاده از متد Read() تست می‌کنیم که آیا

رکوردی تحت تأثیر دستور خط ۱۱ قرار گرفته است یا نه؟ این متد اگر رکوردی تحت تأثیر قرار گرفته باشد، مقدار true و در غیر اینصورت مقدار false را بر می‌گرداند. در داخل بدنه دستور while هم ستون‌هایی را که دوست داریم در داخل ListBox نمایش می‌دهیم:



این برنامه را می‌توان ساده تر هم کرد این کار را می‌توان با استفاده از سازنده‌های SqlConnection و SqlCommand انجام داد یعنی مستقیماً می‌توان رشته اتصال را در سازنده کلاس SqlConnection قرار داد:

```
SqlConnection connection = new SqlConnection(
                                        @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                      + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                      +  "Integrated Security=True;"
                                      +  "Connect Timeout=30"
                                         );
```

و خطوط ۱۰ تا ۱۲ را به صورت زیر با هم ادغام کرد:

```
SqlCommand command = new SqlCommand("SELECT FirstName, LastName, Age FROM Students", connection);
```

ما می‌توانیم در داخل using یک کانکشن ایجاد کنیم. در مورد مزیت این روش گفتیم که وقتی بلاک using تمام شود connection بسته می‌شود.

```
using (SqlConnection connection = new SqlConnection())
{
    connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                + "Integrated Security=True;"
                                + "Connect Timeout=30";


    SqlCommand command  = new SqlCommand();
    command.CommandText = "SELECT FirstName, LastName, Age FROM Students";
    command.Connection  = connection;

    connection.Open();

    SqlDataReader reader;
    reader = command.ExecuteReader();

    while (reader.Read())
    {
        string firstName = reader["FirstName"].ToString();
        string lastName  = reader["LastName"].ToString();
```

```
        int    age       = Convert.ToInt32(reader["Age"]);

        studentsListBox.Items.Add(String.Format("{0} {1}, {2}", firstName, lastName, age));
    }

    reader.Close();
}
```

در مثال بالا بعد از اتمام بلاک connection از بین می‌رود. هنگام کار با دیتابیس ممکن است استثناءهایی رخ دهد. برای کنترل این خطاها بهتر است که از دستور try catch finally استفاده کنید. برای مثال تصور کنید که connection string یعنی مسیری که بانک در آنجا قرار دارد، اشتباه است، در نتیجه exception رخ می‌دهد و ما باید آن را کنترل کنیم. بنابراین از آنجایی که ممکن است در هنگام ارتباط با بانک خطا رخ دهد بهتر است که DbConnection.Open را در بلاک try بگذاریم چون که از زمان باز کردن کانکشن به بعد در هر خط ممکن است که exception رخ دهد.

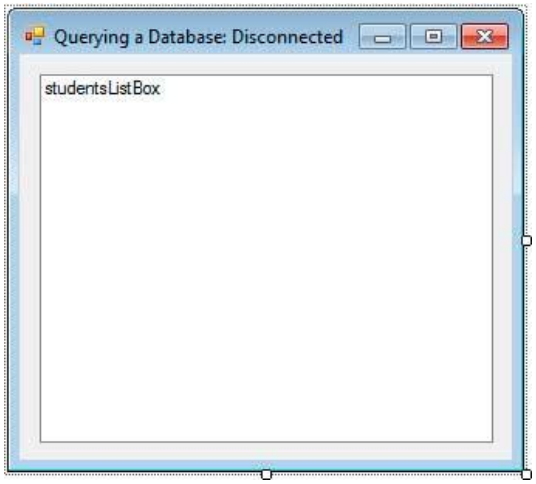
```
SqlConnection connection = new SqlConnection();
SqlCommand    command    = new SqlCommand();
SqlDataReader reader;

private void Form1_Load(object sender, EventArgs e)
{
    connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                + "Integrated Security=True;"
                                + "Connect Timeout=30";

    command.Connection  = connection;
    command.CommandText = "SELECT FirstName, LastName, Age FROM Students";

    try
    {
        connection.Open();

        reader = command.ExecuteReader();
        while (reader.Read())
        {
            string firstName = reader["FirstName"].ToString();
            string lastName  = reader["LastName"].ToString();
            int    age       = Convert.ToInt32(reader["Age"]);

            studentsListBox.Items.Add(String.Format("{0} {1}, {2}", firstName, lastName, age));
        }
    }
    catch (SqlException ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        reader.Close();
        connection.Close();
    }
}
```

در قسمت catch هم SqlException را قرار می‌دهیم و به همان شکل که در کد مشاهده می‌کنید، اگر استثنایی رخ دهد، آن را در MessageBox نمایش می‌دهیم. در قسمت finally هر کدی قرار بگیرد، حتی اگر exception رخ دهد، باز هم اجرا می‌شود. بنابراین منطقی

هست که کد بستن connection را در این قسمت قرار دهیم، بنابراین در هر شرایطی کانکشن بسته می‌شود. سعی کنید هنگام کار با دیتابیس، حتماً از این روش استفاده کنید تا بتوانید exception ها را بگیرید.

پرس و جو در دیتابیس: روش غیر متصل (Disconnected)

ما می‌توانیم عمل پرس و جو در دیتابیس را بدون باز نگه داشتن کانکشن انجام دهیم. برای این کار باید از کلاس‌های DataAdapter و DataSet استفاده کنیم. DataAdapter عمل پرس و جو در دیتابیس را انجام می‌دهد و نتایج را در DataSet ذخیره می‌کند. بعد می‌توانیم به تمام رکوردهای برگردانده شده با استفاده از خاصیت Row که در DataTable وجود دارد یا با استفاده از نام ستون دسترسی داشته باشیم. حالا یک برنامه معمولی برای توضیح این مطلب می‌سازیم. یک برنامه ویندوز فرم می‌سازیم و نام آن را queryingDatabaseDisconnected می‌گذاریم. یکlistBox به روی فرم قرار داده و اسم آن را studentsListBox می‌گذاریم.



بر روی فرم دابل کلیک می‌کنیم تا کنترل کننده رویداد (Event Handler) مربوط به رویداد Load فرم ایجاد شود. یادتان باشد که حتماً فضای های نامی لازم را به فرم اضافه کنید. مانند زیر.

```
using System.Data;
using System.Data.SqlClient;
```

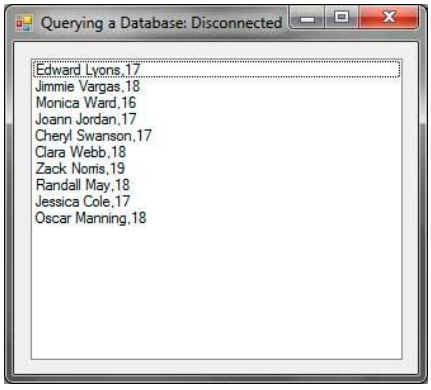
سپس کد زیر را در event Handler می‌نویسیم.

```
private void Form1_Load(object sender, EventArgs e)
{
    SqlConnection connection = new SqlConnection();
    connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                +  "Integrated Security=True;"
                                +  "Connect Timeout=30";

    SqlCommand command = new SqlCommand();
    command.CommandText = "SELECT FirstName, LastName, Age FROM Students";
    command.Connection = connection;


    DataSet dataset = new DataSet();
    SqlDataAdapter adapter = new SqlDataAdapter();
```

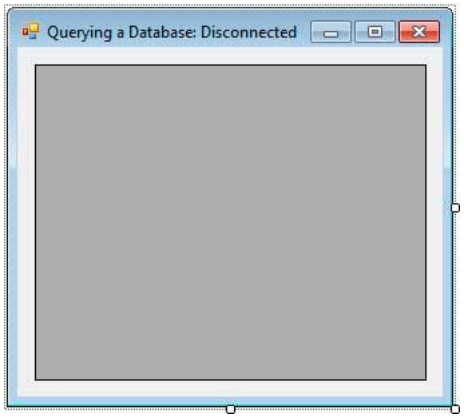
```
    adapter.SelectCommand = command;
    adapter.Fill(dataset, "Students");

    foreach (DataRow student in dataset.Tables["Students"].Rows)
    {
        studentsListBox.Items.Add(String.Format("{0} {1},{2}",
            student["FirstName"], student["LastName"], student["Age"]));
    }
}
```

درباره خطوط ۱۱-۳ در درس قبل توضیح دادیم. تفاوت روش متصل و غیر متصل از خط ۱۴ به بعد مشخص می‌شود. در روش متصل داده‌ها مستقیم از بانک و توسط کلاس SqlDataReader خوانده می‌شدند. ولی در روش غیر مستقیم، داده‌ها ابتدا به وسیله کلاس SqlDataAdapter در داخل DataSet قرار می‌گیرد و سپس عملیات بر روی آنها انجام می‌شود. همانطور که در کد بالا مشاهده می‌کنید، ابتدا یک شیء از DataSet در خط ۱۴ ایجاد می‌کنیم تا داده‌ها را در داخل آن بریزیم. به این نکته خیلی مهم توجه کنید که:

DataSet شامل جداولی است که به آنها DataTable می گویند. هر DataTable معادل یک جدول از دیتابیس اصلی است. پس نتایج پرس و جو در این DataTable قرار می‌گیرد. در ضمن به هر سطر DataTable یک DataRow و به هر ستون از آن یک DataColumn می گویند. در خط ۱۵ یک شیء از کلاس SqlDataAdapter ایجاد کرده و در خط ۱۶ دستوری که قرار است بر روی بانک انجام شود را به آن اختصاص می‌دهیم. این دستور همان دستوری است که در خط ۱۰ مشخص شده است. در خط ۱۷ دستور را با استفاده از متد Fill() اجرا می‌کنیم. این متد دو آرگومان می‌گیرد. آرگومان اول آن همان شیء DataSet و آرگومان دوم اسم جدول یا DataTable ی است که ما می‌خواهیم نتایج را برای ما نگه دارد. نام DataTable کاملاً اختیاری است. متد Fill() نتایج حاصل از اجرای دستور را در یک DataTable به نام Students ذخیره می‌کند. حال تمام کاری که لازم است انجام دهیم این است که، از یک حلقه foreach تمام رکوردهای این DataTable را پیمایش کنیم. همانطور که در داخل حلقه مشاهده می‌کنید، ابتدا یک شیء از کلاس DataRow ایجاد کرده و سپس با استفاده از خاصیت Rows مربوط به DataTable سطرهای جدول Students را یک به یک در داخل آن قرار داده و در خط ۲۱ چاپ می‌کنیم.



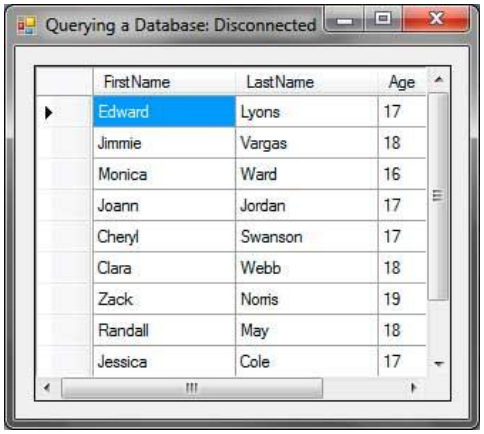
می‌توان این مثال را با استفاده از DataGridView انجام دهیم به جای اینکه از ListBox استفاده کنیم. برای این کار listbox را پاک و DataGridView را اضافه می‌کنیم و نام آن را studentsDataGridView می‌گذاریم:



و کد را به کد زیر تغییر می‌دهیم.

```
private void Form1_Load(object sender, EventArgs e)
{
    SqlConnection connection = new SqlConnection();
    connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                + "Integrated Security=True;"
                                + "Connect Timeout=30";

    SqlCommand command = new SqlCommand();
    command.CommandText = "SELECT FirstName, LastName, Age FROM Students";
    command.Connection = connection;


    DataSet dataset = new DataSet();
    SqlDataAdapter adapter = new SqlDataAdapter();
    adapter.SelectCommand = command;
    adapter.Fill(dataset, "Students");

    studentsDataGridView.DataSource = dataset.Tables["Students"];
}
```

همان طور که می‌بینید دیگر لازم نیست که از حلقه `foreach` استفاده کنید و فقط کافی است که جدول یا `DataSet` را به `dataSource` مربوط به `DataGridView` بدهید به همین راحتی. برنامه را که اجرا کردید، داده‌ها نمایش داده می‌شوند مانند شکل زیر:



در این درس یاد گرفتیم که چگونه با استفاده از یک روش غیر متصل با استفاده از `DataSet` از یک دیتابیس، استفاده کنیم.

اضافه کردن رکورد: روش متصل

اضافه کردن رکوردها (ثبت اطلاعات) با استفاده از کلاس‌های ADO.NET بسیار آسان است. در این درس می‌خواهیم در مورد روش متصل عمل ثبت یا همان insert، توضیح دهیم. حال اجازه دهید با استفاده از مثالی که در آن از دیتابیس University استفاده شده است، اطلاعاتی را در جدول Students وارد کنیم. یک برنامه ویندوزی ایجاد کرده و اسم آن را InsertingRecordConnected می‌گذاریم. فرم را مانند شکل زیر با lableها و TextBoxهای مربوطه پر می‌کنیم. اسم textBoxها را به firstNameTextBox، lastNameTextBox، genderTextBox، ageTextBox و AddressTextBox تغییر داده و هم چنین یک دکمه به اسم addButton بر روی فرم قرار می‌دهیم. دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم.

حال بر روی دکمه دابل کلیک می‌کنیم تا کد مربوط به رویداد Click آن تولید شود. سپس کدهای زیر را به آن اضافه می‌کنیم:

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3       SqlConnection connection = new SqlConnection();
4       connection.ConnectionString =
5                       @"Data Source=(LocalDB)\MSSQLLocalDB;"
6                     + @"AttachDbFilename=|DataDirectory|\University.mdf;"
7                     + "Integrated Security=True;"
8                     + "Connect Timeout=30";
9
10      SqlCommand command = new SqlCommand();
11      command.CommandText =
12                      "INSERT INTO Students " +
13                      "( FirstName,  LastName,  Gender,  Age,  Address) VALUES " +
14                      "(@FirstName, @LastName, @Gender, @Age, @Address)";
15
16      command.Connection = connection;
17
18      command.Parameters.AddWithValue("@FirstName", firstNameTextBox.Text);
19      command.Parameters.AddWithValue("@LastName" , lastNameTextBox.Text);
20      command.Parameters.AddWithValue("@Gender"   , genderTextBox.Text);
21      command.Parameters.AddWithValue("@Age"      , ageTextBox.Text);
22      command.Parameters.AddWithValue("@Address"  , addressTextBox.Text);
23
24      try
25      {
26          connection.Open();
27          int result = command.ExecuteNonQuery();
28          if (result > 0)
29              MessageBox.Show("Student successfully added!");
```

```
30          else
31              MessageBox.Show("Failed to add student!");
32      }
33      catch (SqlException ex)
34      {
35          MessageBox.Show(ex.Message);
36      }
37      finally
38      {
39          connection.Close();
40      }
41  }
```

درباره خطوط ۱۳-۳ کد بالا قبلاً توضیح داده‌ایم. در این خطوط ما محل دیتابیس و دستوراتی که قرار است بر روی آن اعمال شوند را، مشخص کرده‌ایم. تنها نکته ای که وجود دارد مربوط به خطوط ۱۱ و ۱۲ می‌باشد. در خط ۱۱ نام ستون‌های جدول و در خط ۱۲ هم مقادیری که قرار است در داخل ستون‌ها قرار بگیرند، مشخص شده‌اند. همانطور که مشاهده می‌کنید این مقادیر با علامت @ شروع شده‌اند. سؤال اینجاست که این مقادیر چه هستند؟ این مقادیر همان مقادیری هستند که کاربر از طریق TextBox ها وارد می‌کند. به خطوط ۱۹-۱۵ توجه کنید.

کلاس SqlCommand دارای یک خاصیت به نام Parameters می‌باشد. این خاصیت یک متد به نام AddWithValue() دارد. متد AddWithValue() دو آرگومان قبول می‌کند. آرگومان اول همان مقادیری هستند که با @ شروع می‌شوند و آرگومان دوم هم نام TextBox مربوطه است. در کل منظور از این خطوط این است که مثلاً مقدار فلان TextBox را در فلان پارامتر خط ۱۲ قرار بده. در داخل try catch ارتباط با بانک را برقرار کرده و دستور SQL را اجرا می‌کنیم. این کار را با استفاده از متد SqlCommand.ExecuteNonQuery() انجام می‌دهیم (خط ۲۲). این متد برای دستورهایی نظیر insert، update، delete به کار می‌رود که مقدار برگشتی ندارند. این متد یک عدد صحیح را به عنوان نتیجه بر می‌گرداند که نشان دهنده تعداد ردیف‌هایی است که تحت تأثیر این دستور قرار گرفته‌اند. بدیهی است که اگر ۰ برگرداند یک جای کار مشکل دارد. بنابراین این مساله را در خط‌های ۲۳ تا ۲۶ مورد بررسی قرار داده‌ایم. در قسمت catch تلاش می‌کنیم که خطاهای احتمالی را چاپ کنیم که در درس قبل توضیح دادیم و در finally بستن connection را قرار می‌دهیم چرا که حتی در حالتی که exception رخ دهد قسمت finally باز هم اجرا می‌شود و این برای ما مهم است که در هر صورت ارتباط قطع شود. برنامه را اجرا کرده و مانند شکل textboxها را پر می‌کنیم و دکمه add را می‌زنیم.

نتیجه ای که می‌بینیم به صورت زیر است (البته اگر مراحل را به خوبی و به درستی دنبال کرده باشید):

در درس به اضافه کردن رکورد به روش غیر متصل می‌پردازیم.

اضافه کردن رکورد: روش غیر متصل

اضافه کردن رکورد به روش غیر متصل تا حدی با اضافه کردن به روش متصل متفاوت است. این جا ما احتیاج داریم تا از کلاس‌های`DataAdapter` و `DataSet` استفاده کنیم. در این روش ابتدا تغییرات مان را بر روی `DataTable` و سپس تغییرات `DataTable` را بر روی دیتابیس اصلی اعمال می‌کنیم. پس اگر بخواهیم یک ردیف به دیتابیس اضافه کنیم، ابتدا باید یک ردیف به `DataTable` اضافه کنیم و بعد از آن تغییرات را به دیتابیس بفرستیم. حال اجازه دهید که در عمل چگونگی اضافه کردن یک ردیف به دیتابیس را به شما آموزش دهیم. برای انجام دادن این مثال‌ها یک برنامه ویندوزی جدیدی می‌سازیم و نام آن را `InsertingRecordsDisconnected` می‌گذاریم. `label`ها و `textBox` ها را به صورت زیر بر روی فرم قرار داده:

و آنها را هم به ترتیب از بالا به پایین به صورت زیر نام گذاری کنید:

Label	TextBox
First Name	firstNameTextBox
Last Name	lastNameTextBox
Gander	genderTextBox
Age	ageTextBox
Address	addressTextBox

دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم. یک دکمه هم به اسم AddButton بر روی فرم قرار دهید. بر روی دکمه دابل کلیک می‌کنیم تا کد مربوط کنترل کننده رویداد Click را تولید کند. همچنین باید فضاهای نام System.Data و System.Data.SqlClient را به قسمت فضاهای نامی اضافه کنیم.

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3       SqlConnection connection = new SqlConnection();
4       connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
5                                   + @"AttachDbFilename=|DataDirectory|\University.mdf;"
6                                   + "Integrated Security=True;"
7                                   + "Connect Timeout=30";
8
9       SqlCommand command  = new SqlCommand();
10      command.CommandText = "SELECT * FROM Students";
11      command.Connection  = connection;
12
13      DataSet dataset        = new DataSet();
14      SqlDataAdapter adapter = new SqlDataAdapter();
15      adapter.SelectCommand  = command;
16      adapter.Fill(dataset, "Students");
17
18      DataRow row = dataset.Tables["Students"].NewRow();
19
20      row["FirstName"] = firstNameTextBox.Text;
21      row["LastName"]  = lastNameTextBox.Text;
22      row["Gender"]    = genderTextBox.Text;
23      row["Age"]       = Int32.Parse(ageTextBox.Text);
24      row["Address"]   = addressTextBox.Text;
25
26      dataset.Tables["Students"].Rows.Add(row);
27
28      SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
29
30      try
31      {
32          int result = adapter.Update(dataset, "Students");
33
34          if (result > 0)
35              MessageBox.Show("Success!");
36          else
37              MessageBox.Show("Failed!");
38      }
39      catch (SqlException ex)
40      {
41          MessageBox.Show(ex.Message);
42      }
43  }
```

در کد بالا و در خطوط ۷-۳ ابتدا رشته اتصال با بانک را مشخص می‌کنیم. در خطوط ۱۱-۹ هم دستوراتی که قرار است به بانک اعمال شود را، می‌نویسیم. در خط ۱۳ یک DataSet و در خط ۱۴ یک شیء SqlDataAdapter ایجاد می‌کنیم. کلاس SqlDataAdapter دارای یک متد به نام Fill() است که با استفاده از آن دستوراتی که در خط ۱۰ مشخص شده‌اند را، بر روی DataSet اعمال می‌کند. مثلاً در خط ۱۶ کد بالا، یک جدول در داخل DataSet به نام Students ایجاد کرده و داده‌های جدول بانک اصلی را در داخل آن می‌نویسیم. تا اینجای کار یعنی از خطوط ۱۶-۳ ما فقط یک نمونه از دیتابیس اصلی و جدول آن را، به حافظه Ram منتقل کرده‌ایم. در مرحله بعد می‌خواهیم تغییراتی در DataSet بدهیم.

