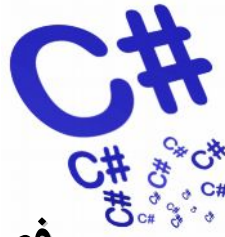




آموزش کاربردی زبان برنامه نویسی C#.NET
تهیه و تنظیم کنندگان : رضا بهرامی راد – مرتضی سلیمان پور
Bahramirad_reza@yahoo.com



فصل اول

مروری بر زبان برنامه نویسی C#.NET

مقدمه

نرم افزار Visual Studio در حدود ده سال پیش از طرف شرکت Microsoft ارائه شد و دنیای متفاوتی را پیش روی برنامه نویسان قرار داد. در ابتدا مقاومت هایی در مقابل این محیط و قابلیت های آن دیده شد ولی الان به جرأت می توان گفت که بسیاری از پروژه های کلان در این محیط انجام می شود.

زبان های C#.NET, VB.NET, J#.NET, C++.NET از زبان های قدرتمند این محیط هستند. محبوبیت این محیط از زمانی که نسخه ۲۰۰۵ SQL Server ارائه شد بالا رفت، زیرا با Integrated شدن دو محیط Microsoft SQL Server و Microsoft Visual Studio توانایی این دو محیط در بستر NET. فوق العاده بالا رفت.

در گذشته زبان هایی مثل VB یا Visual Fox Pro کم و بیش مورد استقبال کاربران قرار گرفته بود ولی چون این زبان ها برای تمام کار ها و پروژه ها کامل نبودند و هر کدام برای اهداف خاصی طراحی شده بودند برنامه نویسان دچار مشکلات اساسی می شدند. اینجا بود که محیط Visual Studio آمد و در بستر NET. انقلابی را در صنعت زبان های برنامه نویسی پدید آورد.

در محیط Visual Studio همه چیز پیش بینی شده است احتمالاً شما هم زمانی که شروع به برنامه نویسی پیرامون یک حوزه خاص می کنید به مسائلی برخورد می کنید که نیاز است تا قبل رسیدن به این مرحله بایستی پیش بینی های لازم را کرده باشیم تا دچار مشکلات فنی نشویم. ولی با کمی تسلط در محیط Visual Studio، به راحتی متوجه می شوید که حتی در زمینه های خیلی استثنایی هم پیش بینی های لازم صورت گرفته است. آن موقع است که متوجه می شوید این محیط چه قابلیت هایی دارد و چه اندازه به برنامه نویسان کمک می کند. محیط Visual Studio دارای محیط های متعددی برای برنامه نویسی است از محیط ویندوز گرفته تا محیط اینترنت، شبکه، موبایل و رایانه های شخصی. تمرکز اصلی این نرم افزار از اولین نسخه های آن تا کنون بر روی خصوصیت IDE بودن آن است که به برنامه نویس اجازه می دهد تا برنامه های کاربردی مستقلی را در محیط های ذکر شده بنویسد.

تا به حال نسخه های مختلفی از نرم افزار Microsoft Visual Studio ارائه شده است، در هر نسخه ای که ارائه شده از معایب نسخه قبلی کم شده و بر قابلیت های نسخه جدید اضافه شده است. ولی از لحاظ ساختار دستوری در هر یک از زبان های برنامه نویسی که در این محیط وجود دارند در نسخه های مختلف یکسان است و تفاوتی نمی کند. بطور مثال ساختار دستورات در زبان C# در نسخه ۲۰۰۸ هیچی تفاوتی با نسخه ۲۰۰۵ یا نسخه ۲۰۱۰ ندارد. ما در این کتاب اصول کاری و پایه کاری خود را بر روی نسخه ۲۰۰۸ این محصول قرار داده ایم، این محصول یکی از پرتیرترین نسخه ها می باشد و با وجود ارائه نسخه ۲۰۱۰ هنوز هم این نسخه از جایگاه بالاتری برخوردار است.

ویژگی های محیط Visual Studio

نرم افزار Visual Studio دارای قابلیت های فراوانی است، در مطالب زیر به معرفی برخی از این قابلیت ها می پردازیم :

Visual Studio تمام مراحل کامپایل یک سورس کد به یک برنامه قابل اجرا را به صورت خودکار انجام می دهد و به برنامه نویس اجازه می دهد تا تنظیمات دلخواه خود را بر روی پروژه انجام دهد.

ویرایشگر کد Visual Studio به صورت هوشمند طراحی شده است و می تواند در زمان نوشتن کد ها، خطا ها را تشخیص داده و در رفع آنها به برنامه نویس کمک کند.

Visual Studio شامل محیط هایی برای طراحی برنامه های ویندوز و برنامه های مبتنی بر وب، موبایل و ... است که به کمک آنها می توانید به سادگی محیط پروژه های خودتان را طراحی کنید.

برای اینکه در .NET یک برنامه تحت سیستم عامل ویندوز ایجاد کنید باید مقدار زیادی کد را که در اغلب برنامه ها بصورت تکراری هستند بنویسید. این مورد در ایجاد برنامه هایی از نوع های دیگر مانند برنامه های تحت وب نیز وجود دارد. Visual Studio با نوشتن خودکار کد ها در سرعت بخشیدن به طراحی برنامه ها کمک قابل توجهی می کند.

Visual Studio دارای ابزار های قدرتمندی برای کنترل قسمت های مختلف یک پروژه از قبیل سورس کد های C# و یا فایل های مورد نیاز برنامه از قبیل فایل های صوتی و تصویری است.

علاوه بر سادگی طراحی برنامه ها در NET. توزیع آنها نیز بسیار ساده تر است و به راحتی می توان آن را بر کامپیوتر های مقصد اجرا کرد.

Visual Studio دارای ابزار های فوق العاده قوی و هوشمندی جهت خطایابی دارد. برای مثال با استفاده از این ابزار ها می توان برنامه ها را خط به خط اجرا کرد و در اجرای هر خط موقعیت برنامه را بررسی کرد.

اصولاً هر آنچه را که یک برنامه نویس فکر می کند در محیط Visual Studio برایش فراهم گردیده است. در سال ۲۰۰۰ در یکی از سایت های خبری، خبری را خواندیم که قرار است نرم افزار Visual Studio آنقدر توسعه یابد که اگر یک فرد برنامه نویس صد سال و به صورت شبانه روز با این نرم افزار کار کند، نمی تواند به صورت کامل مباحث آن را تمام کند. آن زمان این خبر برای ما کمی تعجب برانگیز بود ولی حالا بعد از گذشت ۱۲ سال از آن زمان، الان متوجه می شویم که غول نرم افزاری دنیا یعنی شرکت Microsoft چه نرم افزار قدرتمند و حرفه ای را تولید کرده است.

آشنایی با Microsoft .NET Framework

قبل از تعریف و بررسی مفهوم Microsoft .NET Framework بهتر است که با مفهوم و معنای Platform و Framework آشنا شویم.

در یک تعریف عام، Platform عبارت است از مجموعه ی مؤلفه های پایه ای سخت افزاری و نرم افزاری که در کنار هم قرارگرفتن، زیر بنای تهیه یک نرم افزار را فراهم می کنند. بیش از نود درصد، برنامه های بازار در زمینه نگهداری و مدیریت داده های مالی یا سازمانی، بر اساس آن طراحی شده اند.

این نوع برنامه ها معمولاً شامل ترکیبی از یک سرور مرکزی و پایگاه داده نصب شده روی آن است که محیط لازم را برای درج، اصلاح و گزارش گیری از اطلاعات فراهم می کنند. معمولاً

در سمت دیگر برنامه های Client قرار دارند این برنامه ها باید توسط یک زبان برنامه نویسی مناسب تولید شوند.

وقتی شما یک برنامه نصب می کنید علاوه بر فایل های اصلی برنامه که در Program Files نصب می شود تعدادی فایل نیز در شاخه ویندوز نصب می شوند. این فایل ها همان درایور های مورد نیاز ویندوز برای اجرای برنامه شما هستند که توسط برنامه نویس نوشته نمی شوند بلکه آنها از قبل تعریف شده اند و برنامه نویس از قابلیت های آن در برنامه خود استفاده می کند. بنابراین برای اینکه برنامه شما کار کند باید آن فایل ها به سیستم عامل ویندوز شما اضافه شوند.

Framework مجموعه ای از فایل های مورد نیاز سیستم عامل شامل فایل های DLL، رجیستری و واسط های استاندارد ارتباط برنامه ها با یکدیگر است که برای اجرای برنامه های نوشته شده تحت NET الزامی می باشد.