همانطور که گفتیم DataSet برنامه ما دارای یک DataTable به نام Students است. در این برنامه ما یک ردیف به آن اضافه می‌کنیم. این کار را در خطوط ۲۶-۱۸ انجام داده‌ایم. در خط ۱۸ با استفاده از متد ()NewRow یک ردیف ایجاد می‌شود. در خطوط ۲۴-۲۰ مشخص کرده‌ایم که مقادیری که قرار است در داخل ستون‌های این سطر قرار بگیرد، از TextBox ها و توسط کاربر وارد می‌شود. و در خط ۲۶ هم گفته‌ایم که، این ردیف نهایتاً به DataTable ی به نام Students اضافه شود. تا اینجا ردیفی که می‌خواستیم اضافه کنیم را به dataset اضافه کرده‌ایم. حالا می‌خواهیم آن را به دیتابیس اضافه کنیم. نکته اصلی کد بالا استفاده از کلاس SqlCommandBuilder در خط ۲۸ است. سازنده این کلاس یک شیء از کلاس SqlDataAdapter دریافت کرده و به صورت خودکار دستورات حذف، اضافه و ویرایش را تولید می‌کند. با فراخوانی متد ()Update در خط ۳۲، از آنجاییکه در کد بالا ما یک ردیف به DataTable اضافه کرده‌ایم، این کلاس به طور خودکار عمل اضافه کردن را تشخیص داده و دستور INSERT را تولید و بر روی دیتابیس اصلی اعمال می‌کند. متد ()Update یک مقدار عددی برمی گرداند که این عدد تعداد ردیف‌هایی است که در دیتابیس دستخوش تغییر شده‌اند. بعد از اجرای برنامه صفحه زیر را می‌بینید:

که در صورت پر کردن فیلدها مانند بالا و زدن دکمه AddButton صفحه زیر را مشاهده می‌کنید و رکورد به درستی در دیتابیس اضافه می‌شود :

پاک کردن یک رکورد: روش متصل

الان تصمیم داریم یک رکورد را با استفاده از روش متصل به دیتابیس پاک کنیم. برای انجام این کار هم مانند اضافه کردن یک رکورد به دیتابیس است از متد DbCommand.ExecuteNonQuery() استفاده می‌کنیم. حالا یک برنامه می‌سازیم که Id مربوط به Student را بگیرد و رکورد متناظر با آن را در دیتابیس حذف کند. برای همین یک برنامه ویندوز فرم می‌سازیم و اسم آن را DeletingRecordConnected می‌گذاریم و

مانند شکل زیریک Lable و یک textbox و همچنین دکمه ای هم برای پاک کردن بر روی فرم قرار می‌دهیم. نام textbox را StudentIdTextBox و نام دکمه را بهDeleteButton تغییر می‌دهیم.

دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم. بر روی دکمه دابل کلیک کرده تا کنترل کننده رویداد Click تولید شود. سپس کد زیر را به آن اضافه می‌کنیم.

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3       SqlConnection connection = new SqlConnection();
4       connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
5                                   + @"AttachDbFilename=|DataDirectory|\University.mdf;"
6                                   + "Integrated Security=True;"
7                                   + "Connect Timeout=30";
8
9       SqlCommand command  = new SqlCommand();
10      command.CommandText = "DELETE FROM Students WHERE StudentID = @StudentID";
11      command.Connection  = connection;
12
13      command.Parameters.AddWithValue("@StudentID", studentIdTextBox.Text);
14
15      try
16      {
17          connection.Open();
18          int result = command.ExecuteNonQuery();
19          if (result > 0)
20              MessageBox.Show("Student was removed!");
21          else
22              MessageBox.Show("Can't find student.");
23      }
24      catch (SqlException ex)
25      {
26          MessageBox.Show("An error has occured.");
27      }
28      finally
29      {
30          connection.Close();
31      }
32  }
```

یادتان باشد که حتماً فضاهای نامی لازم را به آن اضافه کنید:

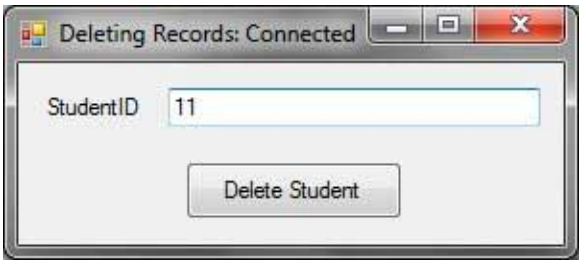
```
System.Data.SqlClient
```

درباره خطوط ۱۱-۳ در درس‌های قبل توضیح داده‌ایم. تنها نکته ای که وجود دارد مربوط به خط ۱۰ است. در این دستور ما مشخص کرده‌ایم که @studentID یک پارامتر است و در خط ۱۳ گفته‌ایم که مقدار آن از studentIdTextBox (همان textbox ای که روی فرم گذاشته‌ایم) دریافت می‌شود. همان طور که در درس‌های قبل توضیح دادیم در کلاس SqlCommand یک خصوصیت (property) به اسم Parameters وجود دارد که به متد AddWithValue() می‌گوید که فلان پارامتر را در پرس و جویی که می‌خواهیم به دیتابیس بفرستیم با چه مقداری جایگزین کند.

مثلاً در خط ۱۰ گفته‌ایم که در داخل دستور SQL که به CommandText دادیم باید مقدار @studentID با مقدار StudentIdTextBox.text جایگزین شود (خصوصیت Text در یک textbox مقداری که در آن وارد می‌شود را در بر می‌گیرد) یعنی آنچه که در TextBox داخل فرم می‌نویسیم توسط برنامه گرفته می‌شود و دقیقاً جای @studentID قرار می‌گیرد. در خط ۱۸ این دستور را اجرا می‌کنیم (یعنی این دستورSQL به سمت دیتابیس فرستاده می‌شود) و نتیجه این دستور یک مقدار عددی است که تعداد ردیف‌هایی است که در دیتابیس پاک شده‌اند.

اگر مقدار برگشت داده شده از متد ExecuteNonQuery() بزرگ‌تر از ۰ باشد یعنی رکوردی تحت تأثیر دستور SQL قرار گرفته و حذف شده است. در این مثال حداقل یک رکورد باید پاک شود مگر اینکه آن Id که در textbox وارد می‌کنیم در دیتابیس وجود نداشته باشد که در این صورت ۰ برگردانده می‌شود. در قسمت catch ما مشخص می‌کنیم که اگر exception ای رخ دهد آن را چاپ می‌کنیم.

در قسمت finally هم connection را می‌بندیم (اینکه چرا این کار را در finally انجام می‌دهیم همان طور که در درس‌های قبلی گفتیم finally یک قابلیت C# است که اگر کدی در آن قرار دهیم این کد حتماً اجرا می‌شود یعنی حتی اگر Exception ای رخ دهد و برنامه بسته شود این کدها اجرا می‌شود. از آنجا که connection حتماً باید بسته شود. بهترین جا برای گذاشتن این کد همان finally است). حالا می‌توانیم برنامه را اجرا کنیم. بعد از اجرا شناسه (id) یک ردیف که می‌خواهیم از دیتابیس حذف شود را، در textbox وارد می‌کنیم و با زدن دکمه Delete Student آن ردیف پاک می‌شود.

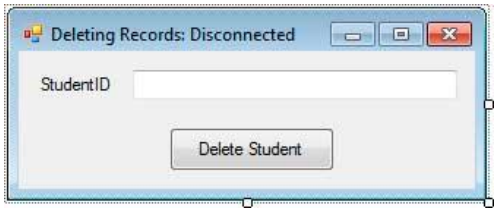


و پیام زیر برای شما نمایش داده می‌شود.

پاک کردن یک رکورد – روش غیر متصل

پاک کردن یک رکورد به روش غیر متصل مانند وارد کردن یک رکورد با استفاده از DataSet و DataAdapter (که در درس‌های قبل توضیح داده شد) به دیتابیس است. ولی به جای آنکه یک dataRow را به DataTable ای که در DataSet استفاده می‌شود اضافه کنیم آن را از DataTable حذف می‌کنیم. موارد زیر مراحلی است که باید در حین انجام کار در این مثال انجام بدهیم. برای این کار یک برنامه ساده می‌سازیم که آن Id که قصد حذف کردن آن را داریم را از کاربر بگیرد. یک برنامه ویندوز فرم به اسم DeletingRecordDisconnected می‌سازیم و یک

label، textbox و یک دکمه به آن اضافه می‌کنیم. اسم آنها را به ترتیب studentIdTextbox و deleteButton می‌گذاریم و دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم.



بر روی دکمه دو بار کلیک می‌کنیم تا کد مربوط به کنترل کننده رویداد Click ایجاد شود. سپس کد زیر را اضافه می‌کنیم.

```
private void deleteButton_Click(object sender, EventArgs e)
{
    SqlConnection connection = new SqlConnection();
    connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
                                + @"AttachDbFilename=|DataDirectory|\University.mdf;"
                                + "Integrated Security=True;"
                                + "Connect Timeout=30";

    SqlCommand command = new SqlCommand();
    command.CommandText = "SELECT * FROM Students";
    command.Connection = connection;

    SqlDataAdapter adapter = new SqlDataAdapter();
    adapter.SelectCommand = command;
    DataSet dataset = new DataSet();
    adapter.Fill(dataset, "Students");

    foreach (DataRow row in dataset.Tables["Students"].Rows)
    {
        if (row["StudentID"].ToString() == studentIdTextBox.Text)
        {
            row.Delete();
        }
    }

    SqlCommandBuilder builder = new SqlCommandBuilder(adapter);

    try
    {
        int result = adapter.Update(dataset, "Students");

        if (result > 0)
            MessageBox.Show("Success!");
        else
            MessageBox.Show("Failed!");
    }
    catch (SqlException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

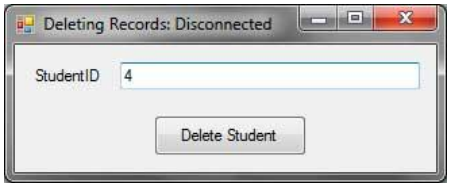
فضاهای نام هم فراموش نشود:

```
using System.Data.SqlClient;
```

درباره خطوط ۳ تا ۱۶ این توضیح تکراری را می‌دهیم که با بانک ارتباط را برقرار می‌کنیم (خطوط ۷-۳) سپس یک شیء از کلاس SqlCommand ایجاد می‌کنیم تا دستوراتی که قرار است بر روی بانک اعمال شوند، را در داخل آن بگذاریم. در خط ۱۳ یک شیء از کلاس SqlDataAdapter ایجاد کرده و دستوراتی که در خط ۱۰ مشخص کرده‌ایم را از کلاس SqlCommand تحویل می‌گیریم. و در خط ۱۶ بر روی Dataset اعمال می‌کنیم. در این خط همانطور که مشاهده می‌کنید یک جدول به نام Students در داخل DataSet ایجاد کرده‌ایم و تمام داده‌های جدول Students از بانک University را در داخل آن قرار داده‌ایم.

حال که مقادیر یک جدول از دیتابیس را در اختیار داریم می‌توانیم رکوردی را حذف کنیم. ابتدا لازم است که به دنبال یک رکورد برای حذف بگردیم. با استفاده از حلقه foreach در میان رکوردهای Dataset می‌گردیم (خط ۱۸) و در داخل شرط if می گوییم که در صورتی که مقدار ستون StudentID یکی از سطرهای DataTable برابر با مقدار وارد شده در textbox بود آن سطر را حذف کن. برای حذف سطر مذکور از متد ()Delete کلاس DataRow استفاده می‌کنیم. در واقع این ردیف از DataTable حذف شده است و باید بعداً از دیتابیس هم حذف شود.

در خط ۲۸ از آنجاییکه ما شیء adapter را به کلاس SqlCommandBuilder داده‌ایم و در خطوط قبل عملیات حذف انجام گرفته است، کلاس SqlCommandBuilder دستور حذف یا DELETE را به طور خودکار تولید کرده و در داخل شیء adapter قرار می‌دهد. و در نهایت در خط ۳۰ با فراخوانی متد ()Update و چک کردن مقدار برگشتی از این متد در خطوط بعد، تغییرات را بر روی دیتابیس اصلی اعمال می‌کنیم. برنامه را اجرا می‌کنیم و صفحه زیر را می‌بینیم.

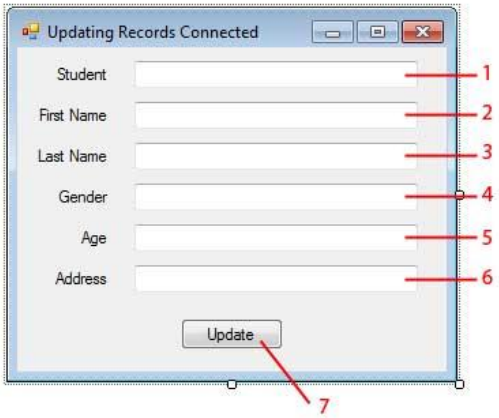


به عنوان مثال عدد ۴ را به عنوان Id وارد می‌کنیم. اگر این مقدار به عنوان StudentID در دیتابیس وجود داشته باشد صفحه زیر را می‌بینیم.

بروزرسانی رکوردها: روش متصل

در این درس نگاهی به این قضیه می‌اندازیم که چگونه می‌توان یک رکورد را در دیتابیس بروزرسانی کرد. مانند درج و حذف رکوردها، برای بروزرسانی آنها هم باید از DbCommand.ExecuteNonQuery استفاده کرد. حالا یک نگاه کلی به برنامه ای که قرار است بنویسیم می‌اندازیم. ماStudentID و موارد جدید برای جایگزین شدن به فیلدهای قبلی مربوط به این id را می‌نویسیم. در برنامه‌های واقعی تر این موارد ابتدا باید به شما نمایش داده شوند و شما انتخاب کنید که کدام فیلد بروزرسانی شود. ولی برای سادگی در این برنامه ما فقط به شما نشان می‌دهیم که چه اطلاعاتی وجود

دارد و سپس تغییرات لازم را می‌دهیم و به دیتابیس می‌فرستیم. یک برنامه ویندوزی می‌سازیم و اسم آن را UpdatingRecordsConnected می‌گذاریم.



جدول زیر که می‌بینید اسامی ابزارهایی است که در فرم گذاشته‌ایم. مثلاً شماره ۱ که textbox اولی است به اسم textBoxStudentID ذخیره شده است.

Label	Name
1	textBoxStudentID
2	textBoxFirstName
3	textBoxLastName
4	textBoxGender
5	textBoxAge
6	textBoxAddress
7	buttonUpdate

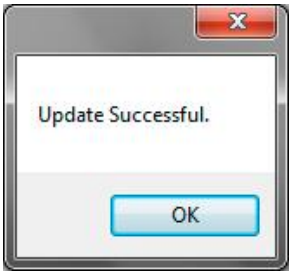
دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم. حالا بر روی دکمه دابل کلیک می‌کنیم تا کنترل کننده رویداد مربوط به رویداد Click آن تولید شود، سپس کد زیر را به آن اضافه می‌کنیم.