چون NET می خواهد از فلسفه سادگی (Keep It Simple) پشتیبانی کند به همین دلیل اساس کار نصب برنامه ها Copy - Only Installation می باشد. یعنی دیگر نیازی به Package کردن برنامه ها توسط برنامه نویس و نصب توسط کاربر نیست بلکه تمامی فایل های کتابخانه ای مورد نیاز را Framework تامین می کند و برنامه ها با روش فقط کپی در سیستم عامل ویندوز کار می کنند.

شرکت Microsoft، Platform .NET را برای تولید نرم افزار های مختلفی که قابلیت اجرا در محیط های شبکه ای را داشته باشند ارائه داده است. زمانی که جهان آکنده از اطلاعات است و اطلاعات بصورت یک قالب واحد ارائه می شود. با گسترش و پیشرفت Cloud Computing ارزش و کارایی محیط های NET. به خوبی روشن خواهد شد و جایگاه واقعی محصولاتی که در این محیط ها تولید می شود برهمگان اثبات خواهد شد هر چند که این محیط هم اکنون نیز از محبوبیت بالایی در سرتاسر جهان برخوردار است.

NET ما را قادر می سازد تا نرم افزار های خود را به صورت کاربردی و به صورت سرویس های اینترنتی طراحی و پیاده سازی کنیم تا بدین وسیله اطلاعات خود را به اشتراک گذاشته و مدیریت کنیم.

در جدول ۱ - ۱ مهمترین لایه های Microsoft .NET را از سیستم عامل تا زبان های برنامه نویسی مشاهده می کنید که در پائین ترین لایه CLR و سپس سیستم عامل قرار گرفته اند.

Microsoft .NET Framework
.NET Language C#, VB, ...
Windows Application, Web Application, ...
XML Web Services ADO.NET - ASP.NET
CLR (Common Language Runtime)
Operating System

جدول ۱ - ۱

آشنایی با JIT

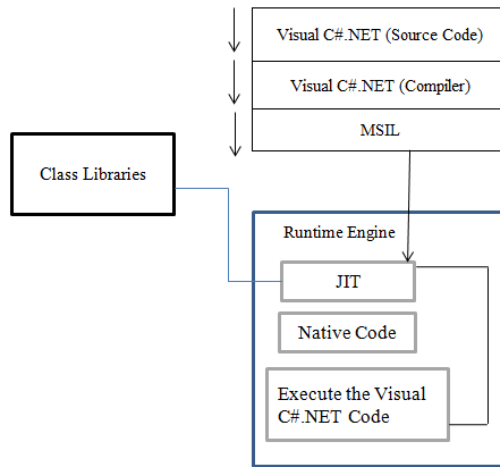
کامپایل شدن برنامه به زبان ماشین در دو مرحله صورت می گیرد. در مرحله اول سورس کد نوشته شده به یکی از زبان های NET توسط IDE به کد MSIL تبدیل می شود و در مرحله دوم در زمان Runtime کد IL توسط CLR در حافظه بار گذاری و اجرا می شود. در این مرحله کد IL ابتدا توسط چارچوب NET به زبان ماشین که اصطلاحاً در اینجا Native Code توسط بخشی از CLR که JIT Engine نامیده می شود صورت می پذیرد. اصطلاح JIT اشاره به روشی دارد که بخشی از CLR برای تبدیل کد IL به زبان ماشین به کار می رود. ایده اصلی پشت موتور JIT از این قرار است که اصولاً وقتی یک برنامه کاربردی با قسمت های گوناگون اجرا می شود و در اختیار کاربر قرار می گیرد، به ندرت تمام امکانات و قابلیت های آن همزمان و آن هم در ابتدای کار مورد استفاده وی قرار می گیرد. مثلاً یک برنامه کاربردی مانند Word دارای ابزار ها و امکانات متنوعی است. برخی از کاربران ممکن است هیچ گاه برخی از ابزار های خاص Word مثل ماکرو نویسی را به کار نبرند. هر یک از این ابزار ها و امکانات، قسمت های سورس کد مخصوص به خود را دارند. در برنامه های مدیریت شده تحت NET، به جای این که زمان سیکل های پردازنده و منابع حافظه صرف تبدیل تمام کد MSIL به Native Code شود، CLR از مکانیزمی استفاده می کند که تنها آن بخش از کد که در یک لحظه واقعاً مورد نیاز است به زبان ماشین ترجمه و تبدیل شود. این

روش را اصطلاحاً Just – In – Time Compiler می نامند. Native Code تولید شده به این ترتیب، در حافظه ذخیره می شود و در دفعات بعدی مورد استفاده قرار می گیرد. CLR یک قلاب یا گیره (مانند یک اشاره گر) به هر یک از انواع داده ها و متد ها، هنگام بار گذاری آنها به حافظه می چسباند. اولین بار که یک متد فراخوانده می شود گیره مورد نظر، کنترل آن تکه از کد MSIL را به کامپایلر JIT می سپارد و کامپایلر JIT نیز کد مزبور را به زبان ماشین تبدیل و برای اجرای پردازنده بهینه می سازد. سپس کامپایلر JIT گیره مورد نظر را به جای کد IL به کد زبان ماشین تهیه شده الصاق می کند تا در فراخوانی های بعدی آن متد، مورد استفاده قرار گیرد. تمام این فرآیند سبب کوتاه شدن زمان کامپایل شدن برنامه می شود. تبدیل کد به Native Code یکی از پیچیده ترین مراحل اجرای برنامه روی چارچوب .NET است. زیرا در این مرحله یک کد عمومی و مستقل از سیستم عامل و سخت افزار باید به کد زبان ماشین مناسب برای اجرا روی آن سیستم عامل و سخت افزار خاص تبدیل شود. از آنجایی که ترکیب انواع سخت افزار ها و سیستم عامل ها تنوع زیادی به وجود می آورند. در عمل به ندرت Native Code تولید شده در زمان اجرا روی یک کامپیوتر عیناً مشابه کد تولید شده روی یک ماشین دیگر است. خصوصاً اگر برنامه کاربردی مورد نظر از منابع گوناگونی اعم از سخت افزاری مانند صوت یا گرافیک و یا سرویس های سیستم عامل استفاده کرده باشد.

این که اجرای یک برنامه تحت .NET روی سیستم عامل ها و سخت افزار های گوناگون تا چه اندازه مشابه و از کارآیی یکسانی برخوردار باشد تا حدود زیادی به قدرت عملکرد و کامل بودن کامپایلر JIT باز می گردد. موتور JIT در اینجا اصطلاحاً یک Wrapper نامیده می شود. یعنی مانند یک بسته بندی و پوشش، ساختار واقعی مجموعه سیستم عامل و سخت افزار هر ماشین را از دید برنامه کاربردی پنهان می کند.

فرآیند کامپایل و اجرای سورس کد برنامه

اکنون به طور خلاصه می توان فرآیند کامپایل شدن و اجرای سورس برنامه را مرور کرد. تصویر ۱ - ۲ فلوچارت اجرای یک برنامه ویژوال C#.NET را نشان می دهد.



تصویر ۲ - ۱

ابتدا سورس برنامه توسط کامپایلر Visual C#.NET در IDE کامپایل می شود. وقتی که کامپایلر، کد MSIL را تولید می کند، Metadata را نیز می سازد. Metadata که شامل انواع داده های به کار رفته در کد و کلیه مشخصات برنامه و چگونگی وابستگی آن به سایر عناصر نرم افزاری است، داخل فهرستی به نام Manifest قرار می گیرد. همه این ها داخل فایلی به نام PE قرار می گیرند. فرمت این فایل Common Object File Format یا به اختصار COFF نامیده می شود. حضور Metadata در کنار کد MSIL کمک می کند تا این کد بتواند در زمان اجرا خصوصیات و مشخصات خود را شرح دهد و در اختیار CLR بگذارد. بنابراین هیچ نیازی به استفاده از Type Library نیست، چون موتور Runtime تمام اطلاعات مورد نیاز را از Metadata استخراج می کند. خروجی عمل کامپایل شدن در این مرحله یک اسمبلی است که معمولاً از یک مازول یا گاهی بیشتر تشکیل شده است. فایل DLL یا EXE حاصله از جنس PE است که برای اجرا شدن روی چارچوب .NET همه اطلاعات لازم را در خود دارد.

هنگامی که برنامه کاربردی حاصله توسط یک برنامه EXE یا هر سرویس یا برنامه اجرایی دیگر مثلاً برنامه های تحت وب فراخوانده می شود کد MSIL روی حافظه بار گذاری می شود و اطلاعات Metadata خوانده می شود و بقیه ی اطلاعات به موتور JIT سپرده می شود. در این مرحله چند کار توسط کامپایلر JIT صورت می گیرد. ابتدا اطلاعات BCL برای

کامپایل کردن کد IL به زبان ماشین خوانده می شود. JIT روتین ها و خواص استفاده شده از کلاس های BCL را برای تولید کد زبان ماشین نیاز دارد. سپس هر تکه از سورس کد که در زمان اجرا واقعاً لازم باشد کامپایل شده و پس از تبدیل به Native Code اجرا و برای استفاده های بعدی موقتاً و تا خاتمه کار برنامه و پاک سازی حافظه از اشیای مربوطه، ذخیره می شود. کد باید به عنوان بخشی از فرآیند کامپایل شدن MSIL به Native Code، مرحله ای به نام Verification Process را بگذرانند. (فرآیند تصدیق و تأیید) در این مرحله، کد IL و Metadata مربوط به آن، به دقت بررسی می شوند تا معلوم شود که انواع داده های تعریف شده در برنامه ایمن و معتبر هستند. به این مفهوم که سورس کد تنها به آن نواحی از حافظه می تواند دسترسی پیدا کند که مجاز است. کدی که به این ترتیب تصدیق و ایمنی آن تأیید می شود قادر به انجام دادن کار های خطرناک و دسترسی های غیر مجاز به نواحی مختلف حافظه و منابع سیستم نیست. این فرآیند به افزایش پایداری و ثبات برنامه و سیستم عامل می انجامد.

آشنایی با BCL

کتابخانه ی کلاس های پایه ای NET Framework. یا به اختصار BCL هرم سلسله مراتب ساختار کلاس های موجود در چارچوب NET را تعریف می کند و معماری شیء گرای آن را کامل می کند. کلاس های پایه چارچوب NET. طوری طراحی شده اند که برنامه نویسان به راحتی بتوانند در وجوه مختلف برنامه نویسی از آنها بهره گرفته و مجموعه کلاس های مورد نظر خود را از آنها تولید نمایند.

در این کتابخانه انواع داده ها نیز تعریف شده اند. کتابخانه کلاس های پایه ای NET Framework یک مجموعه کامل از کلاس ها را برای انجام امور عمومی در برنامه نویسی شامل مدیریت رشته های متنی، دستکاری داده ها، ایجاد ارتباط با بانک های اطلاعاتی، دسترسی به فایل ها و غیره را فراهم آورده است.

علاوه بر کلاس های عمومی، چارچوب NET. مجموعه ای از کلاس های دیگر را نیز که مرتبط با آخرین فناوری های جدید هستند، فراهم آورده است. برنامه نویسان می توانند از این کلاس ها با مشتق کردن اشیاء از آنها استفاده نمایند و از بعضی از آنها برای استفاده

خاص طراحی شده در برنامه های کاربردی خود استفاده یا به تولید کلاس های جدید از این کلاس های پایه و اضافه کردن قابلیت جدید به آنها، به اشیاء جدید مورد نظر خود دست یابند.

گروه کلاس های System ریشه تمامی دیگر کلاس های پایه NET می باشد. کتابخانه پایه ای BCL طوری طراحی شده اند که قابلیت کار کردن با اشیاء غیر NET (سرویس هایی که در سطح سیستم عامل موجودند) را نیز داشته باشند. BCL از نظر ساختاری مشابه کتابخانه ی MFC در Platform قبلی مایکروسافت است که مورد استفاده برنامه نویسان زبان ++C قرار داشت.

آشنایی با CTS

جهت پشتیبانی از ویژگی چند زبانی از CTS استفاده می شود. CTS شامل مجموعه ای از انواع داده ها و استاندارد هایی جهت ایجاد کلاس های سفارشی کاربر می باشد. مزایای CTS به شرح زیر می باشند :

- ۱- ارث بری یک کلاس از کلاس دیگری که توسط زبان دیگری نوشته شده است.
- ۲- ایجاد شیء از کلاسی که توسط زبان دیگری نوشته شده است.
- ۳- استفاده از شیء یا اشاره گری به شیء به عنوان پارامتر متد کلاسی که توسط زبان دیگری نوشته شده است.
- ۴- اشکال زدایی برنامه که شامل اشیاء و کلاس هایی است که در زبان دیگری نوشته شده است.

CTS مجموعه انواع داده های تعریف شده در چارچوب NET را در بر می گیرد. تمام انواع داده ها از نوع داده System.Object مشتق شده اند، بنابراین هر موجودیت در معماری NET در واقع خود یک شیء است. انواع داده ها در CTS به دو دسته تقسیم می شوند :

- ۱- انواع در برگیرنده مقدار (Value Types) : آن دسته از داده هایی هستند که مستقیماً مقداری را در خود دارند (یعنی به محتوای خودشان اشاره می کنند) این مقادیر ممکن است به طور مستقیم در پشته حافظه قرار گیرند یا داخل یک ساختار داده تعریف شده باشند. انواع

داده های Value ممکن است به صورت انواع خاص مورد استفاده CLR در زمان اجراء، تعریف شده توسط برنامه نویس یا از نوع شمارشی باشند.

۲- انواع در برگیرنده مرجع (Reference Types) : انواع داده های Reference، آنهایی هستند که مستقیماً مقداری ندارند، بلکه به آدرسی از حافظه اشاره می کنند که در برگیرنده یک مقدار است. بنابراین ممکن است هم زمان بیش از یک متغیر از نوع مرجع، به یک مقدار یکسان اشاره کند. انواع مرجع ممکن است به صورت خود تعریف یا Self – Describing باشند، مانند نوع کلاس، آرایه ها. که ممکن است به صورت اشاره گر باشند.

آشنایی با زبان برنامه نویسی C#.NET

زبان برنامه نویسی C#.NET از دو زبان ++C و Java متولد شده است و حاوی بسیاری از جنبه های زبان ++C است. اما ویژگی شیء گرایی خود را از زبان برنامه نویسی Java به ارث برده است. هر دو زبان نام برده شده جزء زبان های Hybrid محسوب می شوند اما طراحان زبان C# این مورد را به اندازه زبان ++C مهمی تلقی نکرده اند.

اگر تا به امروز هیچ برنامه ای ننوشته اید ممکن است این کار مشکل به نظر برسد اما زبان C#.NET سعی کرده است این موضوع را تا حد ممکن ساده سازد و به شما اجازه دهد کار های بسیار مشکل را به آسانی انجام دهید. برنامه هایی که با این زبان نوشته می شوند می توانند بر روی سیستم عامل ویندوز اجرا شوند. می توانید با این زبان برنامه های قابل اجرا بر روی اینترنت، برای دستگاههای رایانه های همراه نظیر موبایل و PDA و غیره برنامه بنویسید.

زبان C# از پایه برای استفاده در محیط NET. ایجاد شده است. C# یکی از ۴۴ زبان برنامه نویسی است که توسط Common Language Runtime از NET Framework پشتیبانی می شوند و در همه جا به وسیله Microsoft Visual Studio شناخته می شود. دستورات این زبان همانند زبان های خانواده C به سمی کالن (;) ختم می شود.

دستورات زبان C# به حروف حساس است، یعنی بین حروف کوچک و بزرگ تفاوت قائل می شود.

همانطور که گفتیم زبان C#.NET از فناوری شیء گرایی استفاده می کند برای استفاده از این فناوری باید بتوان از کلاس استفاده کرد. کلاس ها برای ایجاد انواع جدید به کار می روند و شامل متد ها و داده هایی است که داده ها را دستکاری می کنند.

در زبان C#.NET کلاس ها به دو دسته کلی تقسیم می شوند :

۱- کلاس های آماده : این کلاس ها از قبل نوشته و در کتابخانه FCL نرم افزار Visual Studio قرار دارد. این کلاس ها در فضا های نام مختلف قرار می گیرند. برخی از این فضا های نام بطور خودکار به برنامه اضافه می شوند به این منظور از دستور using به صورت using System.Data.SqlClient استفاده می کنیم.

۲- کلاس های نوشته شده توسط برنامه نویس : به خاطر اینکه همه کلاس های مورد نیاز برنامه نویسان از قبل وجود ندارد لذا برنامه نویس با توجه به نیاز خود برخی از کلاس ها را خودش می نویسد.

مفهوم شیء گرایی

یکی از مباحثی که در روند های برنامه نویسی زیاد به گوش می رسد مفهوم شیء گرایی یا OOP است. مرتباً می شنوید که آیا از شیء گرایی اطلاع دارید؟ آیا با روند های شیء گرایی آشنایی دارید؟

بعضی ها در مورد مساله شیء گرایی تصورات اشتباهی دارند مثلاً قرار دادن یک کنترل ساده مانند یک TextBox روی Form طراحی را برنامه نویسی شیء گرا در نظر می گیرند. درست است که این کار با روند های شیء گرایی انجام می شود اما مفهوم واقعی شیء گرایی این نیست.