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3       SqlConnection connection = new SqlConnection();
4       connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
5                                   + @"AttachDbFilename=|DataDirectory|\University.mdf;"
6                                   + "Integrated Security=True;"
7                                   + "Connect Timeout=30";
8
9       SqlCommand command = new SqlCommand();
10      command.CommandText = "UPDATE Students SET "
11                          + "FirstName = @FirstName, LastName = @LastName,"
12                          + "Gender = @Gender, Age = @Age, Address = @Address "
13                          + "WHERE StudentID = @StudentID";
14      command.Connection = connection;
15
```

```
16      command.Parameters.Clear();
17      command.Parameters.AddWithValue("@FirstName", textBoxFirstName.Text);
18      command.Parameters.AddWithValue("@LastName" , textBoxLastName.Text );
19      command.Parameters.AddWithValue("@Gender"   , textBoxGender.Text   );
20      command.Parameters.AddWithValue("@Age"      , textBoxAge.Text      );
21      command.Parameters.AddWithValue("@Address"  , textBoxAddress.Text  );
22      command.Parameters.AddWithValue("@StudentID", textBoxStudentID.Text);
23
24      try
25      {
26          connection.Open();
27          int result = command.ExecuteNonQuery();
28
29          if (result > 0)
30              MessageBox.Show("Update Successful!");
31          else
32              MessageBox.Show("Update Failed!");
33      }
34      catch (SqlException ex)
35      {
36          MessageBox.Show("An error occured.");
37      }
38      finally
39      {
40          connection.Close();
41      }
42  }
```

کد بالا زیاد نیاز به توضیح ندارد. درباره خطوط ۳-۱۴ که در درس‌های قبل توضیح داده‌ایم. فقط درباره دستور SQL خطوط ۱۳-۱۰ توضیح می‌دهیم. این دستور SQL در قسمت WHERE (خط ۱۳) به وسیله StudentId به دیتابیس می‌گوید که کدام دانشجو را بروزرسانی بکند و که مقادیر موجود در ستون‌های FirstName، LastName، Gender، Age و Address را با مقادیر که کاربر از جعبه‌های متن وارد می‌کند، جایگزین می‌کند. در خط ۱۶ با استفاده از متد Clear() تمام پارامترهای احتمالی از عملیات‌های قبلی را پاک می‌کنیم، قبل از اینکه برای آن پارامترهای جدید تعریف کنیم. از آن جا که دستور update در SQL یک Non-Query محسوب می‌شود (یعنی هیچ رکوردی به عنوان نتیجه بر نمی‌گرداند) ما از متد ExecuteNonQuery() برای بروزرسانی استفاده می‌کنیم. مقدار بازگشتی از این متد تعداد رکورهای تغییر کرده در اثر دستور SQL خطوط ۱۳-۱۰ است. این مقدار در متغیر result ذخیره می‌کنیم و در قسمت شرط با مقدار ۰ مقایسه می‌کنیم. اگر مقدار آن بزرگ‌تر از ۰ بود یعنی بروزرسانی صورت گرفته و در دستور if پیام مربوطه چاپ می‌شود در غیر این صورت هم پیام مربوطه چاپ می‌شود. حال پروژه را اجرا می‌کنیم و StudentId مربوط به آن دانجشجویی که می‌خواهیم اطلاعاتش را تغییر دهیم با اطلاعات جدیدش وارد می‌کنیم.

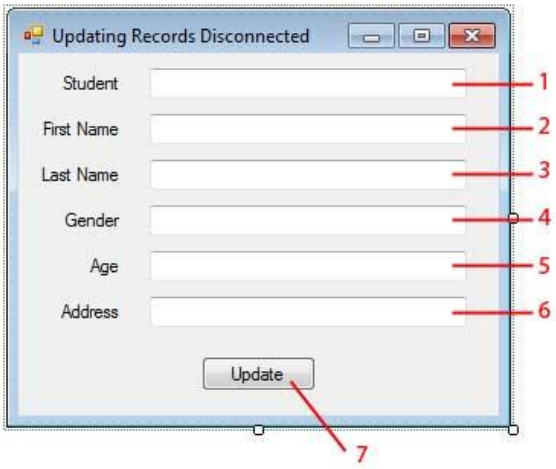
اگر هیچ خطایی وجود نداشته باشد پیام موفقیت به ما فرستاده می‌شود مثل زیر.



در این درس هدف این بود که نشان داده شود که چگونه یک رکورد در دیتابیس را بروزرسانی کنیم. ولی بروزرسانی یک رکورد بدون اینکه مقدارش را بدانیم کار آسانی نیست. در آینده یک آموزش از همه چیزهایی که تا حالا گفته شده خواهیم داشت (اعم از پاک کردن، اضافه کردن و ...) ولی اول باید ببینیم که چگونه یک رکورد را به روش غیر متصل بروزرسانی کنیم.

بروزرسانی رکوردها: روش غیر متصل

در این درس به بروزرسانی دیتابیس به روش غیرمتصل می‌پردازیم. مانند اضافه کردن و پاک کردن رکوردها با استفاده از روش غیرمتصل، بروزرسانی رکوردها هم از کلاس‌های `DataAdapter` و `DataSet` و `CommandBuilder` استفاده می‌کند. در این درس می‌خواهیم بر اساس شناسه یک دانشجو (`StudentId`) اطلاعات آن را تغییر دهیم. یک چنین برنامه ای باید اول دانشجویان را به ما نشان دهد و سپس ما از بین آنها یکی را انتخاب کنیم و تغییر دهیم. ولی برای سادگی ما در این برنامه خودمان `StudentId` و اطلاعات جدید را می‌دهیم و در ادامه آنها را تغییر می‌دهیم. یک برنامه ویندوزی می‌سازیم و در فیلد اول شناسه دانشجویی که می‌خواهیم اطلاعاتش را تغییر دهیم و در فیلدهای بعدی اطلاعاتی که قرار است برای او ثبت شود، را می‌نویسیم. پس در فیلد اول باید `StudentId` را بنویسیم.



جدول زیر که می‌بینید اسامی TextBox و Label هایی است که در فرم گذاشته‌ایم. آنها را بر طبق شماره‌های بالا به صورت زیر نامگذاری می‌کنیم:

Label	TextBox	Label
1	textBoxStudentId	Student
2	textBoxFirstName	First Name
3	textBoxLastName	Last Name
4	textBoxGender	Gender
5	textBoxAge	Age
6	textBoxAddress	Address
7	updateButton	

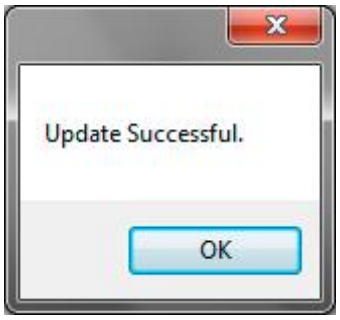
بعد از ساختن ظاهر برنامه، دیتابیس University را در کنار فایل exe برنامه یعنی پوشه Debug قرار می‌دهیم. بر روی دکمه دابل کلیک می‌کنیم تا کنترل کننده رویداد مربوط به رویداد Click تولید شود سپس کدهای زیر را در داخل آن می‌نویسیم:

```
1   private void updateButton_Click(object sender, EventArgs e)
2   {
3       SqlConnection connection = new SqlConnection();
4       connection.ConnectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;"
5                                   + @"AttachDbFilename=|DataDirectory|\University.mdf;"
6                                   + "Integrated Security=True;"
7                                   + "Connect Timeout=30";
8
9       SqlCommand command = new SqlCommand();
10      command.CommandText = "SELECT * FROM Students";
11      command.Connection = connection;
12
13      DataSet dataset = new DataSet();
14      SqlDataAdapter adapter = new SqlDataAdapter();
15      adapter.SelectCommand = command;
16      adapter.Fill(dataset, "Students");
17
```

```
18      foreach (DataRow row in dataset.Tables["Students"].Rows)
19      {
20          if (row["StudentID"].ToString() == textBoxStudentID.Text)
21          {
22              row["FirstName"] = textBoxFirstName.Text;
23              row["LastName"]  = textBoxLastName.Text;
24              row["Gender"]    = textBoxGender.Text;
25              row["Age"]       = textBoxAge.Text;
26              row["Address"]   = textBoxAddress.Text;
27          }
28      }
29
30      SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
31
32      try
33      {
34          int result = adapter.Update(dataset, "Students");
35
36          if (result > 0)
37              MessageBox.Show("Update Successful.");
38          else
39              MessageBox.Show("Update Failed.");
40      }
41      catch (SqlException ex)
42      {
43          MessageBox.Show(ex.Message);
44      }
45  }
```

درباره خطوط ۱۶-۳ توضیح نمی‌دهیم، چون قبلاً این کار را کرده‌ایم. در خط ۱۸ با استفاده از یک حلقه foreach تمام سطرهای DataTable را برای پیدا کردن سطری که مقدار ستون StudentId برابر مقداری که کاربر از طریق جعبه متن textboxStudentID وارد می‌کند پیمایش می‌کنیم. اگر همچین سطری وجود داشت تمامی ستون‌های دیگر این سطر را با مقادیر وارد شده از سایر textbox ها پر می‌کنیم. در خط ۳۰ یک شیء adapter را به کلاس SqlCommandBuilder ارسال می‌کنیم و اگر در خطوط ۲۸-۱۸ عملیات آپدیتی انجام شده باشد، کلاس SqlCommandBuilder دستور UPDATE را به طور خودکار تولید و به adapter می‌دهد تا در خط ۳۴ و با فراخوانی متد Update() این شیء و مقایسه مقدار برگشتی از این متد در خطوط بعد عملیات بروزرسانی را بر روی بانک اصلی اعمال کند. برنامه را که اجرا می‌کنیم. فرم زیر را می‌بینیم و studentID دانشجوی مورد نظر را که می‌خواهیم تغییرش بدهیم را وارد می‌کنیم و فیلدهای بعدی را هم آن جور که می‌خواهیم در دیتابیس ذخیره شود، پر می‌کنیم.

سپس دکمه را می‌زنیم اگر همه چیر موفقیت آمیز باشد پیام زیر را مشاهده می‌کنیم:



در این درس هدف این بود که نشان داده شود که چگونه یک رکورد در دیتابیس را بروزرسانی کنیم. ولی بروزرسانی یک رکورد بدون اینکه مقدارش را بدانیم کار آسانی نیست. در آینده یک آموزش از همه چیزهایی که تا حالا گفته شده خواهیم داشت (اعم از پاک کردن، اضافه کردن و. (...

اتصال به دیتابیس Access

با استفاده از ADO.NET می‌توان به دیتابیس‌های Access وصل شد و به داده‌های آن دسترسی پیدا کرد. با استفاده از Data provider ای به نام OLE DB و با استفاده از فضای نام System.Data.Oledb می‌توان به دیتابیس Access دسترسی پیدا کرد. در این آموزش یاد می‌گیریم که یک databaseaccess بسازیم و یک برنامه هم می‌سازیم که به آن متصل شود. در اینجا از نسخه Microstoft Access 2007 استفاده می‌کنیم.

Microsoft Access را باز می‌کنیم و در صفحه Home بر روی آیکون دیتابیس کلیک می‌کنیم یک صفحه ظاهر می‌شود که نام دیتابیس و آدرس آن را ذکر می‌کنید. اسم دیتابیس را Members.accdb بگذارید و آدرس آن را هر جایی که دوست دارید انتخاب کنید، ولی برای سادگی آن را در C:\root ذخیره می‌کنیم. بر روی create کلیک می‌کنیم و سپس به یک صفحه View جدید و با یک جدول جدید راهنمایی می‌شویم. حالا اینجا می‌توانیم ستون اضافه کنیم ولی راه حل ساده تر این است که به قسمت design برویم و در آنجا فیلدها و ستون‌ها را اضافه کنیم سپس یک صفحه باز می‌شود و از ما می‌خواهد که یک اسم برای جدول انتخاب کنیم. اسم آن را Members می‌گذاریم. تا زمانی که در قسمت design هستیم می‌توانیم هر فیلدی را می‌توانیم اضافه کنیم. فیلد پیشفرضی که می‌بینیم ID است و نوع آن هم autonumber است. Data type یا نوع داده به ما می‌گوید نوع اطلاعاتی که قرار است در اینجا ذخیره شود چیست؟ در ضمن یک کلید هم کنار آن مشاهده می‌کنیم، این به آن معنا است که این فیلد در نقش primary key عمل می‌کند. Primary key به ما این را می‌گوید که مقادیری که قرار است در این فیلد ذخیره شود باید به صورت یکتا باشد. نوع داده Autonumber به این معنی است که زمانی که یک رکورد اضافه می‌کنیم خود Access برای ما این فیلد را مقداردهی می‌کند. توضیحات ستون (column description) اختیاری است و برای ما مورد استفاده این فیلد را توضیح می‌دهد. شما می‌توانید آن را بدون مقداردهی رها کنید یا با استفاده از آن، هدف فیلد را مشخص کنیم. حالا فیلدهای زیر را به جدول اضافه می‌کنیم:

نام فیلد	نوع داده ای
FirstName	Text
LastName	Text

Age	Number

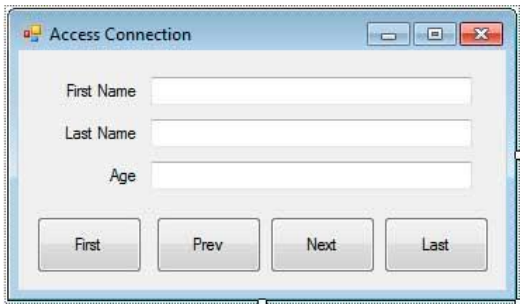
نوع داده‌ای Text به ما نشان می‌دهد که ما انتظار داریم که یک فیلد متنی در اینجا بنشیند و نوع داده‌ای Number یعنی انتظار داریم که مقدار داده‌ای عددی جایگزین آن باشد. وقتی این کارها تمام شد به DataSheet بر می‌گردیم برای این کار به View>DataSheet می‌رویم. حالا می‌توانیم جدول را تولید کنیم. مقادیر زیر را به آن اضافه می‌کنیم. فیلد ID به صورت اتوماتیک توسط Access تولید می‌شود.

FirstName	LastName	Age
John	Smith	21
Mark	Mayer	23
Alvin	Minsky	27
Vince	Roster	22
Ben	Marcus	18
Ellise	Fisher	25

حالا که ما یک سری اطلاعات در دیتابیس ذخیره کردیم، می‌توانیم برنامه‌ای هم بنویسیم که به دیتابیس وصل شده از آن اطلاعات را بخواند. ولی اول مطمئن شوید که دیتابیس را به درستی ساخته و ذخیره کرده‌اید.

پرس و جو در دیتابیس Access

در این درس ما یک برنامه برای پرس و جو و اتصال به یک منبع داده‌ای اکسس می‌سازیم. ویژوال استودیو را باز می‌کنیم و یک برنامه ویندوزی ایجاد کرده و نام پروژه را AccessConnection می‌گذاریم. سه Label و سه Textbox بر روی آن قرار می‌دهیم و نام برچسب‌ها را به Name، First Name و Age و خصوصیت اسم (name) مربوط به textboxها را به txtFirstName و txtLastName و txtAge تغییر می‌دهیم. چهار دکمه به این فرم اضافه می‌کنیم و اسم آنها را به btnFirst و btnNext و btnLast و btnPrev تغییر می‌دهیم.



بر روی این فرم دابل کلیک می‌کنیم تا کد زیر که مربوط به فرم است را برای ما بسازد.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace AccessConnection
```

```
{
    public partial class Form1 : Form
    {
        private OleDbConnection connection;
        private OleDbCommand command;
        private OleDbDataAdapter adapter;
        private DataSet dataset;
        private string firstname;
        private string lastname;
        private int age;
        private int position;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            connection = new OleDbConnection();
            command = new OleDbCommand();
            adapter = new OleDbDataAdapter();
            dataset = new DataSet();

            connection.ConnectionString =
                @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Members.accdb;" +
                "Persist Security Info=False";

            command.Connection = connection;
            command.CommandText = "SELECT * FROM Members";

            adapter.SelectCommand = command;

            try
            {
                adapter.Fill(dataset, "Members");
            }
            catch (OleDbException)
            {
                MessageBox.Show("Error occured while connecting to database.");
                Application.Exit();
            }

            ShowValuesOfRow(0);
        }

        private void ShowValuesOfRow(int pos)
        {
            DataRow row = dataset.Tables["Members"].Rows[pos];
            position = pos;

            firstname = row["FirstName"].ToString();
            lastname = row["LastName"].ToString();
            age = (int)row["Age"];

            txtFirstName.Text = firstname;
            txtLastName.Text = lastname;
            txtAge.Text = age.ToString();
        }
    }
}
```

ما از System.Data.Oledb به عنوان فضای نام استفاده می‌کنیم. این فضای نام شامل کلاس‌هایی برای کار کردن با دیتابیس Access، و همچنین کلاس DataSet، برای نگهداری اطلاعات و کار با آنها می‌باشد. OleDbConnection را برای ایجاد ارتباط با دیتابیس و گرفتن اطلاعات،

OleDbCommand را برای ساختن دستور مورد نظر، DataSet و یک سری فیلد برای نگهداری مقادیر دیتابیس درست می‌کنیم. فیلد position به ما می‌گوید که کدام سطر را باید در textbox نمایش دهیم. در داخل کنترل کننده رویداد مربوط به Load، اشیاء لازم برای وصل شدن به دیتابیس و سپس connectionString مربوط به OleDbConnection را مقداردهی می‌کنیم. این connection string اطلاعات لازم برای اتصال به دیتابیس را در اختیار ما قرار می‌دهد. همچنین دستور مربوط به Sql را در CommandText مشخص می‌کنیم. این دستور مشخص می‌کند که تمام اطلاعات و ردیف‌های دیتابیس Members را بگیریم.

در خط ۳۸ خصوصیت مربوط به SelectCommand مربوط به OleDbDataAdapter را با command مقداردهی می‌کنیم. از متد OleDbDataAdapter.Fill() برای اجرا کردن دستوری که در SelectCommand وجود دارد استفاده می‌کنیم تا اطلاعات را در DataSet مربوطه وارد کند، برای اینکه اگر استثنایی رخ داد بتوانیم آن را کنترل کنیم، این کد را در try ... catch وارد کرده‌ایم. از متد ShowValuesOfRow استفاده می‌کنیم که مقادیر آن ردیف یا ایندکس را نمایش دهیم. عدد صفر را به عنوان آرگومان به این تابع می‌فرستیم تا این تابع ردیف اول را به نشان دهد. در داخل متد با استفاده از خصوصیت Tables در شیء DataSet و با استفاده از خصوصیت Rows که مربوط به Tables است ردیف مربوطه را در می‌آوریم و یک DataRow می‌سازیم و به آن منتسب می‌کنیم. ما می‌خواهیم از جدول Members این مورد را استخراج کنیم که با استفاده از خود اسم جدول این کار را می‌کنیم. هم چنین می‌خواهیم آن ردیف از جدول را برداریم که ایندکس آن pos است که همه این موارد را در خط ۵۵ می‌بینیم. نتیجه در شیء DataRow که یک ردیف در DataTable را برای ما مشخص می‌کند را مشخص می‌کند سپس position را مقداردهی می‌کنیم. سپس مقدار هر ستون از DataRow به فیلد آن مقداردهی می‌شود و آن فیلد را به TextBox مربوطه منتصب می‌کنیم تا در فرم نمایش داده شود مانند شکل زیر.

حالا برای دکمه‌ها باید تعریف کنیم که چه کاری باید انجام دهند. اول بر روی btnFirst کلیک می‌کنیم و تا کد خود را به آن اضافه کنیم. کد زیر را به آن اضافه می‌کنیم.

```
private void btnFirst_Click(object sender, EventArgs e)
{
    ShowValuesOfRow(0);
}
```

کاملاً مشخص است که این کد برای ما چه کاری انجام می‌دهد. متد ShowValueOfRow() را با آرگومان ۰ صدا می‌زند که در واقع ردیف یک را به ما نمایش می‌دهد. برای btnLast بر روی آن کلیک می‌کنیم و مقدار زیر را به آن اضافه می‌کنیم.

```
private void btnLast_Click(object sender, EventArgs e)
{
    int lastIndex = dataset.Tables["Members"].Rows.Count - 1;

    ShowValuesOfRow(lastIndex);
}
```

با استفاده از خصوصیت Count از Row تعداد ردیف‌هایی که در جدول Members قرار دارد را بدست می‌آوریم و یک واحد از آن کم می‌کنیم که کار ایندکس را برای ما انجام دهد (ایندکس از ۰ شروع می‌شود) و از این ایندکس (که ایندکس آخر است) استفاده می‌کنیم و آن را به متد ShowValuesOfRow() می‌دهیم تا ردیف آخر را برای ما در textboxها نمایش دهد. در کد زیر هم کد مربوط به btnPrev را نمایش می‌دهیم.

```
private void btnPrev_Click(object sender, EventArgs e)
{
    if (position > 0)
    {
        position--;
        ShowValuesOfRow(position);
    }
}
```

در کد بالا یک واحد از position کم می‌کنیم و نمایش می‌دهیم. در داخل شرط If هم اول چک می‌کنیم که از صفر کمتر نشده باشد که اگر کمتر شود در داخل متد ShowValuesOfRow() به ما Exception می‌دهد (Exception indexOutOfRang). در کد زیر هم btnNext را نمایش می‌دهیم.

```
private void btnNext_Click(object sender, EventArgs e)
{
    int lastIndex = dataset.Tables["Members"].Rows.Count - 1;

    if (position < lastIndex)
    {
        position++;
        ShowValuesOfRow(position);
    }
}
```

در این کد هم مانند قبل ایندکس آخر را بدست می‌آوریم و از آن برای اینکه مقدار position از آن بیشتر نشود استفاده می‌کنیم (این کار را در شرط if انجام می‌دهیم). اگر این چک نشود ممکن است ما به جایی از جدول دسترسی پیدا کنیم که وجود ندارد (بیشتر از تعداد ردیف‌ها جلو برویم) که در این صورت exception می‌دهیم (IndexOutOfRange Exception). حالا برنامه را اجرا می‌کنیم و می‌توانیم با استفاده از دکمه First به ردیف اول و با دکمه Prev به ردیف قبلی و با دکمه Next به ردیف بعدی و با دکمه Last به ردیف آخر دسترسی پیدا کنیم. ما می‌توانیم از همین کد برای وصل شدن به دیتابیس با Source Sql Data استفاده کنیم تنها کاری که باید بکنیم این است که نوع Provider را تغییر دهیم. و هیمچنین باید Connection String را تغییر دهیم که با Server Sql سازگار باشد.