خیلی ساده و روان به بررسی مفهوم شیء گرایی می پردازیم :

اصولاً از خودتان تا به حال پرسیده اید که اصلاً شیء چیست؟ جواب این سوال ساده است، همان اشیائی است که شما در دنیای واقعی با آنها سر و کار دارید. شما در عالم واقعی به مداد، خودکار، ماشین و هر وسیله جامدی که بتوانید لمس کنید و جان نداشته باشد شیء می گوئید. ولی در عالم برنامه نویسی اینگونه نیست ممکن است یک انسان هم شیء باشد بطور مثال دانشجو برای دانشگاه در حکم یک شیء است چون گزینه ای است که می تواند در روند

های برنامه نویسی مورد استفاده قرار بگیرد. هر شیء دارای خواص و رفتار است مانند Size, Color و ... هر شیء ای که خاصیت داشته باشد بالطبع رفتار نیز دارد.

خوب بطور مثال الان شما شیء دانشجو را در اختیار دارید شما بایستی آن را وارد دنیای مجازی کرده و برنامه نویسی کنید. برای این کار بهترین حالت آن است که برای هر شیء یک کلاس تصور کنید زمانی که هر شیء را در یک کلاس تصور می کنید می توانید آن را در دنیای مجازی پیاده سازی کرده و برای این کلاس خود با روند های برنامه نویسی رفتار و خاصیت تعریف کنید.

این دانشجو ممکن است خاصیت های متفاوتی داشته باشد مانند نام، فامیلی، شماره دانشجویی و ... این دانشجو ممکن است رفتار های متفاوتی هم داشته باشد. منظور رفتار هایی نیست که خودش انجام می دهد هدف رفتار هایی است که نسبت به آن شیء انجام می شود مانند ثبت نام کردن، اخراج شدن، فارغ التحصیل شدن و ...

تمامی کنترل ها و ابزار هایی که در محیط Visual Studio وجود دارند در حکم یک شیء هستند زیرا خاصیت و رفتار دارند. وقتی شما از این کنترل ها استفاده می کنید یک سری کد بصورت خودکار Generate می شوند و کلاس ها با یکدیگر تعامل دارند و ارتباط برقرار می کنند. در فصل های بعدی بطور کامل با این مفاهیم درگیر می شوید و دید وسیعی نسبت به مباحث شیء گرایی پیدا می کنید.

انواع داده های اولیه در زبان C#.NET

به مقادیری که به عنوان ورودی یک برنامه از طریق دستگاه های ورودی وارد می شوند داده ها گفته می شود. داده ها پس از ورود به رایانه، پردازش و از طریق دستگاه های خروجی به عنوان نتایج برنامه ارائه می شوند که به این نتایج اطلاعات گفته می شود. انواع داده های اولیه را در زبان C#.NET در جدول ۲ - ۱ می توانید مشاهده کنید.

نوع	اندازه به بایت	شرح
byte	۱	بدون علامت (۰ تا ۲۵۵)
char	۱	کاراکتر های Unicode
bool	۱	مقادیر true یا false
sbyte	۱	علامت دار (۱۲۷- تا ۱۲۸)
short	۲	مقادیر ۳۲,۷۶۸- تا ۳۲,۷۶۷
ushort	۲	مقادیر ۰ تا ۶۵۵۳۵
int	۴	مقادیر بیتی ۲,۱۴۷,۴۸۳,۶۴۷- تا ۲,۱۴۷,۴۸۳,۶۴۷
uint	۴	مقادیر بین ۰ تا ۴,۲۹۴,۹۶۷,۲۹۷
float	۴	اعداد اعشاری تا ۷ رقم با ارزش
double	۸	اعداد اعشاری تا ۱۶ رقم با ارزش
decimal	۸	ممیز ثابت تا ۲۸ رقم و مکان نقطه اعشار
long	۸	مقادیر صحیح از ۹,۲۲۳,۳۷۲,۰۳۶,۸۵۴,۷۷۵,۸۰۷ تا ۹,۲۲۳,۳۷۲,۰۳۶,۸۵۴,۷۷۵,۸۰۸
ulong	۸	اعداد صحیح از ۰ تا ۰xffffffffffffffff

جدول ۱ - ۲

مقدار حافظه ای که انواع داده های مختلف به خود اختصاص می دهند یکسان نیست. با نگاه کردن به یک عدد نمی توان گفت که چقدر حافظه اشغال کرده است. به عنوان مثال، اعداد ۲۹.۹۹۹ و ۷۰۱ هر دو یک مقدار حافظه اشغال می کنند. با این که امروزه دیگر حافظه، یک مشکل کلیدی به حساب نمی آید و روی سیستم های رایانه ای به میزان کافی وجود دارد و شما به عنوان برنامه نویس نباید نگران آن باشید ولی همیشه سعی کنید برای داده هایتان نوعی را انتخاب کنید که حافظه کمتری را اشغال کند.

انواع داده های عددی در زبان برنامه نویسی C#.Net، مقدار حافظه مورد نیاز هر کدام و محدوده ای از اعداد را که می توانند در خود جای دهند را در جدول ۱ - ۲ می توانید مشاهده کنید. هنگام تعریف داده ها این جدول را مد نظر داشته باشید.

ذکر این نکته لازم است که یادگیری محدوده اعداد موجود در این جدول، ضرورتی ندارد و حتی برنامه نویسان حرفه ای نیز ممکن است محدوده اعداد مربوط به هر نوع داده عددی را ندانند و در صورت نیاز به چنین جدولی رجوع کنند.

تعریف متغیر

همانطور که می دانید متغیر محلی از حافظه است که برای نگهداری موقت داده ها و اطلاعات، مورد استفاده قرار می گیرد. و به عنوان ابزاری برای کار بر روی اطلاعات در اختیار مجری الگوریتم است. به عبارت دیگر، می توان گفت که برای هر مکان از حافظه نامی در نظر می گیریم و به آن متغیر می گوئیم.

برای نامگذاری متغیر ها در زبان C#.NET می توان از ترکیب حروف، اعداد و _ استفاده کرد به شرطی که حرف اول متغیر با رقم شروع نشود.

بعد از تعریف متغیر باید نوع آنها را تعریف کنید یعنی متغیر چه نوع داده هایی را ذخیره کند.

اعلان متغیر ها در زبان C#.Net به صورت زیر است :

نام متغیر نوع داده نوع دسترسی

سطح دسترسی تعیین می کند که در چه محدوده هایی می توان به متغیر ها دست یافت. مهمترین آنها Public و Private هستند. Public موجب دسترسی به متغیر در کل کلاس می شود و Private موجب می شود تا متغیر را بتوانید در همان بلاک دستیابی داشته باشید. در صورتی که سطح دسترسی تعیین نگردد Private در نظر گرفته می شود.

int x;	تعریف متغیر x از نوع صحیح
double d;	تعریف متغیر d از نوع double
string sa , sb;	تعریف متغیر sa و sb از نوع string

البته در زمان تعریف متغیر ها می توان به آنها مقدار داد.

```
int x = ۵ , y = ۱۰;  
string s = "good";
```

ثوابت

ثوابت مقادیر در برنامه هستند که در طول اجرای برنامه ثابت هستند و تغییر مقدار آنها امکان پذیر نیست. برای تعریف ثابت ها در زبان C#.NET می توان از کلمه کلیدی `const` و یا دستور `#define` استفاده کرد.

برای تعریف ثوابت با دستور `#define` به صورت زیر عمل می شود :

```
#define sum 200
#define P 3.14
```

همانطور که در دستورات بالا مشاهده می کنید ثوابت `sum` با مقدار ۲۰۰ و `P` با مقدار ۳.۱۴ با دستور `#define` تعریف شده است. دقت داشته باشید که در انتهای دستور `#define` علامت `;` قرار نمی گیرد. به خاطر این که دستور `#define` از دستورات پیش پردازنده است، نه از دستورات زبان `C#`. پیش پردازنده نوعی مترجم است که دستورات توسعه یافته ای را از زبان برنامه سازی گرفته و به دستورات قابل فهم برای کامپایلر همان زبان تبدیل می کند.

برای تعریف ثوابت با دستور `const` به صورت زیر عمل می شود :

```
const مقدار = نام ثابت نوع داده
```

دستورات زیر را در نظر بگیرید :

```
const int a = 20 , int b = 30;
const char s = 'a';
```

ثوابت `a` و `b` از نوع `int` و با مقادیر اولیه ۲۰ و ۳۰ تعریف شده اند. و ثابت `s` از نوع `char` و با مقدار اولیه `a` تعریف شده است.

اگر پس از تعریف ثوابت، در ادامه برنامه سعی کنید مقادیر آنها را عوض کنید، کامپایلر خطایی را اعلان خواهد کرد.

ثوابت دارای ویژگی های مهمی هستند که به ترتیب عبارتند از :

۱- بایستی در هنگام تعریف مقدار دهی شوند، زمانی که یک مقدار به آنها انتساب داده می شود، دیگر نمی توان مقدار آنها را تغییر داد.

۲- مقدار یک ثابت باید در زمان کامپایل قابل محاسبه باشد، بنابراین نمی توان یک ثابت را با مقدار یک متغیر، مقدار دهی کرد. اگر بخواهید این کار را انجام دهید باید از فیلد های فقط خواندنی استفاده کنید. (مباحث مربوط به فصل ۵).

۳- ثابت ها همواره static هستند. البته توجه داشته باشید که در مورد آنها از کلمه کلیدی static استفاده نکنید، این کار مجاز نیست.

۴- ثوابت، برنامه شما را با جایگزین کردن نام های خوانایی که مقادیر آنها بسیار قابل فهم است با اعداد و رشته های غیر قابل فهم، خواناتر می کنند.

۵- ثوابت، اصلاح برنامه شما را آسان تر می کنند. به طور مثال فرض کنید که در برنامه C# خود، یک ثابت به نام SalesTax دارید و این ثابت دارای مقدار ۶ درصد است اگر میزان مالیات در پایان ایجاد برنامه تغییر کند شما به سادگی می توانید با اعمال مقدار جدید به ثابت، رفتار محاسبه مالیات را تغییر دهید. دیگر نیازی به جستجو برای یافتن مقادیر ۰.۰۶ و تغییر هر یک از آنها نیست.

۶- ثابت ها بروز اشتباه در برنامه را کاهش می دهند. اگر در قسمتی از برنامه، غیر از جایی که ثابت ها تعریف شده است سعی کنید که به یک مقدار ثابت، مقدار دیگری بدهید کامپایلر پیغام خطا خواهد داد.

عملگر ها

عملگر ها نماد هایی هستند که اعمال خاصی را بر روی عملوند ها انجام می دهند. در زبان C# انواع مختلفی از عملگر ها وجود دارد که با توضیح کوتاهی به معرفی آنها می پردازیم :

عملگر های محاسباتی : این عملگر ها، اعمال محاسباتی را بر روی عملوند ها انجام می دهند.

عملگر های رابطه ای : عملگر هایی هستند که دو عملوند را با یکدیگر مقایسه می کند.

عملگر های منطقی : عملگر هایی هستند که بر روی عبارت منطقی عمل می کنند.

عملگر های ترکیبی : از ترکیب عملگر های محاسباتی و علامت = ایجاد می شوند.

عملگر های بیتی : عملگر هایی هستند که برای تست کردن، مقدار دادن یا شیفت دادن و سایر اعمال بر روی عملوند ها بکار می روند.

در جداول زیر انواع عملگر های محاسباتی، رابطه ای، منطقی، ترکیبی و بیتی را مشاهده می کنید.

عملگر	نام
>	بزرگتر
>=	بزرگتر مساوی
<	کوچک تر
<=	کوچک تر مساوی
=	مساوی
!=	نا مساوی

جدول ۴ - ۱ عملگر های رابطه ای

عملگر	نام
-	تفریق و منهای یکانی
+	جمع
*	ضرب
/	تقسیم
%	باقیمانده تقسیم
--	عملگر کاهش
++	عملگر افزایش

جدول ۳ - ۱ عملگر های محاسباتی

عملگر	نام
&	و
	یا
^	یای انحصاری
~	نقیض
>>	شیفت به راست
<<	شیفت به چپ

جدول ۶ - ۱ عملگر های بیتی

عملگر	نام
+ =	انتساب جمع
- =	انتساب تفریق
* =	انتساب ضرب
/ =	انتساب تقسیم
% =	انتساب باقیمانده تقسیم

جدول ۵ - ۱ عملگر های ترکیبی

عملگر	نام
!	نقیض
&&	و
	یا

جدول ۷ - ۱ عملگر های منطقی

عملگر های & و *

با استفاده از عملگر & می توانیم به آدرس متغیر ها دسترسی داشته باشیم. عملگر * نیز برای دسترسی غیر مستقیم به حافظه مورد استفاده قرار می گیرد. به عنوان مثال، اگر فرض کنیم آدرس متغیر x در p قرار داشته باشد، از طریق آدرس x و با استفاده از عملگر * می توانیم به محتویات x دسترسی پیدا کنیم. با فرض این که p می تواند حاوی آدرس x باشد، دستورات

زیر را در نظر بگیرید :

$p = x$;

$*p = 5$;

$m = *p$;

در دستور اول آدرس متغیر x در p قرار می گیرد. در دستور دوم، جایی که آدرس آن در p است (همان x)، برابر با ۵ می شود. در دستور سوم، محتویات محلی که آدرسش در p است (۵) در m قرار می گیرد.

عملگر ؟

این عملگر، عبارتی را ارزیابی کرده براساس ارزش آن عبارت (درستی یا نادرستی)، نتیجه عبارت دیگر را در متغیری قرار می دهد :

عبارت ۲ : عبارت ۱ ؟ عبارت ۲ = متغیر

اگر عبارت ۱ دارای ارزش درستی باشد، مقدار ارزیابی شده در عبارت ۲ در متغیر قرار می گیرد، و گرنه مقدار ارزیابی شده در عبارت ۳ در متغیر قرار خواهد گرفت. دستورات زیر را در نظر بگیرید :

$\text{int } x, y$;

$x = 5$;

$y = x > 5 ? x * 2 : x * 5$;

عبارت ۱ دارای ارزش نادرستی است، زیرا x از ۵ بزرگتر نیست. بنابراین مقدار موجود در عبارت ۳ که برابر با $5 * 5 = 25$ است، در متغیر y قرار می گیرد.

عملگر کاما (,)

این عملگر برای انجام چند عمل در یک دستور به کار می رود. و کاربرد آن به صورت زیر است :

متغیر = (عبارت ۱ , عبارت ۲) ;

عبارت ۱ به نحوی با عبارت ۲ در ارتباط است. به طوری که، ابتدا عبارت ۱ ارزیابی می شود و عبارت ۲ می تواند از نتیجه آن استفاده کند و حاصل عبارت ۲ در متغیر قرار می گیرد. دستورات زیر را در نظر بگیرید :

```
int x , y ;
```

```
y = (x = 2 , x * 4 / 2) ;
```

در عبارت ۱، مقدار ۲ در x قرار می گیرد و این مقدار در محاسبه عبارت $x * 4 / 2$ شرکت کرده حاصل آن، یعنی ۴ در y قرار می گیرد.

عملگر sizeof

شما می توانید اندازه مورد نیاز یک نوع مقدار روی پشته را با استفاده از عملگر sizeof مشخص کنید.

```
Console.WriteLine (sizeof (int)) ;
```

خروجی کد بالا مقدار ۴ خواهد بود زیرا یک نوع int چهار بایت است.

عملگر پرانتز

وقتی در یک عبارت چند عمل وجود دارد، ترتیب انجام آنها به تقدم عملگرها بستگی دارد. عملگرهای C# از نظر تقدم اجرا به چند دسته تقسیم می شوند. برای مثال، عملگر $*$ به عملگر $+$ تقدم دارد، یا عملگرهای محاسباتی و چسباندن رشته قبل از عملگرهای مقایسه و منطقی انجام می شوند (عملگرهایی که دارای تقدم یکسان هستند، از چپ به راست انجام خواهند شد). از عملگر پرانتز معمولاً برای جدا سازی شرط ها از یکدیگر استفاده می شود. اما کاربرد دیگر این عملگر این است که باعث افزایش الویت شرط هایی می شود که در داخل این عملگر قرار خواهد گرفت. برای مثال به عبارت زیر توجه کنید :

$$5 + 3 - 2 * 4 = 0$$

یعنی با توجه به ترتیب الویت های عملگرهای به کار رفته در این عبارت، چون الویت ضرب از جمع و تفریق بیشتر است ابتدا $2 * 4 = 8$ محاسبه می شود و بعد چون الویت عملگرهای جمع و ضرب با هم برابر است از چپ به راست عمل می کنیم، یعنی ابتدا $5 + 3 = 8$ محاسبه می شود و بعد حاصل عبارت اول از عبارت دوم کم (تفریق) می شود و نتیجه برابر با عدد صفر خواهد شد یعنی $8 - 8 = 0$.

ولی اگر همین عبارت را با استفاده از پرانتزها به صورت زیر بنویسیم داریم :

$$5 + (3 - 2) * 4 = 9$$

چون الویت پرانتز از هر عملگری بالاتر است، پس ابتدا عبارت داخل پرانتز محاسبه می شود یعنی $1 = 2 - 3$. حال از میان دو عملگر باقیمانده یعنی جمع و ضرب، چون الویت ضرب از جمع بیشتر است ابتدا حاصل عبارت داخل پرانتز یعنی عدد ۱ در عدد ۴ ضرب می شود $1 = 4 * 4$. و در نهایت این عدد با عدد ۵ جمع می شود یعنی $9 = 5 + 4$.