فصل ششم

معماری سه لایه

معماری سه لایه چیست؟

در معماری سه لایه تمام برنامه به چندین بخش تقسیم می‌شود. این بخش‌ها می‌توانند فیزیکی یا منطقی باشند. هر بخش کار خاصی را انجام می‌دهد مثلاً نمایش اینترفیس کاربر یا دسترسی به داده‌ها. برنامه می‌تواند به هر تعداد لایه داشته باشد ولی به هر حال بیشتر برنامه‌ها سه لایه مجزا دارند که عبارت‌اند از:

- Presentation Layer
- Business Logic Layer
- Data Access Layer

همان طور که احتمالاً حدس زده‌اید، لایه Presentation چیزی نیست جزء بخشی از نرم‌افزار که با کاربر ارتباط برقرار می‌کند (اینترفیس یا ظاهر برنامه شما) نمایش داده‌ها به کاربر نهایی و اجازه به او برای ارتباط داشتن با داده‌ها، اصلی‌ترین وظیفه این لایه است.

در بیشتر موارد داده‌هایی که توسط کاربر وارد می‌شوند نیاز به اعتبارسنجی یا پردازش اضافی دارند. این مسوولیت لایه Business Logic است. در نهایت داده‌های برنامه شما نیاز به ذخیره و بازیابی از طریق یک منبع داده دارند (مثلاً سیستم مدیریت دیتابیس‌های رابطه ای یا RDBMS و یا XML ...) این وظیفه توسط لایه دسترسی به داده یا Data Access، انجام می‌شود. به طور خلاصه، فرآیند مورد نظر ما این گونه کار می‌کند:

- کاربر برای داده‌های برنامه درخواستی ارسال می‌کند.
- لایه Data Access داده‌های مورد نظر را بازیابی می‌کند و از طریق لایه Business Logic آنها را به لایه نمایش می‌فرستد.
  بعضی مواقع لایه دسترسی به داده‌ها، این داده‌ها را مستقیماً به لایه نمایش ارسال می‌کند.
- لایه نمایش اطلاعاتی که باید نمایش داده شوند را از طریق لایه Business Logic دریافت می‌کند.
- کاربر داده‌ها را تغییر می‌دهد و عمل مناسب در مورد آنها را اجرا می‌کند (مثل اضافه یا به روز کردن داده‌ها).
- لایه Business Logic صحت داده‌های وارد شده توسط کاربر را بررسی می‌کند (داده‌ها را اعتبار سنجی می‌کند).
- اگر داده‌ها معتبر باشند آنها را برای به روز رسانی در بانک اطلاعاتی به دست لایه دسترسی به داده می‌سپارد.

مزیت‌های برنامه‌های چند لایه

برنامه‌ها به چند بخش منطقی جدا از هم تقسیم می‌شوند و اتصال میان UI (رابط کاربری)، پردازش‌ها و بانک اطلاعاتی کم می‌شود.

- تغییر در بانک اطلاعاتی یا روال‌های دسترسی به داده‌ها تاثیری در لایه نمایش یا برنامه Client نخواهد گذاشت.
- برنامه کلاینت با عبارات SQL آمیخته نخواهد شد.
- نام جداول و ستون‌ها به طور موثری از برنامه کلاینت حذف می‌شوند.
- برنامه کلاینت نمی‌فهمد که داده‌ها از کجا آمده‌اند (چیزی که به آن location transparency گفته می‌شود).
- تغییر یا گسترش برنامه بسیار ساده تر خواهد شد، بدون نیاز به تغییر یا کامپایل مجدد برنامه کلاینت.

نکته منفی در معماری سه لایه این است که شما باید تعداد زیادی بخش و کلاس از هم جدا در نرم‌افزار بسازید. اما به هر حال مزایای این روش بیشتر و برتر از معایب آن است.

انتخاب‌های لایه Presentation

دو انتخاب اصلی برای ساخت یک لایه نمایش در دات‌نت وجود دارد. آنها فرم‌های ویندوزی یا فرم‌های وبی ASP.NET هستند. با استفاده از ویندوز فرم‌ها شما می‌توانید برنامه‌های دسکتاپ فرم محور (form base) معمول را بسازید. برنامه‌های ویندوز فرمی می‌توانند المآنهای رابط کاربری بسیار غنی به کاربر پیشنهاد کنند. آنها کم و بیش شبیه به فرم‌های ویژوال بیسیک هستند. جذاب‌ترین گزینه برای توسعه لایه نمایش استفاده از وب فرم‌های ASP.NET است. کنترل‌هایی مثل Datagridview و Calendar یک رابط کاربری قدرتمند را با مقدار کمی کد فراهم می‌کنند. انتخاب‌هایی که در بالا برای ساخت یک لایه نمایش بررسی کردیم، می‌توانند توسط زبان‌های مختلفی مثل سی‌شارپ یا ویژوال بیسیک دات‌نت پیاده سازی شوند.

انتخاب‌های لایه Business Logic

لایه Business logic از چندین بخش که کارهایی نظیر اعتبار سنجی کار، گردش کار یا کارهای مشابه را انجام می‌دهند تشکیل شده است. کامپوننت‌های دات‌نت این لایه را شکل می‌دهند.

انتخاب‌های لایه Data Access

این لایه با دستکاری داده‌ها مثل اضافه، حذف و به روز رسانی آنها سر و کار دارد. داده‌هایی که به آنها اشاره کردیم می‌توانند در RDBMS یا XML قرار داشته باشند. شما باید لایه دسترسی به داده را چنان طراحی کنید که دیگر لایه‌ها نیازی به دانستن وضعیت منبع داده‌ها نداشته باشند.

ADO.NET فناوری دسترسی به داده تحت دات‌نت است. اگر چه ADO.NET از طریق کلاس‌های DataReader اجازه دسترسی به داده‌های در هنگام اتصال را می‌دهد ولی بیشترین تمرکز روی دسترسی به داده‌ها در زمان متصل نبودن است. Dataset نقش کلیدی را در این مورد بازی می‌کند. در بعضی موارد شما می‌توانید ADO را هم برای دسترسی به داده‌ها استفاده کنید، ولی استفاده از آن باید دلیل معتبری داشته باشد. از ADO استفاده نکنید، فقط به خاطر اینکه RecordSetها را دوست دارید!

این جا هم کامپوننت‌های دات‌نت لایه را تشکیل می‌دهند. همچنین وب سرویس‌ها هم می‌توانند لایه دسترسی به داده را شکل دهند. این مخصوصاً زمانی درست است که دیتابیس شما فراهم کننده (provider) داده ندارد. در این گونه موارد شما می‌توانید مقداری کد برای اتصال به داده‌ها و پر کردن دیتاست‌ها و بازگرداندن نتایج درون Dataset به درخواست کننده داده بنویسید.

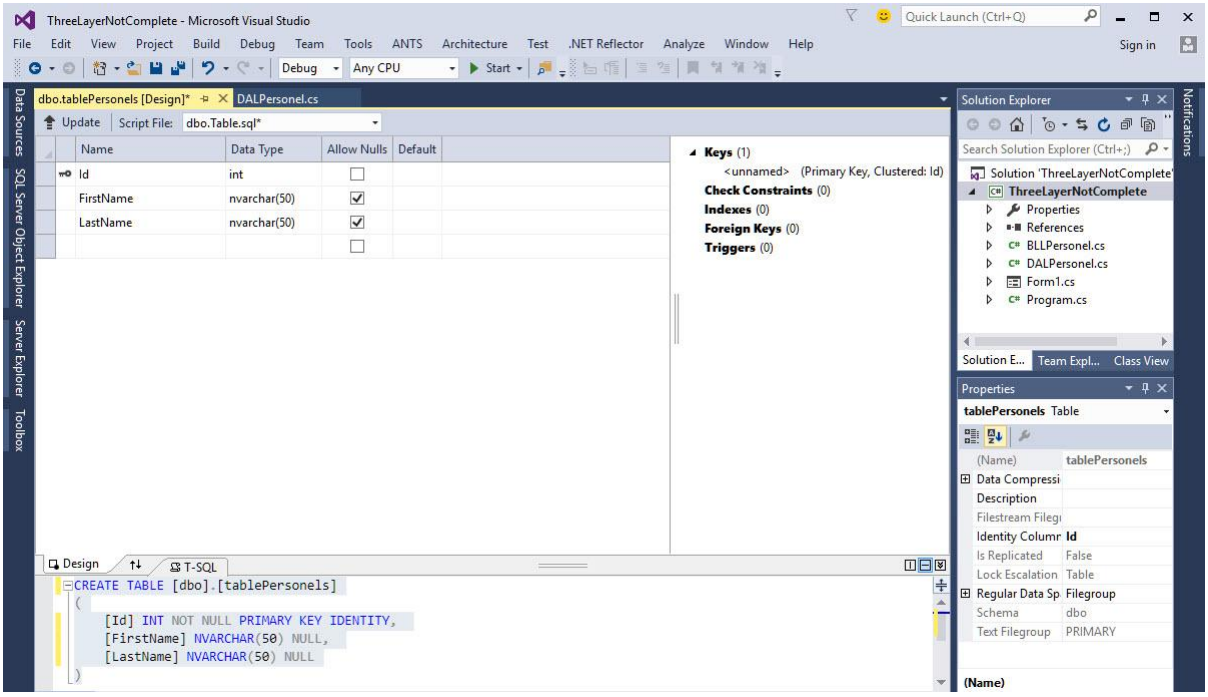
علاوه بر ADO.NET شما می‌توانید از امکانات سیستم مدیریت دیتابیس خود مثل توابع و یا روال‌های ذخیره شده (Procedures Stored) استفاده کنید. در این سری آموزشی قصد داریم در قالب یک پروژه عملی این معماری را به شما آموزش دهیم.

تشریح لایه‌ها در معماری سه لایه

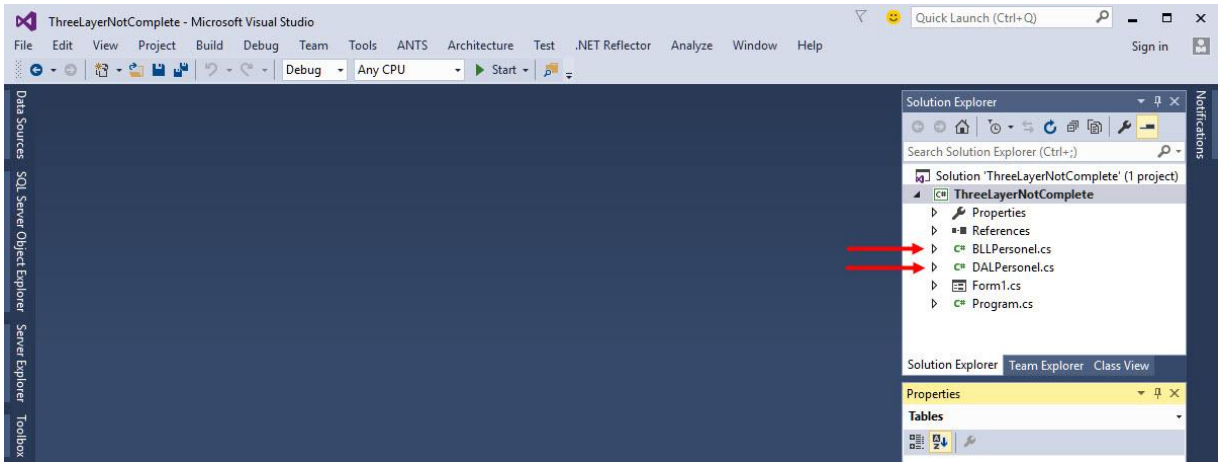
هر پروژه‌ای که در آن از تکنیک برنامه‌نویسی سه لایه استفاده می‌شود، به سه بخش کلی (لایه) تقسیم می‌شود:

- Presentation Layer: در این لایه ظاهر برنامه قرار می‌گیرد، از طریق این لایه کاربران می‌توانند با برنامه ارتباط برقرار کنند و داده‌های خود را مشاهده یا وارد نمایند.
- Business Logic Layer: منطق اصلی برنامه در این لایه قرار می‌گیرد. قوانین و کدهای معتبر سازی (Validation) و دسترسی (Authentication) در این لایه قرار می‌گیرند.
- Data Access Layer: همانطور که از اسمش می‌توان فهمید در این لایه کدهای دسترسی به داده نوشته می‌شود. در این لایه می‌توانید از تکنولوژی‌های مختلف بانک اطلاعاتی همانند Entity Framework، Linq to Sql، Ado.net استفاده کنید .

قصد دارم که با دو کلاس DALPersonel(Data Access Layer) و BLLPersonel(Business Logic Layer) و به کمک فرم نمایش داده‌ها برنامه‌نویسی سه لایه را جهت ورود داده‌ها برای جدول Personel پیاده سازی نماییم. ابتدا یک بانک اطلاعاتی (Databse.mdf) با یک جدول با نام tablePersonels و سه فیلد به نام‌های Id، FirstName، LastName ایجاد می‌کنیم:



سپس دو کلاس به نام‌های BLLPersonel و DALPersonel ایجاد می‌کنیم:



بر روی کلاس DALPersonel دابل کلیک کرده و کدها زیر را در آن قرار دهید (رشته اتصال بانک اطلاعاتی (خط ۵) را مناسب با سیستم و مشخصات بانک اطلاعاتی خود تغییر دهید):

```
public DataTable SelectData()
{
    string selectQuery = "SELECT * FROM tablePersonels;";
    SqlCommand Command1 = new SqlCommand();
    SqlConnection Connection1 = new SqlConnection(@"Data Source=(LocalDB)\v11.0;"
                + @"AttachDbFilename=|DataDirectory|\DataBase.mdf;"
                + @"Integrated Security=True;Connect Timeout=30");

    Command1.CommandText = selectQuery;
    Command1.Connection = Connection1;

    SqlDataAdapter DataAdapter1 = new SqlDataAdapter(Command1);

    try
    {
        DataTable result = new DataTable();
        DataAdapter1.Fill(result);
        return result;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return null;
    }
    finally
    {
        Command1.Connection.Close();
    }
}
```

سپس بر روی کلاس BLLPersonel دابل کلیک کرده و کد زیر را در آن قرار دهید:

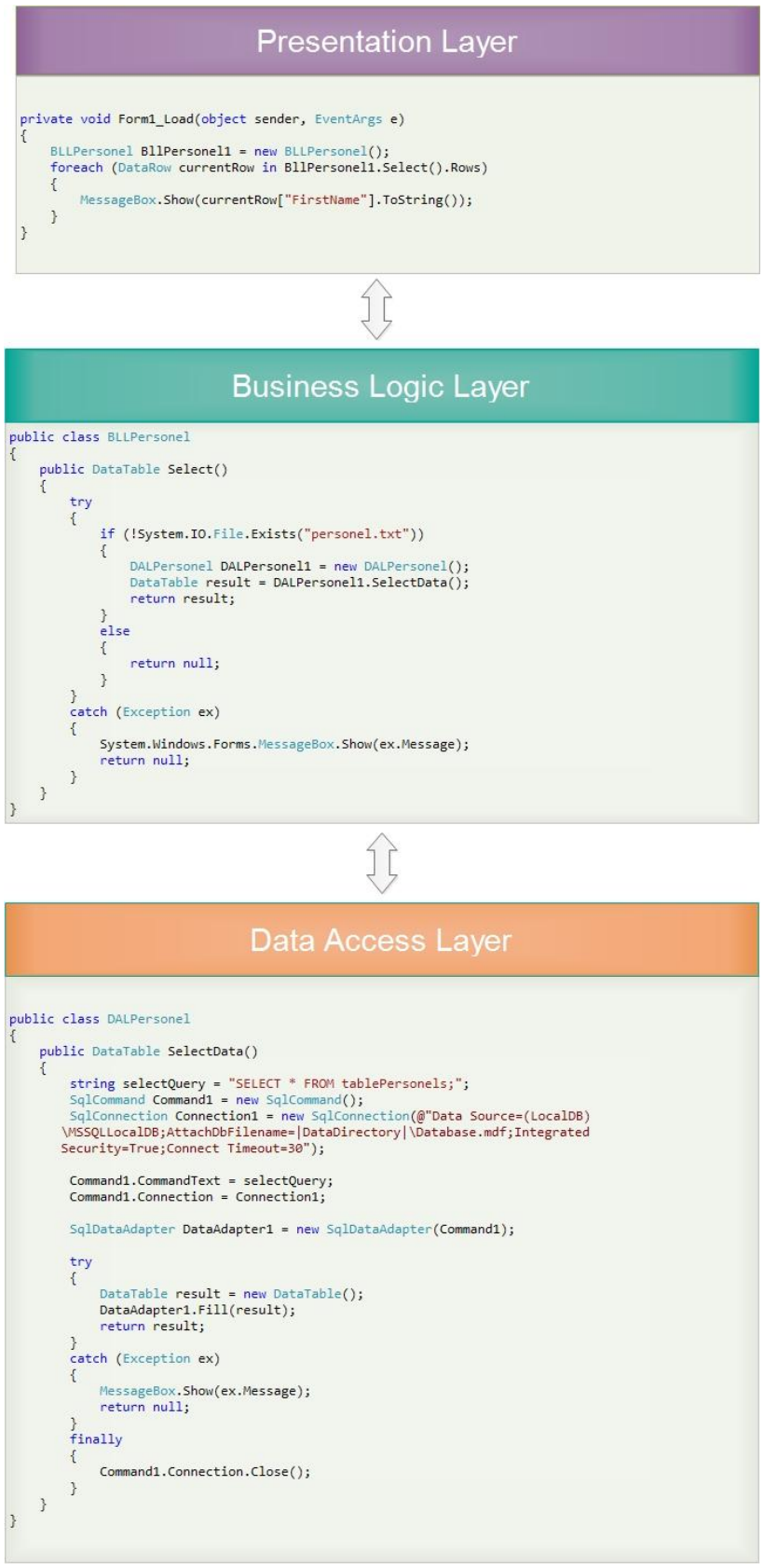
```
public DataTable Select()
{
    try
    {
        if (!System.IO.File.Exists("personel.txt"))
        {
            DALPersonel DALPersonel1 = new DALPersonel();
            DataTable result =  DALPersonel1.SelectData();
            return result;
```

```
        }
        else
        {
            return null;
        }
    }
    catch (Exception ex)
    {
        System.Windows.Forms.MessageBox.Show(ex.Message);
        return null;
    }
}
```

از آنجاییکه لایه BLL منطق برنامه را بر عهده دارد، در این لایه بررسی می‌کنیم که اگر شروطی حاکم بود عمل اتصال به لایه Data Access انجام شود. مثلاً در مثال بالا ابتدا بررسی می‌کنیم که آیا فایل personel.txt در مسیر پروژه قرار دارد یا خیر. اگر وجود نداشته باشد ارتباط با بانک برقرار می‌کنیم. این شرط فقط به عنوان مثال نوشته شده است و در برنامه‌های شما ممکن است شروط خاص خود را داشته باشید. بر روی فرم دابل کلیک کرده و کد زیر را در رویداد Load آن می‌نویسیم :

```
BLLPersonel BllPersonel1 = new BLLPersonel();
foreach (DataRow currentRow in BllPersonel1.Select().Rows)
{
    MessageBox.Show(currentRow["FirstName"].ToString());
}
```

در کد بالا از کلاس BLLPersonel شیء ای را می‌سازیم. این کد باعث ارتباط لایه Presentation با Business Logic می‌شود. در شکل زیر نمای کلی ارتباط لایه‌های مذکور نشان داده شده است:


Presentation Layer
private void Form1_Load(object sender, EventArgs e)
{
BLLPersonel BllPersonel1 = new BLLPersonel();
foreach (DataRow currentRow in BllPersonel1.Select().Rows)
{
MessageBox.Show(currentRow["FirstName"].ToString());
}
}
Business Logic Layer
public class BLLPersonel
{
public DataTable Select()
{
try
{
if (!System.IO.File.Exists("personel.txt"))
{
DALPersonel DALPersonel1 = new DALPersonel();
DataTable result = DALPersonel1.SelectData();
return result;
}
else
{
return null;
}
}
catch (Exception ex)
{
System.Windows.Forms.MessageBox.Show(ex.Message);
return null;
}
}
}
Data Access Layer
public class DALPersonel
{
public DataTable SelectData()
{
string selectQuery = "SELECT * FROM tablePersonels;";
SqlCommand Command1 = new SqlCommand();
SqlConnection Connection1 = new SqlConnection(@"Data Source=(LocalDB)
\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Database.mdf;Integrated
Security=True;Connect Timeout=30");
Command1.CommandText = selectQuery;
Command1.Connection = Connection1;
SqlDataAdapter DataAdapter1 = new SqlDataAdapter(Command1);
try
{
DataTable result = new DataTable();
DataAdapter1.Fill(result);
return result;
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
return null;
}
finally
{
Command1.Connection.Close();
}
}
}

مزایای کد نویسی به شکل بالا

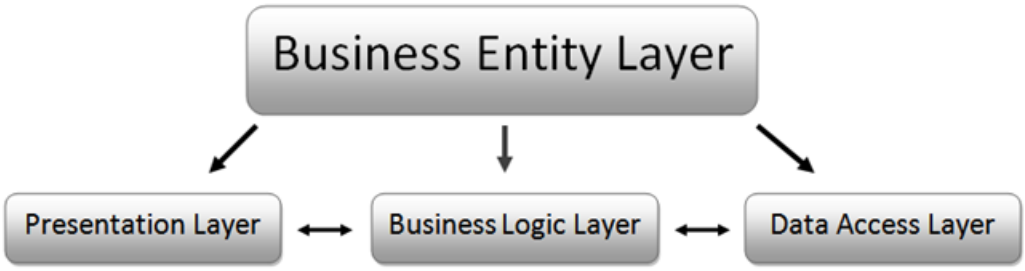
در برنامه بالا کدها به شیوه‌ی تمیز تر و بهتری نوشته شده‌اند. کدها در هم پیچیده نیستند و منطق برنامه به بخش‌های جدا از هم تقسیم می‌شود . مزیت مهم‌تری که کد نویسی بالا دارد این است که ممکن است در برنامه‌ای از تکنولوژی قدیمی مثل ADO.NET برای دسترسی به داده استفاده کرده باشیم و بعد از مدتی که تصمیم گرفتیم که از تکنولوژی جدیدتری مانند Entity Framework یا LINQ TO SQL برای دسترسی به داده استفاده کنیم. اگر منطق برنامه مانند شکل بالا به قسمت‌های مناسبی تقسیم بندی شده باشد و لایه‌ی Data Access وجود داشته باشد به راحتی می‌توانیم فقط این لایه را دستکاری کنیم.

معایب کد نویسی به شکل بالا

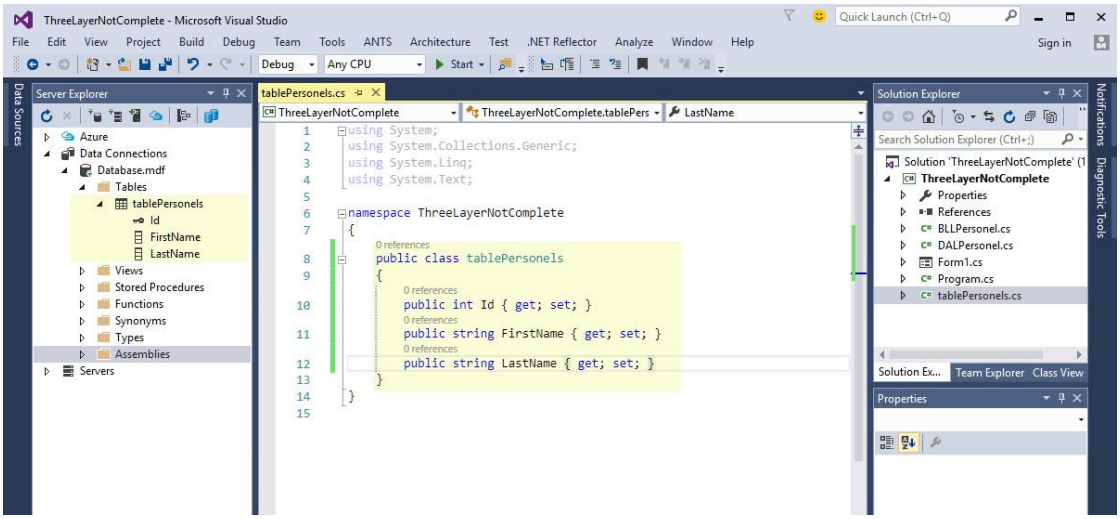
در شکل بالا اگر به کدهای لایه‌ی نمایش (Presentation) دقت کنید، این لایه هیچگونه شناختی از فیلدها و ستون‌های بانک اطلاعاتی ندارد و در زمان اجرا ساختار بانک را درک می‌کند. معمولاً وظیفه حل این مشکل بر عهده لایه‌ی Business Logic هست به این صورت اگر این لایه ساختار بانک را بشناسد، لایه‌ی Presentation نیز با توجه به ارتباطی که با این لایه دارد ساختار بانک را درک می‌کند. راه حلی که برای این مشکل وجود دارد این است که می‌توانیم یک کلاس اضافی به پروژه اضافه کنیم و ساختار بانک را در آن تعریف کنیم، سپس از این کلاس در هر سه لایه استفاده کنیم. به این ترتیب هر سه لایه از ساختار بانک اطلاعاتی باخبر می‌شوند. این لایه جدید Business Entity Layer نام دارد .

Business Entity Layer

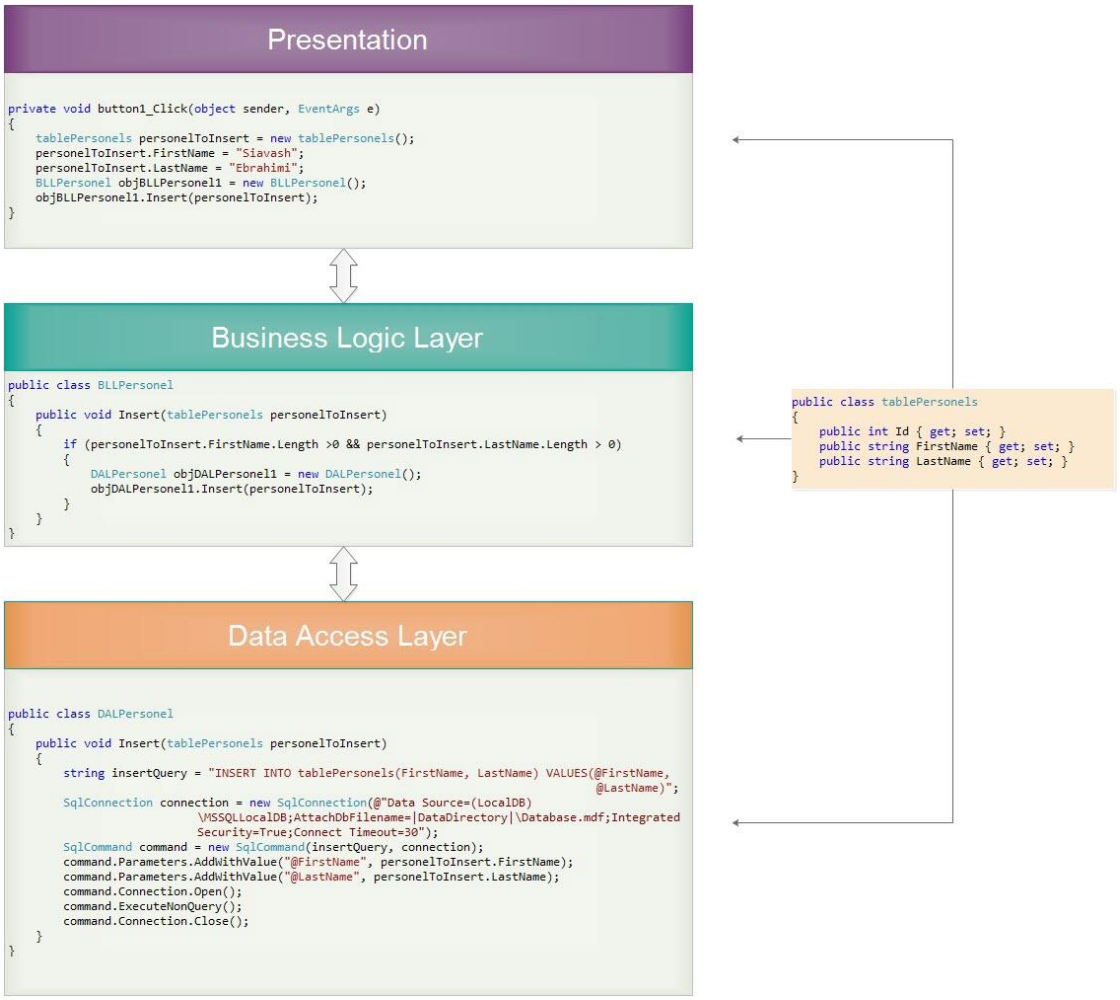
در این لایه به ازای هر جدول موجود در بانک، یک کلاس همنام با آن به همراه خصوصیاتی متناظر با فیلدهای جدول ایجاد می‌کنیم، پس از ایجاد لایه‌ی موجودیت (Entity)، این لایه به طور مشترک در بین تمامی لایه‌های استفاده می‌شود:



به عنوان مثال در بانک اطلاعاتی شکل زیر یک جدول به نام tablePersonels وجود دارد. این جدول دارای فیلدهایی به نام Id، FirstName، LastName می‌باشد. برای تبدیل این جدول به یک کلاس Entity Layer شما باید یک کلاس هم نام به این جدول و خصوصیاتی همنام با فیلدهای آن ایجاد کنید.



در شکل زیر عمل Insert را با استفاده از معماری جدید می‌نویسیم:



در مثال بالا کدها نسبت به مثال قبلی شکل بهتر و قابل فهمتری دارند ولی بازهم مشکلاتی دارد. به عنوان مثال جداسازی کدها می‌تواند به شکل بهتری انجام شود. مثلاً هر لایه را می‌توان در یک Class Library قرار داد و با استفاده از Referenceها بین لایه‌ها ارتباط برقرار کرد. برای

تقسیم بندی صحیح لایه‌ها بهتر است هر لایه را در یک پروژه از نوع Class Library یا همان Dllها قرار دهیم (به جزء لایه‌ی Presentation).

کتابخانه‌ی Data Access

در این لایه به ازای هر جدول موجود در بانک یک کلاس همنام با آن ایجاد می‌کنیم و در آن کدهای دسترسی به جدول مانند چهار عمل اصلی را می‌نویسیم. خروجی کامپایل این پروژه از نوع Dll می‌باشد و باید به پروژه Business Logic آن را اضافه کنیم. این کار را برای این انجام می‌دهیم که بتوانیم به کدهای این پروژه در لایه‌ی جدید (Business Logic) دسترسی داشته باشیم .

کتابخانه‌ی Business Logic

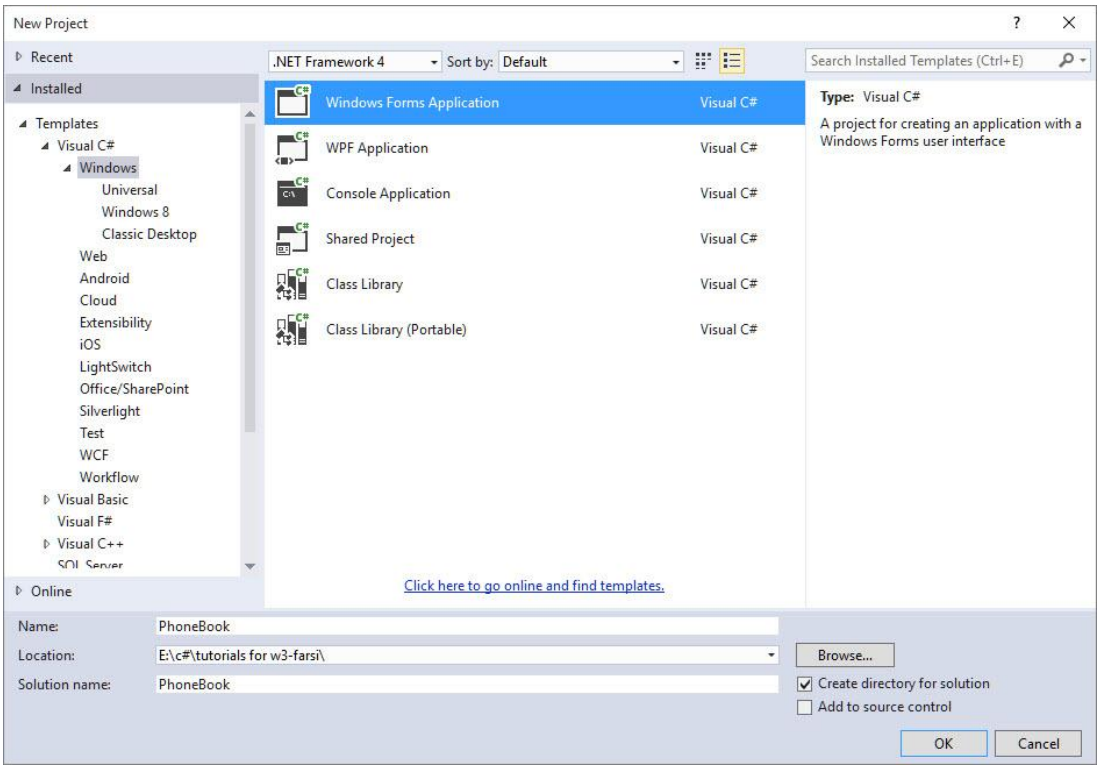
در این لایه به ازای هر کلاس موجود در کتابخانه‌ی Data Access Layer یک کلاس جهت مدیریت عملکرد متدها و توابع کلاس‌های کتابخانه Data Access ایجاد می‌کنیم. این لایه فایل DLL خروجی حاصل از کتابخانه Data Access Layer را استفاده می‌کند. خروجی کامپایل این پروژه یک فایل Dll است که در لایه Presentation Layer استفاده خواهد شد. مثلاً اگر در لایه‌ی Data Access یک کد ثبت نوشته باشیم باید در این پروژه یک متد همنام با آن ایجاد کنیم و شروطی را بررسی کنیم مثلاً اگر مقادیر تمامی فیلدها پر شده باشد، عمل ثبت را انجام دهد .

کتابخانه‌ی Business Entity

در این لایه به ازای هر جدول موجود در بانک یک کلاس همنام با آن، و به ازای هر ستون موجود یک خصوصیت همنام با آن ایجاد می‌کنیم. خروجی کامپایل این پروژه یک فایل Dll بوده که در هر سه لایه به صورت مشترک استفاده می‌شود .