الویت عملگر ها

اگر در یک عبارت از عملگر هایی استفاده کردید که از انواع مختلف این سه نوع می باشند، الویت اجرای آنها به ترتیب زیر می باشد :

- ۱- عملگر های محاسباتی
- ۲- عملگر های رابطه ای
- ۳- عملگر های منطقی
- ۴- عملگر های بیتی
- ۵- عملگر های ترکیبی

نکته!

- ۱- در میان تمام عملگر ها، بالاترین الویت با پرانتز است. و عملگر های یکتایی الویت بالاتری نسبت به عملگر های دو تایی دارند.
- ۲- اگر در عبارتی بیشتر از یک جفت پرانتز داشته باشیم، الویت پرانتز ها در عبارت از چپ به راست است.
- ۳- اگر در عبارتی چند عملگر با الویت یکسان داشته باشیم، ترتیب الویت آنها از چپ به راست است.

در صفحه بعد، لیست کاملی از عملگر های موجود در زبان C# به همراه الویت آنها در عبارات ارائه شده است.

بالاترین تقدم [] ()

sizeof ++ ~ !

یکانی -

* / %

+ -

<< >>

< <= > >=

== !=

&

~

|

&&

||

?

= += -= *= /= %=

پایین ترین تقدم ,

تبدیل انواع

بدیهی است که در یک برنامه C#، انواع مختلف با هم ترکیب می شوند. به عنوان مثال، ممکن است بخواهید یک نوع صحیح (int) را با نوع اعشاری (float) با هم جمع کنید. دستورات زیر را در نظر بگیرید :

```
int a = 10 ;
```

```
float b , c = 20.4 ;
```

```
b = a + c ;
```

دستور اول متغیر a را از نوع int و با مقدار ۱۰، و دستور دوم متغیرهای b و c را از نوع float تعریف می کند و مقدار c را برابر با ۲۰.۴ تعیین می کند. دستور سوم a را با c جمع می کند و نتیجه را در b قرار می دهد. قبل از این که a با c جمع شود، ابتدا از نوع صحیح به نوع اعشاری تبدیل می گردد تا دو مقدار هم نوع با هم جمع شوند.

این نوع تبدیل را که بدون دخالت برنامه نویس صورت می گیرد، تبدیل ضمنی می نامند. اکنون مثال زیر را در نظر بگیرید :

```
string s = "211" ;
```

```
int y , x = 12 ;
```

دستور اول رشته s را با مقدار رشته ای "۲۱۱" و دستور دوم متغیرهای y و x را تعریف می کند که مقدار x برابر با ۱۲ است. حال دستورات زیر را تصور کنید :

```
y = x + s ;
```

این دستور با خطا مواجه می شود، زیرا سیستم نمی تواند s را که از نوع رشته ای است به نوع صحیح تبدیل کند و حاصل جمع آن با x را در y قرار دهد. در چنین مواردی برنامه نویس باید با امکانات موجود در زبان، ابتدا نوع s را از رشته ای به صحیح تبدیل کند و سپس با x جمع کرده حاصل را در y قرار دهد. این نوع تبدیل را تبدیل صریح می نامند.

تبدیلات غیر صریح

تبدیلات غیر صریح در زبان C# از دو جهت قابل بررسی است :

۱- تبدیل نوع در عبارات محاسباتی، مانند :

```
z = x + y * m + k ;
```

۲- تبدیل نوع در احکام انتساب، مانند :

```
x = y ;
```

در مورد تبدیل انواع در عبارات محاسباتی این قانون حاکم است که نوع کوچکتر مانند int به نوع بزرگتر مانند float تبدیل می گردد. جدول ۸ - ۱ تبدیلات غیر صریح پشتیبانی شده در زبان C# را نشان می دهند :

از	به
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, 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, double	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

جدول ۸ - ۱

تبدیلات صریح

گاهی لازم است تبدیل نوع صریح توسط برنامه نویس صورت گیرد. به عنوان مثال، نوع رشته ای به نوع عددی تبدیل شود یا بر عکس. برای این منظور از متد هایی که قبلا نوشته شدند و در زبان C# وجود دارند استفاده می شود. متد هایی که در زبان C# برای تبدیل نوع به کار می روند، مربوط به شیء Convert می باشند که بعضی از آنها عبارتند از :

ToInt^{۱۴}, ToInt^{۳۲}, ToInt^{۱۶}, ToDouble, ToDecimal, ToChar, ToByte, ToBoolean, ToString, ToSingle, ToSByte.

مثال زیر را ببینید :

```
string s = "112" ;
```

```
int x ;
```

```
x = Convert.ToInt32 (s) ;
```

دستور آخر، رشته S را به مقدار عددی ۱۱۲ تبدیل کرده و در x قرار می دهد. متد ToInt^{۳۲} به صورت زیر نیز قابل استفاده است :

```
x = Convert.ToInt32 (s, 10) ;
```

پارامتر دوم این متد یک مقدار عددی است که مبنای عددی را مشخص می کند و به صورت رشته ای در S ذخیره شده است. در مثال فوق مشخص می کند که مبنای مقدار عددی رشته S برابر با ۱۰ است.

کلمات کلیدی

کلمات کلیدی یا کلمات رزرو شده، کلماتی هستند که در زبان C# دارای معنای خاصی می باشند و با حروف کوچک نوشته می شوند. زبان برنامه نویسی C# دارای ۷۷ کلمه کلیدی است. این کلمات در کامپایلر به رنگ آبی نمایش داده می شوند. لیست تمام کلمات رزرو شده در C# در جدول ۹ - ۱ آمده است :

abstract	as	base	char	default
byte	case	catch	decimal	enum
class	const	continue	else	finally
delegate	do	double	false	goto
event	explicit	extern	foreach	interface
fixed	float	for	int	namespace
if	implicit	int	long	out
internal	is	lock	operator	public
new	null	object	protected	sealed
override	params	private	sbyte	string
readonly	ref	return	static	true
short	sizeof	stackalloc	throw	unchecked
struct	switch	this	ulong	void
try	typeof	unit	virtual	
unsafe	ushort	using	break	
while	volatile	bool	checked	

جدول ۹ - ۱

ساختار های کنترلی

در برنامه های ساده، دستورات برنامه به صورت ترتیبی اجرا می شوند. گاهی نیاز است بعضی از دستورات چندین بار اجرا شوند و یا تحت شرایطی خاصی اجرا شوند. برای پیاده سازی چنین دستوراتی از ساختار های کنترلی استفاده می شود. انواع ساختار های کنترلی عبارتند از :

۱- ساختار های ترتیب

۲- ساختار های تصمیم

۳- ساختار های تکرار

ساختار های تصمیم

تصمیم گیری، از مهمترین عملیاتی است که در برنامه نویسی استفاده می شود. هر برنامه در شرایطی که منطق برنامه ایجاب می کند، تصمیمی را اتخاذ می نماید که ادامه روند کار، به

نتیجه همین تصمیم بستگی خواهد داشت. در برخی از برنامه ها، لازم است براساس شرایطی تصمیم گرفته شود که چه مجموعه ای از دستورات باید اجرا شوند و چه مجموعه ای از دستورات نباید اجرا شوند. برای پیاده سازی این تصمیمات از ساختارهای تصمیم استفاده می شود. از جمله این ساختارها در زبان C#.NET دستورهای if و switch است.

ساختار تصمیم if

این ساختار یک عبارت شرطی را بررسی می کند در صورتی که نتیجه بررسی شرط دارای ارزش درستی باشد مجموعه ای از دستورات اجرا می شوند و در غیر این صورت مجموعه دیگری از دستورات اجرا خواهند شد. این ساختار به صورت زیر بکار می رود.

```
if (عبارت شرطی)
{
    ;مجموعه دستورات ۱
}
else
{
    ;مجموعه دستورات ۲
}
```

ساختار if تو در تو

گاهی ممکن است که بخواهید شرط های متعددی را بررسی کنید برای این منظور باید از if های تو در تو استفاده کنیم. این روش باعث طولانی تر شدن برنامه می شود و از خوانایی برنامه می کاهد.

```
if (شرط ۱)
; دستور ۱
else if (شرط ۲)
; دستور ۲
else if (شرط ۳)
; دستور ۳
else if (شرط n)
; دستور n
else
; دستور
```

ساختار switch

علاوه بر دستور if، از دستور switch نیز می توان برای تصمیم گیری استفاده کرد. هنگامی که بخواهیم براساس مقادیر مختلف یک متغیر یا یک عبارت، تصمیم گیری های مختلفی داشته باشیم از این ساختار استفاده می کنیم. به طور کلی پیشنهاد می شود که هر وقت تعداد انتخاب ها براساس مقادیر مختلف یک عبارت یا متغیری، بیش از سه باشد از ساختار switch استفاده کنید. زیرا if های تو در تو از خوانایی برنامه می کاهد. دستور switch به صورت زیر مورد استفاده قرار می گیرد.