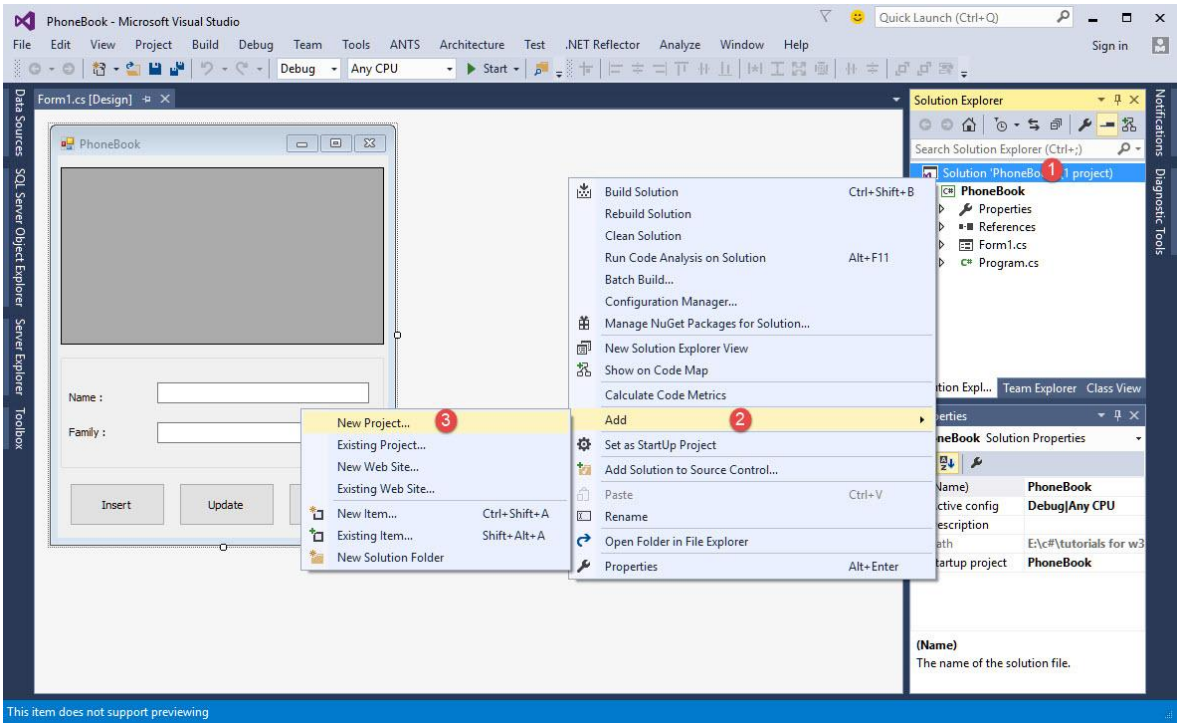
سیستم ثبت مشخصات فردی – با استفاده از معماری سه لایه

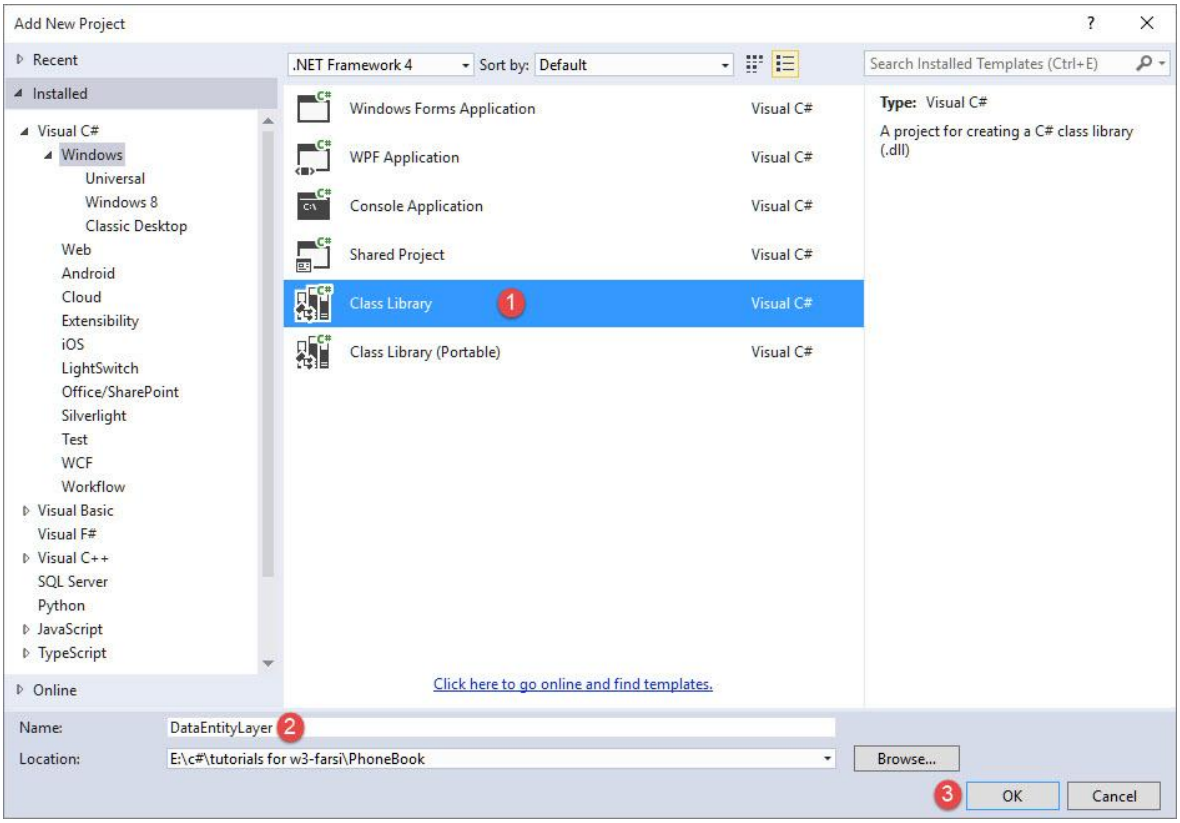
در این درس می‌خواهیم سیستم ثبت مشخصات فردی را استفاده از معماری سه لایه بازنویسی کنیم. برای اینکار یک پروژه از نوع Windows Form Application به صورت زیر ایجاد کنید:



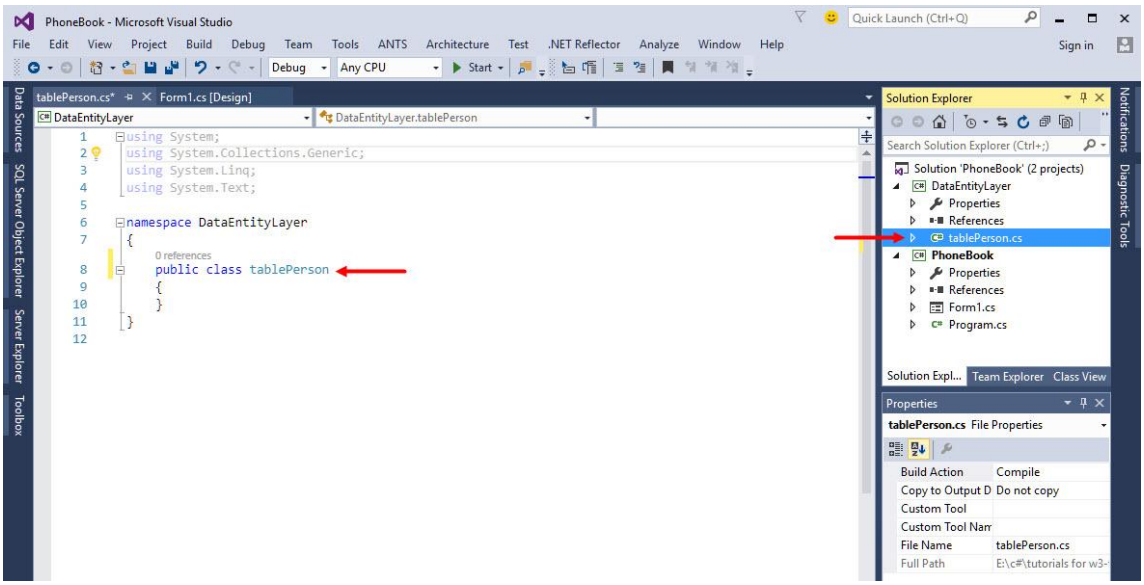
اگر دقت کرده باشید نسخه `.Net Framework` را به ۴ تنظیم کرده‌ایم، در این برنامه از قابلیت‌های این نسخه استفاده نشده است شما می‌توانید از هر نسخه‌ای که مایل باشید (۲ به بالا) استفاده کنید، ولی بهتر است از نسخه‌های جدید استفاده کنید، زیرا احتمال دارد که در پروژه به قابلیت‌هایی که در این نسخه‌ها ارائه شده است نیاز پیدا کنید. ظاهر برنامه‌ای که می‌خواهیم طراحی کنیم به شکل زیر است:

برروی `Solution` راست کلیک کرده و به شکل زیر یک پروژه `Class Library` به نام `DataEntityLayer` ایجاد کنید

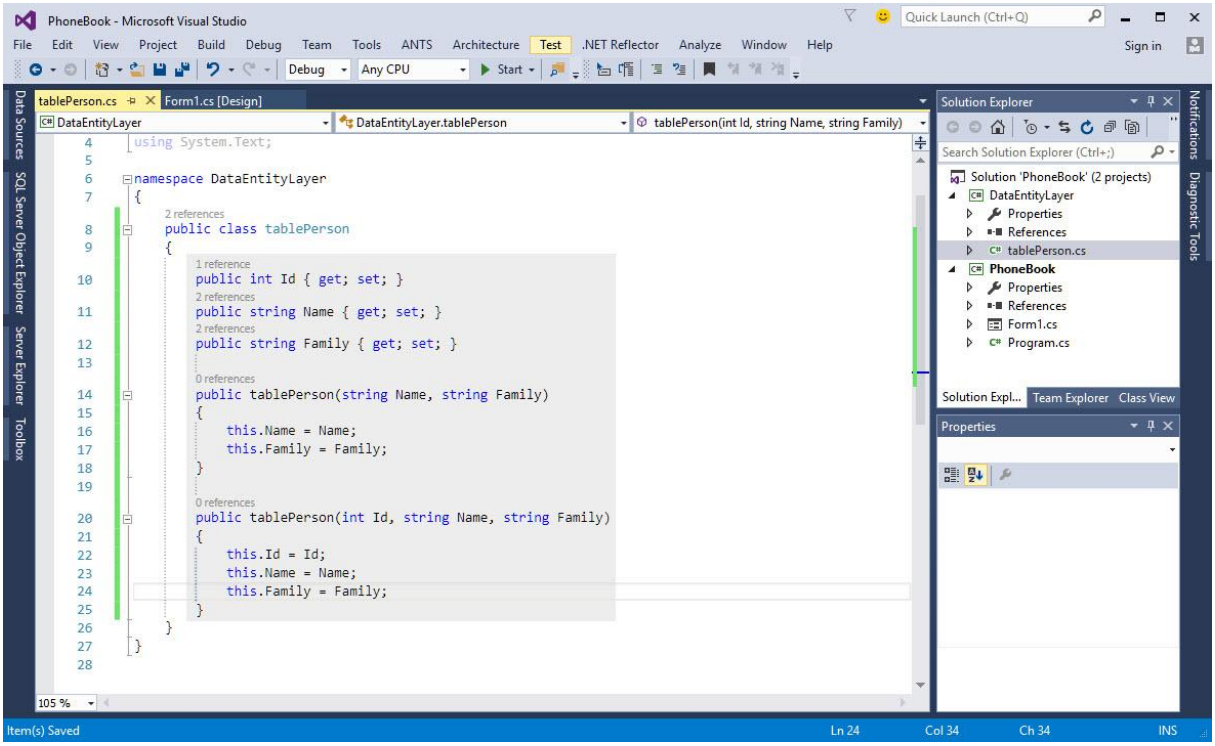




بعد از اضافه کردن پروژه یک کلاس به صورت خودکار به نام `Class1.cs` ایجاد می‌شود. نام آن را به `tablePerson` تغییر دهید. همچنین بر روی آن دابل کلیک کنید و نام کلاس داخل آن را نیز به `tablePerson` تغییر داده، تا به صورت زیر درآید

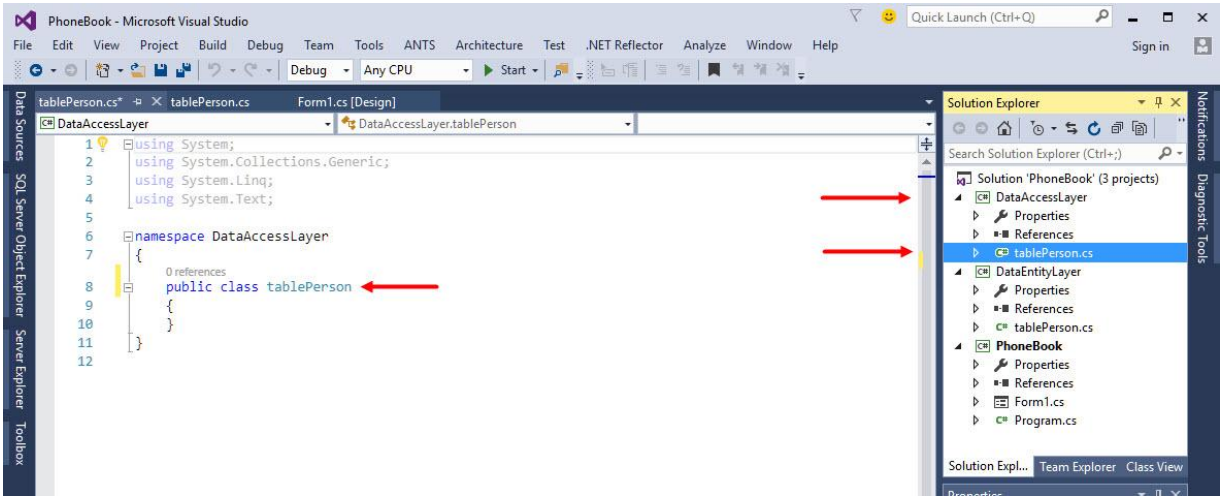


همانطور که در شکل زیر مشاهده می‌کنید به ازای هر فیلد موجود در بانک یک خصوصیت هم نام و هم نوع با آن ایجاد می‌کنیم

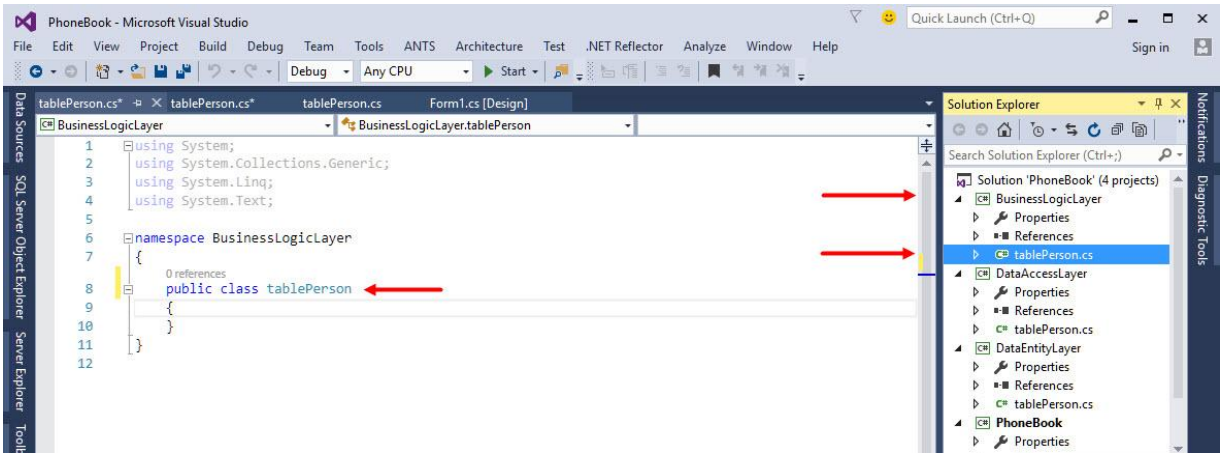


برای نمونه سازی سریع از این کلاس، دو سازنده برای آن تعریف کرده‌ایم. در مرحله بعد باید یک پروژه از نوع `Data Access Layer` ایجاد کنیم. همانند مراحل بالا و به شکل مشابه یک پروژه از نوع `Class Library` با نام `DataAccessLayer` ایجاد کنید

.



همانطور که در شکل بالا مشاهده می‌کنید همانند لایه Entity به ازای هر جدول یک کلاس همنام ایجاد می‌کنیم. در این کلاس باید کدهای CRUD (ثبت، ویرایش، حذف، انتخاب) را بنویسید. فعلاً با کد نویسی کاری نداریم و به ساخت لایه‌ی آخر یعنی Business Logic Layer می‌پردازیم. برای ساخت این لایه (Business Logic) همانند لایه‌های دیگر یک پروژه از نوع Class Library به Solution اضافه می‌کنیم و نام آن را به BusinessLogicLayer تغییر می‌دهیم. سپس نام کلاسی که به طور پیش‌فرض به این لایه اضافه می‌شود را tablePerson تغییر می‌دهیم .



کار ساخت لایه‌ها به اتمام رسیده است. حال باید ارتباط بین این ۳ لایه را برقرار کنیم. در درس بعد درباره ارتباط بین لایه‌ها توضیح می‌دهیم.

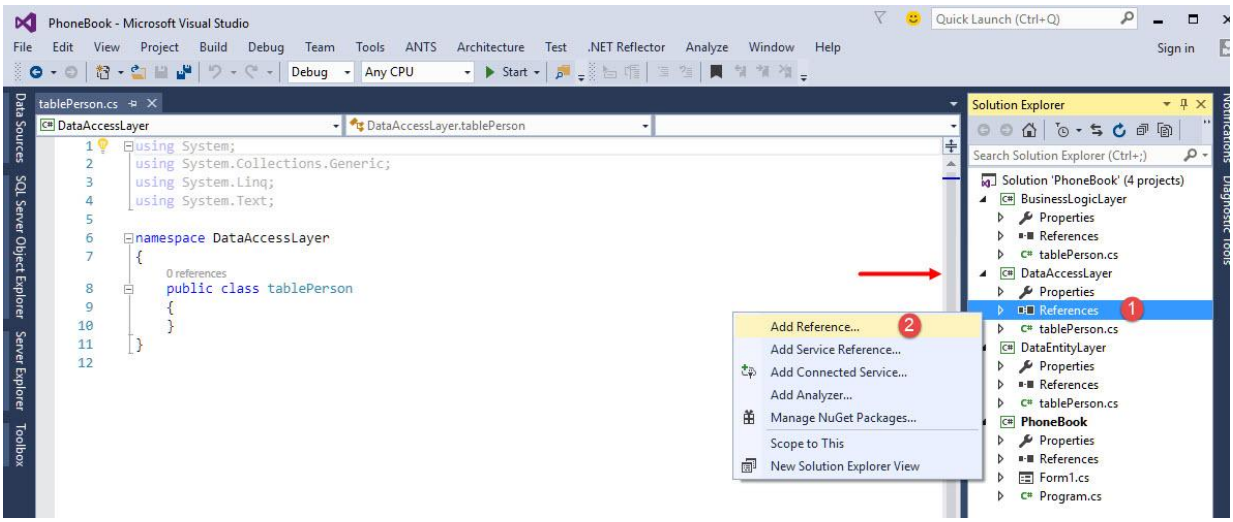
برقراری ارتباط بین لایه‌ها

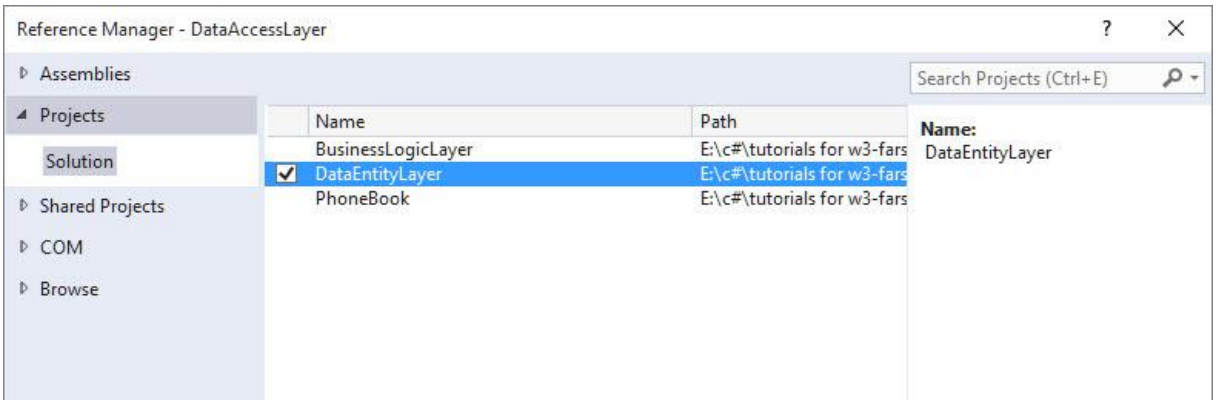
در این درس می‌خواهیم نحوه ارتباط بین لایه‌ها در معماری سه لایه را برای شما توضیح دهیم. مراحل برقراری ارتباط به شکل زیر می‌باشد:

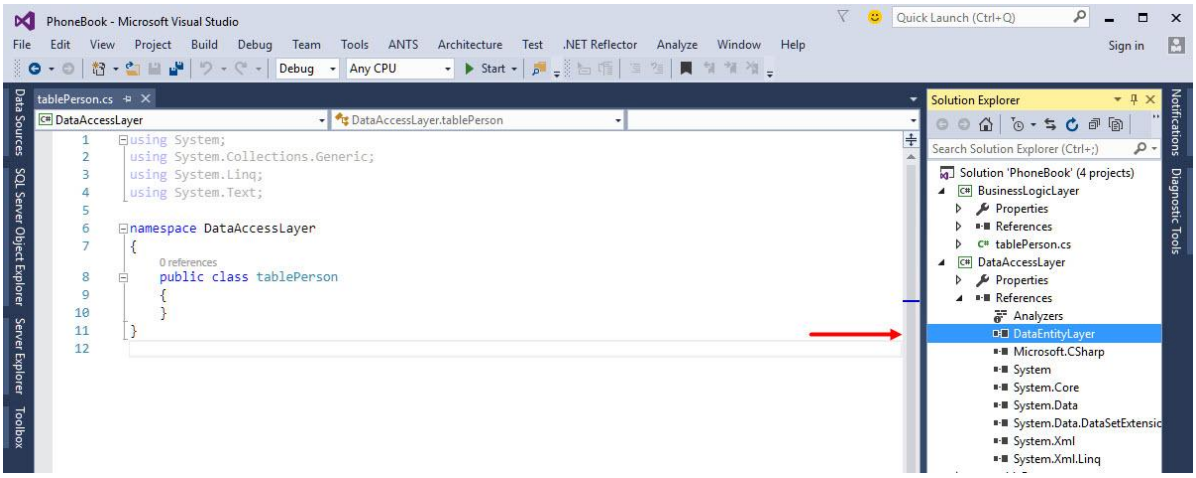
- Data Entity را به سه لایه‌ی دیگر اضافه می‌کنیم .
- لایه‌ی Data Access را به لایه‌ی Business Logic اضافه می‌کنیم.
- لایه‌ی Business Logic را به Presentation Layer وصل می‌کنیم .

- وصل کردن Data Entity Layer به سه لایه‌ی دیگر.

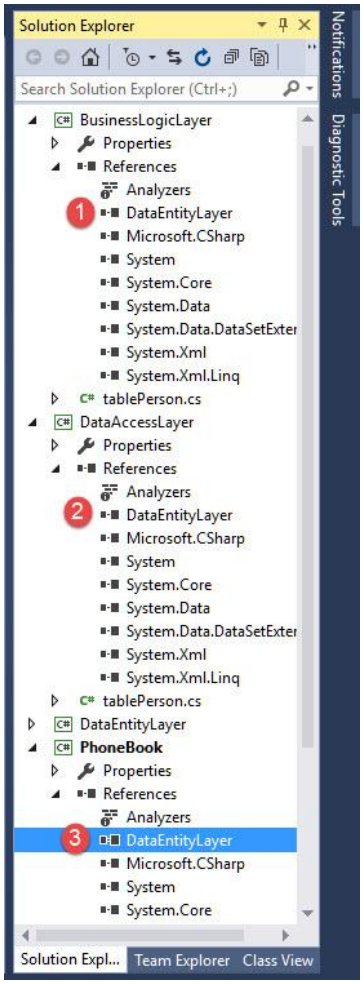
برای اینکار باید به شکل زیر عمل کنید:





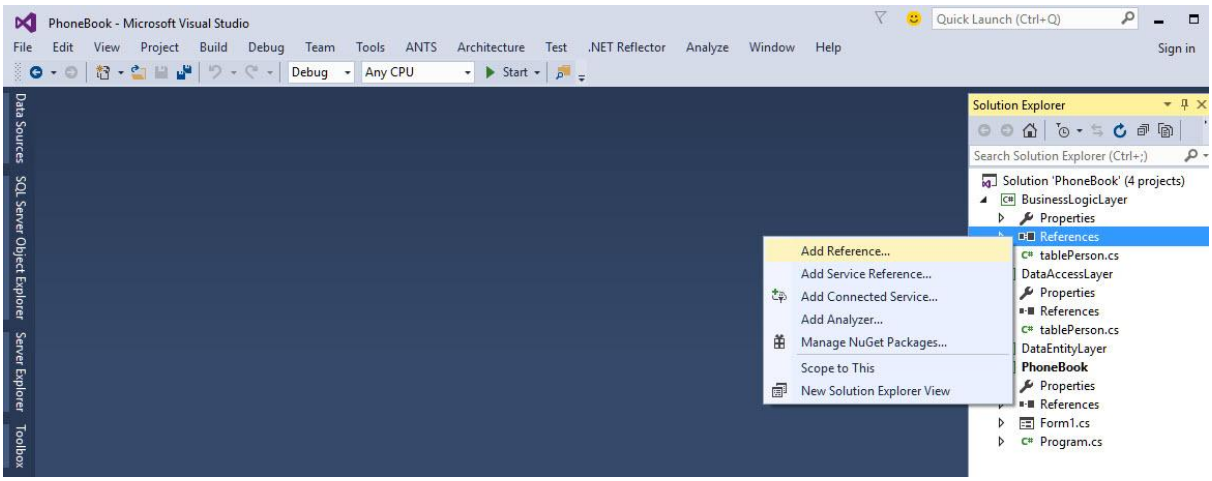


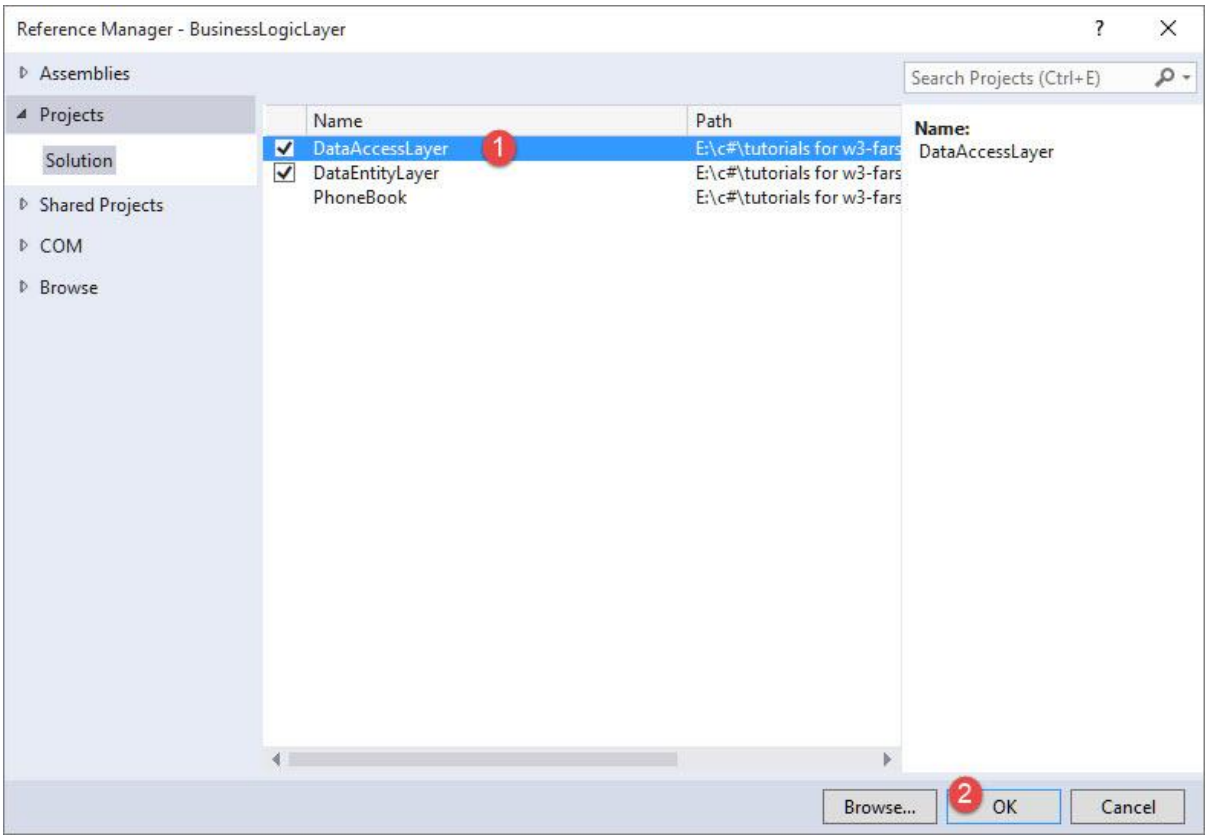
همانطور که در شکل بالا مشاهده می‌کنید لایه‌ی Data Entity را با موفقیت با لایه‌ی Data Access وصل کرده‌ایم. برای دو لایه‌ی دیگر (Business Logic و Presentation) هم عملیات مشابه را انجام می‌دهیم. در نهایت شکل نهایی Solution Explorer به صورت زیر در می‌آید:



اضافه کردن لایه‌ی Data Access به Business Logic

برای اضافه کردن لایه‌ی Data Access به Business Logic همانند مراحل بالا باید از گزینه‌ی Add Reference استفاده کنیم:

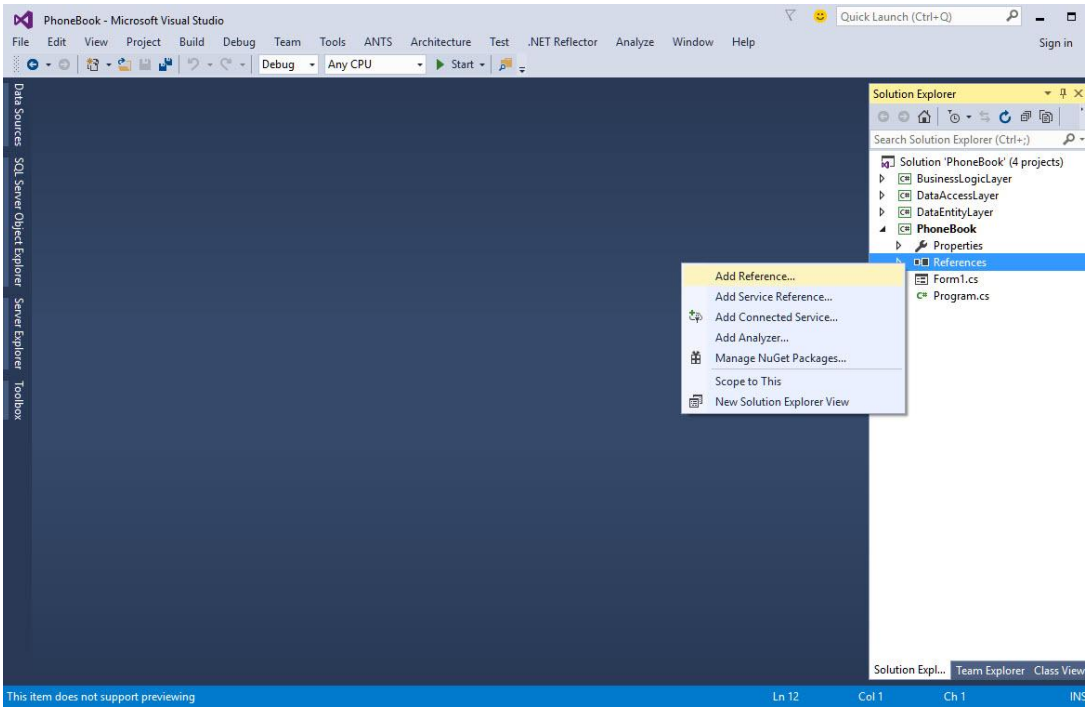


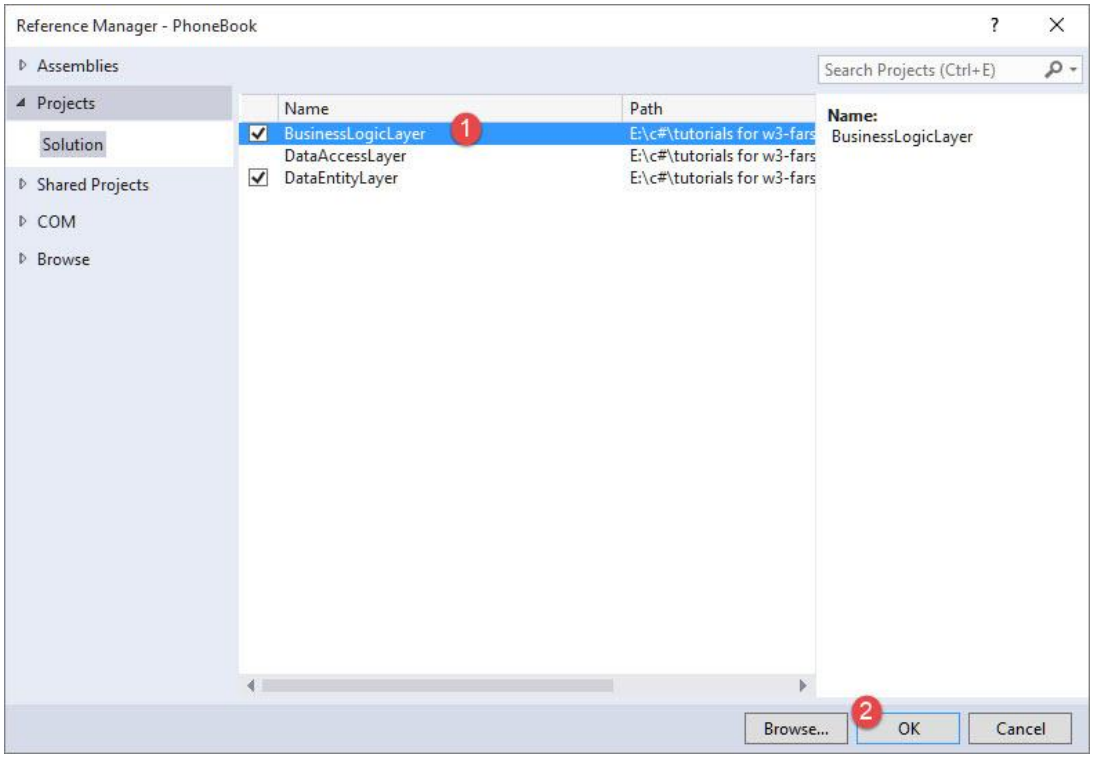


عمل اتصال لایه‌ی Data Access به Business Logic با موفقیت انجام شد.

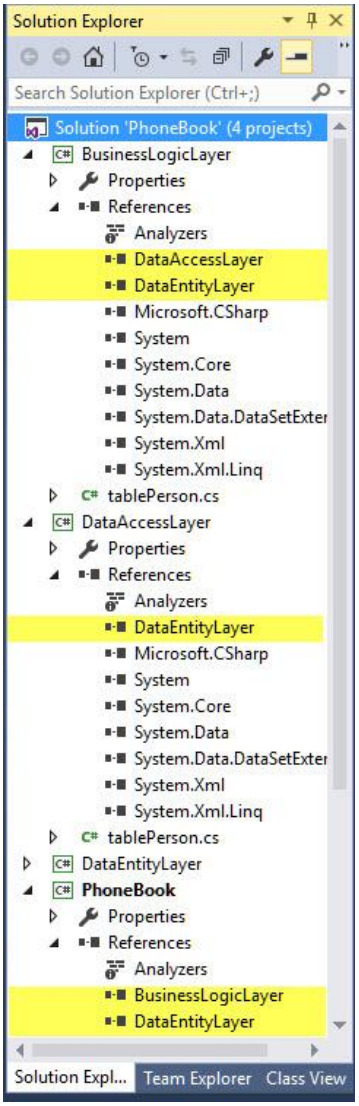
متصل کردن لایه‌ی Business Logic به Presentation Layer

به طریق مشابه مرحله ۲ می‌توانید این دو لایه را نیز به همدیگر وصل نمایید





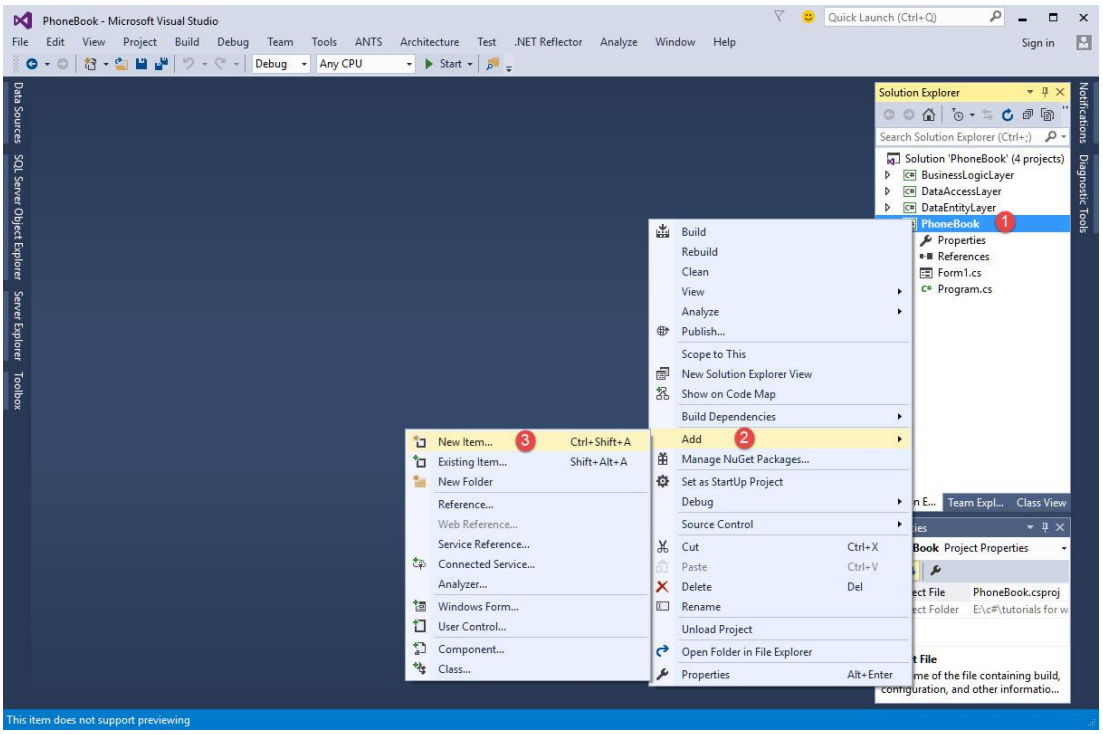
در نهایت نمای کلی پنجره‌ی `Solution Explorer` به شکل زیر در می‌آید:

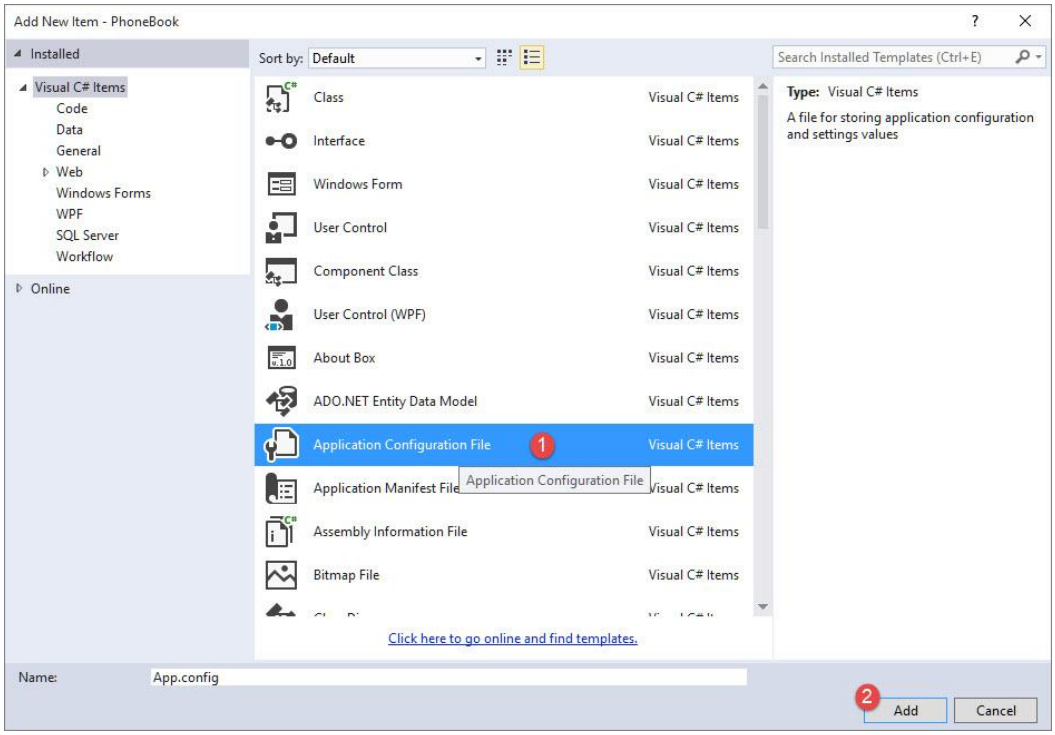


کار ساخت لایه‌ها و ارتباط آنها با پایان رسید. در درس بعد شما را با نحوه‌ی کد نویسی هر یک از این لایه‌ها آشنا می‌کنیم.