(عبارت) switch

```
{
  case مقدار ۱ :
    دستورات ۱
    break ;
  case مقدار ۲ :
    دستورات ۲
    break ;
    ...
  default :
    دستورات n
    break ;
}
```

ابتدا عبارت موجود در مقابل switch به مقدار صحیح ارزیابی می شود و مقدار آن تعیین می گردد. اگر این مقدار با مقدار ۱ برابر بود دستورات ۱ اجرا می شوند و دستور break که بعد از آنها قرار دارد کنترل برنامه را از ساختار switch خارج می کند. اگر با مقدار ۱ برابر نبود با مقدار ۲ مقایسه می شود و اگر با این مقدار برابر بود دستورات ۲ اجرا می شوند و دستور break که بعد از آنها قرار دارد کنترل برنامه را از ساختار switch خارج می کند. تا زمانی که عبارت محاسبه شده با یکی از مقادیر ذکر شده برابر نبود عمل مقایسه با مقادیر بعدی ادامه می یابد. چنانچه مقدار عبارت با هیچ کدام از مقادیر برابر نبود دستورات n اجرا می شوند و کنترل از ساختار switch خارج می گردد.

ساختار های تکرار

مفهوم ساختار های تکرار را با چند مثال شرح می دهیم. فرض کنید در یک کارخانه تولیدی، کارگری را استخدام کرده و به وی می گوید که برچسب خاصی را بر روی جعبه محصولات کارخانه بچسباند. روزانه ممکن است هزاران محصول جدید تولید شود و این کارگر وظیفه خود را انجام دهد. نیاز نیست که برای چسباندن هر برچسب به کارگر بگویید که چه کاری را انجام دهد. فقط کافی ست همان بار اول، این عمل را به او تفهیم کنید تا بتواند در طول روز و روز های بعد، این عمل تکراری را انجام دهد. مثال هایی از این قبیل، در زندگی روزمره هر فرد نیز به وفور یافت می شود.

در انجام عملیات رایانه ای نیز، بعضی از اعمال به دفعات تکرار می شوند و بسیاری از کار ها هستند که ماهیت آنها تکراری ست. به عنوان مثال، خواندن نام و معدل ۵۰ دانشجو از ورودی، خواندن ساعات اضافه کاری ۳۰ کارمند، خواندن نمرات دانش آموزان و محاسبه معدل وی و هزاران مثال دیگر، نمونه هایی از عملیات تکراری هستند.

برای انجام این گونه عملیات، در تمام زبان های برنامه نویسی، ساختار هایی وجود دارند که سبب تکرار اجرای دستورات می شوند. به این گونه ساختار ها (حلقه های تکرار) می گویند. حلقه های تکرار را می توان دو گروه عمده تقسیم بندی کرد : حلقه های تکرار معین و حلقه های تکرار نامعین.

حلقه تکرار معین، حلقه ای است که تعداد دفعات تکرار آن مشخص است و با مقادیر متغیری به نام شمارنده تعیین می شود. مانند حلقه for.

حلقه تکرار نامعین، حلقه ای است که تعداد دفعات تکرار آن مشخص نیست و براساس درستی یا نادرستی شرطی، حلقه ادامه می یابد. مانند حلقه while.

حلقه تکرار for

همانطور که می دانید با استفاده از دستورات if نیز می توان یک حلقه تکرار ایجاد کرد تا عملیات خاصی را به تعداد دفعات معین تکرار کنند. ولی اغلب برنامه نویسان ترجیح می دهند که از دستور if برای اینگونه عملیات ها استفاده نکنند و برای اجرای دستورات مشخصی از برنامه به تعداد دفعات معین، از ساختار حلقه for استفاده کنند.

در این ساختار متغیری وجود دارد که تعداد دفعات تکرار را کنترل می کند. این متغیر شمارنده یا اندیس حلقه تکرار نام دارد. اندیس حلقه دارای مقدار اولیه و در هر بار اجرای دستورات حلقه، مقداری به آن اضافه می شود که گام حرکت حلقه نام دارد. گام حرکت حلقه می تواند عددی صحیح و اعشاری، مثبت و یا منفی و یا کاراکتری و یا حتی حاصل یک عبارت محاسباتی باشد. یکی دیگر از اجزای حلقه for، شرط حلقه است. شرط حلقه مشخص می کند که دستورات داخل حلقه تا کی باید اجرا شوند. اگر شرط دارای ارزش درستی باشد دستورات داخل حلقه اجرا می شوند و گرنه کنترل برنامه از حلقه خارج می شود. ساختار دستور for را در زیر می توانید مشاهده کنید.

```
for (گام حرکت ; شرط حلقه ; مقدار اولیه اندیس حلقه)
{
    دستور ۱;
    دستور ۲;
    ...
    دستور n;
}
```

در زمان استفاده از حلقه for به نکات زیر توجه کنید :

در هر بار اجرای دستورات، اندازه گام حرکت، به مقدار جاری شمارنده اضافه می شود. در این حالت، اگر مقدار اولیه شمارنده از مقدار نهایی بزرگتر باشد حلقه تکرار اصلاً اجرا نخواهد شد.

در هر بار اجرای دستورات به اندازه گام حرکت از مقدار جاری شمارنده کاسته می شود. در این حالت، اگر مقدار اولیه شمارنده از مقدار نهایی کوچکتر باشد حلقه تکرار اصلاً اجرا نخواهد شد.

مقدار شمارنده حلقه، در بدنه حلقه نباید توسط برنامه نویس تغییر داده شود.

اگر حلقه تکرار for به ; ختم شود، دستور بعد از آن فقط یک بار اجرا خواهد شد.

در صورتی که مقدار نهایی شمارنده، با یک عبارت محاسباتی یا متغیری تعیین شده است بدان معنی است که عدد ثابتی نیست و اگر این متغیر در بدنه حلقه تغییر کند در تعداد دفعات

تکرار، تاثیری نخواهد داشت زیرا مقدار نهایی شمارنده حلقه، فقط یکبار و آن هم هنگام شروع حلقه، خوانده می شود و در متغیر درونی دستور for ذخیره می گردد. پس از خارج شدن از حلقه تکرار، مقدار شمارنده، آخرین مقداری خواهد بود که در طول اجرای حلقه به خود اختصاص داده است و همواره بسته به صعودی یا نزولی بودن حلقه، مقدار شمارنده یک واحد بیشتر یا کمتر از مقدار نهایی است. در صورتی که حلقه for فاقد مقدار اولیه اندیس حلقه، شرط حلقه و گام حرکت باشد، حلقه تکرار بی نهایت را ایجاد می کند مانند ساختار زیر. برای خاتمه دادن به اجرای حلقه تکرار بی نهایت، باید کلید Ctrl + Break را از صفحه کلید فشار دهید.

```
for (;;)
{
    دستور ۱;
    دستور ۲;
    ...
    دستور n;
}
```

در صورتی که فقط یک دستور در حلقه وجود داشته باشد نیاز به استفاده از {} نیست.

حلقه تکرار while

هنگامی که تعداد دفعات تکرار مجموعه ای از دستورات، نامعین باشد نمی توان از حلقه for استفاده کرد و باید از حلقه های دیگر مانند while و do استفاده کرد. ساختار کلی این حلقه تکرار به صورت زیر است :

```
while (شرط)
{
    دستور ۱;
    دستور ۲;
    ...
    دستور n;
}
```

این حلقه تکرار تا زمانی که شرط مورد نظر درست باشد دستورات داخل حلقه، تکرار و اجرا

خواهند شد. اما به محض این که شرط نقض شد حلقه خاتمه یافته و دستورات بعد از آن اجرا می شوند. از حلقه while می توان به جای حلقه for استفاده کرد در این صورت شمارنده حلقه تکرار از سوی برنامه نویس تغییر می کند. در صورتی که تعداد دستورات یکی باشد نیازی به نوشتن {} نیست. توجه کنید که شرط حلقه بایستی در داخل حلقه تکرار نقض شود اگر همیشه شرط حلقه درست باشد و هیچگاه نقض نشود حلقه تکرار بی نهایت ایجاد می شود.

حلقه تکرار do

ساختار تکرار do مانند ساختار تکرار while است. با این تفاوت که در ساختار while شرط حلقه، در ابتدای حلقه بررسی می شود ولی در ساختار تکرار do شرط حلقه در انتهای حلقه بررسی می شود. بنابراین دستورات موجود در حلقه do، در هر حال حداقل یک بار اجرا می شوند.

```
do
{
    دستور ۱؛
    دستور ۲؛
    ...
    دستور n؛
}
while (شرط)؛
```

در این ساختار نیز در صورتی که تعداد دستورات تنها یک مورد باشد نیازی به نوشتن {} نیست. چنانچه شرط حلقه در داخل حلقه تکرار نقض نشود، این ساختار نیز حلقه تکرار بی نهایت را ایجاد خواهد کرد.