عملیات انتخاب، درج، حذف و ویرایش

در این درس شما را با نحوه‌ی کد نویسی در لایه‌ها آشنا می‌کنم. همانطور که در درس‌های قبلی خواندیم، بانک اطلاعاتی برنامه متشکل از یک جدول با سه فیلد می‌باشد. حال قصد داریم چهار عمل انتخاب، درج، حذف و ویرایش را برای این جدول پیاده سازی کنیم. ابتدا برای راحتی کار و ساده تر شدن نگهداری و تغییر برنامه باید رشته اتصال به بانک اطلاعاتی را در یک مکان تعریف کنیم تا اگر نیاز به تغییر آن داشتید فقط یک مکان را تغییر دهید. برای این کار باید رشته اتصال را در فایل `App.config` برنامه قرار دهید، به شکل زیر این کار را انجام می‌دهیم:



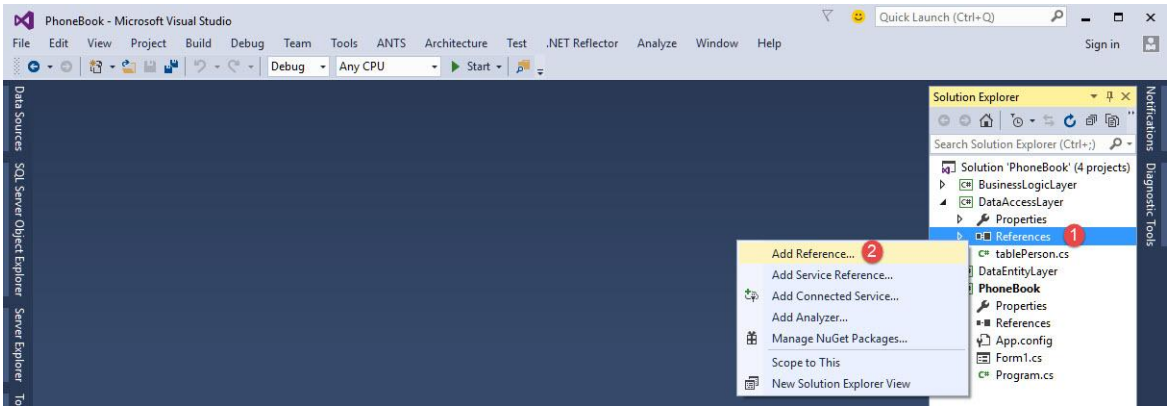


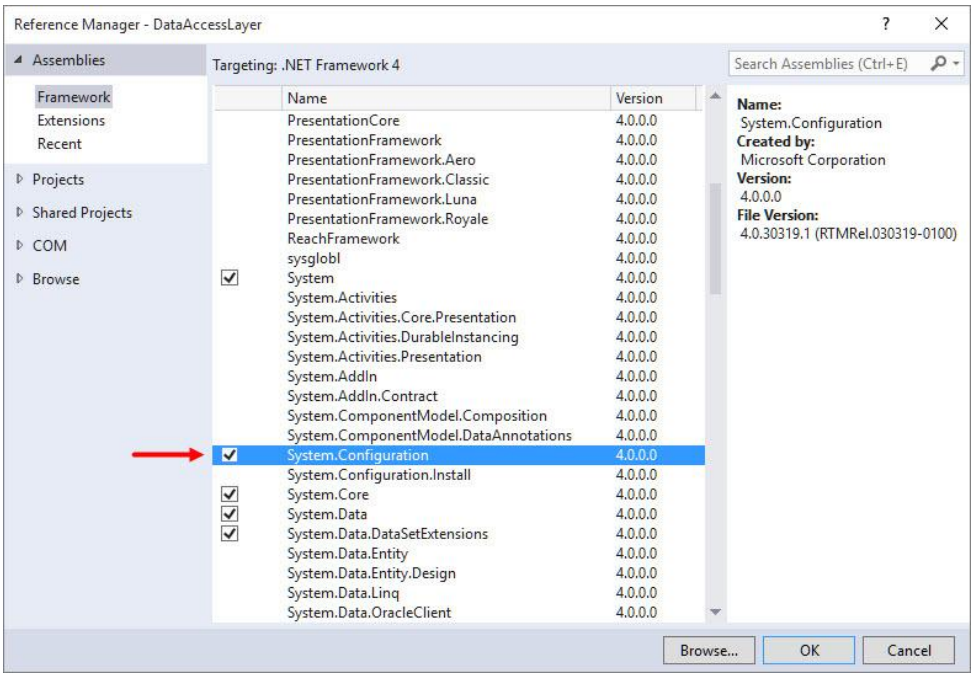
بر روی فایل ایجاد شده دوبار کلیک کرده و کدهای آن را به شکل زیر تغییر دهید :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="DatabaseConnectionString"
         connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
```

```
                    AttachDbFilename=|DataDirectory|\PhoneBookDatabase.mdf;
                    Integrated Security=True;Connect Timeout=30"/>
  </connectionStrings>
</configuration>
```

حال چونکه می‌خواهیم از این رشته اتصال در قسمت DataAccessLayer استفاده کنیم، باید اسمبلی System.Configuration را به این قسمت اضافه کنیم. برای این کار به مراحل زیر توجه کنید :





و در نهایت فضای نام System.Configuration را به کلاس tablePerson موجود در لایه DataAccessLayer اضافه کنید. برای نوشتن کدهای دسترسی به بانک و چهار عمل اصلی باید کدها را در لایه‌ی Data Access بنویسیم.

انتخاب اطلاعات

ابتدا بر روی فایل tablePerson.cs در لایه‌ی DataAccessLayer دوبار کلیک کنید Connection را به صورت زیر تعریف کنید:

```
using System;
using System.Data;
using System.Data.SqlClient;

using System.Configuration;

namespace DataAccessLayer
{
    public class tablePerson
    {
        private SqlConnection databaseConnection = new SqlConnection();

        public tablePerson()
        {
            this.databaseConnection.ConnectionString =
                 ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;
        }
    }
}
```

در کد بالا نام رشته اتصالی را که قبلاً در فایل App.Config تعریف کرده بودیم را در خاصیت ConnectionStrings کلاس ConfigurationManager و سپس خروجی آنرا در خاصیت ConnectionString فیلد databaseConnection قرار داده‌ایم. برای نوشتن عمل واکشی اطلاعات، بر روی فایل tablePerson.cs در لایه‌ی DataAccessLayer دوبار کلیک کرده و کد زیر را در کلاس tablePerson قرار دهید:

```
public DataTable Select()
{
    string selectQuery = "SELECT * FROM tablePerson;";
    SqlDataAdapter SqlDataAdapter1 = new SqlDataAdapter(selectQuery, this.databaseConnection);
    DataTable DataTable1 = new DataTable();
    SqlDataAdapter1.Fill(DataTable1);
    SqlDataAdapter1.SelectCommand.Connection.Close();
    return DataTable1;
}
```

کد بالا عمل واکشی اطلاعات را به صورت یک متد در لایه‌ی DataAccessLayer تعریف می‌کند. حال باید یک متد نظیر را در لایه‌ی BusinessLoginLayer برای آن تعریف کنیم. بر روی فایل tablePerson.cs لایه‌ی Business Logic دوبار کلیک کرده و کد زیر را بنویسید:

```
public DataTable Select()
{
    DataAccessLayer.tablePerson DALTablePerson1 = new DataAccessLayer.tablePerson();
    return DALTablePerson1.Select();
}
```

کد بالا یک شیء از کلاس tablePerson موجود در لایه‌ی Data Access ساخته و متد Select() آن را فراخوانی می‌کند. در مرحله آخر باید اطلاعات را در فرم نمایش دهیم. بر روی فرم دوبار کلیک کرده و کد زیر را در رویداد Load آن بنویسید:

```
BusinessLogicLayer.tablePerson BLLTablePersonels1 = new BusinessLogicLayer.tablePerson();
this.dataGridView1.DataSource = BLLTablePersonels1.Select();
```

برنامه را اجرا کنید و نتیجه را مشاهده کنید.

ثبت اطلاعات

نمای کلی عملیات ثبت به شکل زیر می‌باشد:

ابتدا یک متد در لایه DataAccessLayer برای عمل درج اطلاعات می‌نویسیم، بر روی فایل tablePerson در این لایه دوبار کلیک کرده و یک متد به شکل زیر به این کلاس اضافه کنید:

```
public void Insert(DataEntityLayer.tablePerson newPerson)
{
    string insertQuery = "Insert Into tablePerson(Name,Family) VALUES(@Name, @Family);";
    try
    {
        SqlCommand insertCommand = new SqlCommand(insertQuery, this.databaseConnection);
        insertCommand.Parameters.AddWithValue("@Name", newPerson.Name);
        insertCommand.Parameters.AddWithValue("@Family", newPerson.Family);
        this.databaseConnection.Open();
        int result = insertCommand.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally {
        this.databaseConnection.Close();

    }
}
```

همانطور که در خط ۸ مشاهده می‌کنید از آنجاییکه DataEntityLayer.tablePerson معادل یک رکورد از بانک اطلاعاتی می‌باشد، مقادیر پارامترها را از خصوصیات این شیء می‌گیریم و در بانک ثبت می‌کنیم. حال به مانند متد Select() یک متد معادل در لایه BusinessLogicLayer برای متد Insert() اضافه می‌کنیم. بر روی فایل tablePerson در لایه BusinessLogicLayer دوبار کلیک کرده و متدی به شکل زیر به این کلاس اضافه کنید:

```
public void Insert(DataEntityLayer.tablePerson newPerson)
{
    if (!string.IsNullOrEmpty(newPerson.Name) && !string.IsNullOrEmpty(newPerson.Family))
    {
        try
        {
            DataAccessLayer.tablePerson DALTablePerson1 = new DataAccessLayer.tablePerson();
            DALTablePerson1.Insert(newPerson);
        }
        catch (Exception)
        {
            throw new Exception("Error in inserting into the database, Please try again");
        }
    }
    else
    {
        throw new Exception("Please Enter Name And Family ...");
    }
}
```

در متد بالا اعتبارسنجی مقادیری که در خصوصیات Name و Family وجود دارد را با استفاده از متد IsNullOrEmpty() کلاس String انجام می‌دهیم. اگر کاربر مقادیر را وارد کرده باشد عمل اصلی را بر روی بانک اطلاعاتی انجام می‌دهیم. حال بر روی دکمه‌ی درج در فرم برنامه دوبار کلیک کرده و کد زیر را می‌نویسیم:

```
DataEntityLayer.tablePerson tablePerson1 =
    new DataEntityLayer.tablePerson(txtName.Text, txtFamily.Text);

BusinessLogicLayer.tablePerson BLLTablePerson1 = new BusinessLogicLayer.tablePerson();

try
{
    BLLTablePerson1.Insert(tablePerson1);
    MessageBox.Show("New person has been successfully inserted into database");
    this.dataGridView1.DataSource = BLLTablePerson1.Select();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

در خط ۲ یک شیء از کلاس tablePerson که در لایه‌ی DataEntityLayer قرار دارد، ایجاد می‌کنیم. داده‌هایی که از فرم می‌گیریم (نام و نام خانوادگی) نظیر به نظیر در خصوصیات این شیء قرار می‌دهیم. سپس این شیء را برای عملیات اعتبار سنجی به لایه‌ی Business Logic تحویل می‌دهیم.

حذف اطلاعات

برای نوشتن متد حذف ابتدا بر روی فایل tablePerson در لایه DataAccess دوبار کلیک کرده و سپس متد زیر را به این کلاس اضافه کنید :

```
public void Delete(int index)
{
    string deleteQuery = "DELETE FROM tablePerson Where Id = @Id;";
    try
    {
        SqlCommand SqlCommand1 = new SqlCommand(deleteQuery, this.databaseConnection);
        SqlCommand1.Parameters.AddWithValue("@Id", index);
        SqlCommand1.Connection.Open();

        int result = SqlCommand1.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
        this.databaseConnection.Close();
    }
}
```

در مرحله دوم متد زیر را به کلاس tablePerson در لایه BussinessLogicLayer اضافه کنید :

```
public void Delete(int index)
{
    if (index > 0)
    {
        try
        {
            DataAccessLayer.tablePerson DALTablePerson1 = new DataAccessLayer.tablePerson();
            DALTablePerson1.Delete(index);
        }
        catch (Exception)
        {
            throw new Exception("Error in deleting record...Please try again");
        }
    }
    else
    {
        throw new Exception("Please enter person id");
    }
}
```

در نهایت بر روی دکمه‌ی حذف بر روی فرم دوبار کلیک کرده و کد زیر را در رویداد کلیک آن بنویسید :

```
if (this.dataGridView1.Rows.Count > 0)
{
    if (MessageBox.Show("Are you sure , you want to delete this record?",
                        "Warning",
                        MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        int personelRowIndex =
                Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[0].Value.ToString());

        BusinessLogicLayer.tablePerson BLLTablePerson1 = new BusinessLogicLayer.tablePerson();
        try
```

```
        {
            BLLTablePerson1.Delete(personelRowIndex);
            this.dataGridView1.DataSource = BLLTablePerson1.Select();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

در کد بالا ابتدا بررسی می‌کنیم که آیا رکوردی برای حذف وجود دارد یا نه. در صورتی درستی این مطلب شماره آن رکورد را گرفته و برای حذف به لایه BusinessLogicLayer ارسال می‌کنیم .

ویرایش اطلاعات

ابتدا متد زیر را به کلاس tablePerson در لایه DataAccessLayer اضافه می‌کنیم :

```
public void Update(DataEntityLayer.tablePerson personToUpdate)
{
    string updateQuery = "UPDATE tablePerson SET Name=@Name,Family=@Family WHERE Id=@Id;";
    try
    {
        SqlCommand updateCommand = new SqlCommand(updateQuery, this.databaseConnection);
        updateCommand.Parameters.AddWithValue("@Name", personToUpdate.Name);
        updateCommand.Parameters.AddWithValue("@Family", personToUpdate.Family);
        updateCommand.Parameters.AddWithValue("@Id", personToUpdate.Id);
        updateCommand.Connection.Open();
        updateCommand.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        throw ex;
    }
    finally
    {
        this.databaseConnection.Close();
    }
}
```

سپس متد معادل را برای اعتبارسنجی مقادیر به لایه منطقی (BusinessLogicLayer) اضافه می‌کنیم :

```
public void Update(DataEntityLayer.tablePerson personToUpdate)
{
    if (personToUpdate.Id > 0 &&
        !string.IsNullOrEmpty(personToUpdate.Name) &&
        !string.IsNullOrEmpty(personToUpdate.Family))
    {
        try
        {
            DataAccessLayer.tablePerson DALTablePerson1 = new DataAccessLayer.tablePerson();
            DALTablePerson1.Update(personToUpdate);
        }
        catch (Exception)
        {
            throw new Exception("Error in updating information , Please try again");
        }
    }
    else
    {
```

```
        throw new Exception("Please enter all information");
    }
}
```

و در نهایت بر رور دکمه‌ی ویرایش بر روی فرم دوبار کلیک کرده و کد زیر را در رویداد کلیک آن بنویسید :

```
if (this.dataGridView1.Rows.Count > 0)
{
    if (this.btnUpdate.Text == "Update")
    {
        string Name   = this.dataGridView1.CurrentRow.Cells[1].Value.ToString();
        string Family = this.dataGridView1.CurrentRow.Cells[2].Value.ToString();

        this.txtName.Text = Name;
        this.txtFamily.Text = Family;
        this.btnUpdate.Text = "Submit Update";
        this.btnInsert.Enabled = false;
        this.btnDelete.Enabled = false;
    }
    else
    {
        try
        {
            int personIndex = Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[0].Value.ToString());

            DataEntityLayer.tablePerson personToUpdate =
               new DataEntityLayer.tablePerson(personIndex, txtName.Text.Trim(),txtFamily.Text.Trim());

            BusinessLogicLayer.tablePerson BLLTablePerson1 =
               new BusinessLogicLayer.tablePerson();

            BLLTablePerson1.Update(personToUpdate);
            this.dataGridView1.DataSource = BLLTablePerson1.Select();
            MessageBox.Show("Record updated successfully");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        finally
        {
            this.btnUpdate.Text = "Update";
            this.btnInsert.Enabled = true;
            this.btnDelete.Enabled = true;
            this.txtName.Text = string.Empty;
            this.txtFamily.Text = string.Empty;
        }
    }
}
```

این دکمه باید یک وضعیت ۲ حالتی داشته باشد. اگر متد روی این دکمه "Update" باشد باید داده‌ها را از Datagridview به جعبه‌های متن انتقال دهیم و متن روی آن را به "Submit Update" تغییر دهیم. اگر متن روی آن "Submit Update" باشد باید عملیات ویرایش را انجام دهیم. روند عمل ویرایش نیز مانند ثبت و حذف می‌باشد. یعنی یک شیء از کلاس tablePerson لایه‌ی DataEntityLayer می‌سازیم و بعد از اعتبار سنجی آن به وسیله‌ی لایه‌ی Business Logic آن را برای عملیات نهایی به لایه‌ی Data Access تحویل می‌دهیم.