مفهوم آرایه

لیست (فهرست اسامی) دانش آموزان کلاس خود را در نظر بگیرید. اگر در برنامه ای نیاز باشد که نام خانوادگی تمام همکلاسی های خود را از وردی دریافت و ذخیره کنید به تعداد همکلاسی هایتان باید متغیر تعریف نمایید که انجام این عمل و کار کردن با این تعداد متغیر در یک برنامه، کار مشکلی است. در چنین مواردی باید از مفهوم لیست (آرایه) استفاده کرد.

در ریاضیات هم هر وقت کمیت های هم نام وجود داشته باشند آنها را به صورت یک لیست به کار می بریم. فهرست اسامی دانش آموزان یک کلاس را دوباره در نظر بگیرید و فرض کنید که می خواهیم برای هر دانش آموز ۱۰ نمره را دریافت و معدل نمرات او را حساب کنیم، در این صورت نیز تعداد متغیر ها زیاد است و برنامه نویسی و اشکال زدایی چنین برنامه ای، کار مشکل و به دور از منطق است.

همانطور که می دانید هر متغیر در هر لحظه می تواند یک مقدار داشته باشد و با انتساب مقدار جدید، مقدار قبلی از بین می رود. پس اگر تعداد متغیر ها در برنامه ای زیاد باشد می توان از آرایه ها استفاده کرد.

آرایه یک متغیر است که می تواند دارای مقادیر و عناصر هم نوع مختلفی باشد یعنی آرایه، مجموعه ای از عناصر هم نوع است که تحت یک نام در رایانه ذخیره می شوند. هر عنصر آرایه را یک (متغیر اندیس دار) می نامند. اندیس، یک عدد یا متغیر عددی است که به همراه نام آرایه ذکر می شود. به عنوان مثال، فهرستی از اعداد طبیعی از ۱۰ تا ۲۰ را در نظر بگیرید.

۱۰	۱۱	۱۲	۱۳	۱۴	۱۵	۱۶	۱۷	۱۸	۱۹	۲۰
$N(۱)$	$N(۲)$									$N(۱۱)$

اگر بخواهیم این مقادیر را در حافظه رایانه ذخیره کنیم، بهتر است از آرایه استفاده کنیم. نام این آرایه را N در نظر می گیریم. این آرایه، دارای ۱۱ عنصر است و هر عنصر با یک اندیس مشخص شده است که اندیس، موقعیت عنصر در آرایه را تعیین می کند. مثلاً عدد ۱۱ در خانه دوم آرایه قرار دارد و در متغیر اندیس $N(۲)$ ذخیره شده است.

در واقع زمانی از آرایه استفاده می شود که بخواهیم با مجموعه ای از اطلاعات همجنس کار کنیم. آرایه ها با توجه به تعداد اندیس هایشان به انواع مختلف تقسیم می شوند. آرایه ای یک اندیس را آرایه یک بعدی، آرایه دارای دو اندیس را آرایه دو بعدی و آرایه ای که n اندیس داشته باشد را آرایه n بعدی می نامند.

برای استفاده آرایه بایستی دو کار انجام شود :

۱- اعلان آرایه

۲- تخصیص فضا به آرایه

برای اعلان یک آرایه یک بعدی به صورت زیر عمل می شود :

؛ نام آرایه [] نوع آرایه

نام آرایه [] نوع آرایه

؟

برای تخصیص فضا به آرایه باید نمونه هایی از آن را ایجاد کنید. این کار با دستور new و به صورت زیر انجام می شود.

؛ [تعداد عناصر آرایه] نوع آرایه new = نام آرایه

همانطور که گفتیم آرایه متغیر اندیس داری است. بنابراین برای بازیابی و مقدار دهی به عناصر آن از اندیس به صورت زیر استفاده می شود.

؛ [اندیس آرایه] نام آرایه

برای تعریف آرایه چند بعدی باید به صورت زیر عمل کنید :

؛ [طول بعد 1 و ... و طول بعد 2 و طول بعد 1] نوع آرایه new = نام آرایه [....]

پیمایش عناصر آرایه

برای پیمایش عناصر آرایه می توان از انواع حلقه های تکرار استفاده کرد. ولی استفاده از حلقه تکرار foreach بهترین روش پیمایش آرایه ها و کلکسیون ها می باشد. این ساختار به صورت زیر به کار می رود :

(نام کلکسیون یا نام آرایه in متغیر نوع) foreach

```
{  
    دستور ۱  
    دستور ۲  
    ....  
    دستور n  
}
```

آشنایی با متدها

متد ها یکی از مهمترین مباحث برنامه نویسی است. فرض کنید که در برنامه شما یک بخش

بایستی عملیات خاصی را انجام دهد و این قسمت از کد باید چندین بار تکرار شود برای این منظور ما این چند خط کد را بصورت یک تابع بسته بندی می کنیم و هر جا که لازم بود از آن استفاده می کنیم. طرز تعریف متد ها در زبان برنامه نویسی C#.NET به صورت زیر است :

(نوع و اساس پارامتر های ورودی) نام تابع نوع تابع خروجی سطح دسترسی به تابع

```
{
    دستور۱
    دستور۲
    ....
    دستور n
}
```

بدنه تابع

استفاده از توضیحات موضوع دیگری که در ظاهر بسیار ساده به نظر می رسد افزودن توضیحات به کد است. زبان C#.NET از توضیحاتی مشابه با توضیحات یک خطی (//...) و چند خطی (/*...*/) زبان C استفاده می کند.

```
//This is a program for you
```

```
/* This comment
spans multiple lines */
```

هر چیزی که در یک توضیحات تک خطی از // تا پایان خط نوشته شود، توسط کامپایلر نادیده گرفته خواهد شد. و نیز هر چیزی که از /* تا */ نوشته شود در توضیحات چند خطی نادیده گرفته خواهد شد. مسلماً شما نمی توانید ترکیبی از /* را در توضیحات چند خطی به کار برید، زیرا این ترکیب به عنوان پایان توضیحات تلقی خواهد شد.

قرار دادن توضیحات چند خطی در میان یک خط از کد امکان پذیر است :

```
Console.WriteLine (/* Here's a comment! */ "This will compile");
```

این گونه توضیحات درون خطی باید با احتیاط به کار روند زیرا می توانند از خوانایی کد بکاهند. اگر چه، این توضیحات می توانند در هنگام اشکال زدایی مفید واقع شوند. مثلاً شما بعضی اوقات می خواهید به طور موقت به یک مقدار متفاوت اجرا شود :

```
DoSomething (Width, /*Height*/ 100);
```

البته کاراکتر های توضیحات که در رشته واقع شده اند، به عنوان کاراکتر های معمول با آنها

رفتار می شود.

```
string s = "/* This is a free program */" ;
```

مستندات XML

زبان C# علاوه بر توضیحات زبان C که در بخش پیش نشان داده شد، یک ویژگی بسیار خوب دارد که ما نمی توانیم در این فصل آن را فراموش کرده و توضیح ندهیم و آن هم توانایی تولید مستنداتی از توضیحات خاص به صورت خودکار در قالب XML است. این توضیحات تک خطی هستند اما به جای استفاده از دو خط مورب از سه خط مورب (///) استفاده می کنند. شما می توانید در این توضیحات، تگ های XML را که شامل مستندات انواع و اعضای آن هستند در کد خود قرار دهید.

سخن پایانی

تا به اینجا سعی کردیم تا مروری بر ساختار دستورات زبان برنامه نویسی C#.NET داشته باشیم. نگران نباشید که در این فصل به صورت عملی مثال هایی را در ارتباط با ساختار دستورات ذکر نکردیم ما در این فصل هدفمان این بود که خواننده را بصورت کلی با ساختار دستورات آشنا سازیم. در فصل های بعدی خواننده را بطور کامل و در قالب مثال های عملی و بصورت ویژوال با این دستورات آشنا خواهیم ساخت.

یادگیری این زبان برای افرادی که دارای سابقه آشنایی با یکی از زبان های برنامه نویسی C, C++ و یا Java باشند کار مشکلی نخواهد بود، حتی افرادی که دارای آشنایی اولیه با Java Script و یا دیگر زبان های برنامه نویسی مانند Visual Basic می باشند، امکان پذیر و راحت خواهد بود. برخی از برنامه نویسان حرفه ای بر این باور هستند که C#.NET نسبت به VB.NET با اقبال بیشتر و سریع تری مواجه خواهد شد، چرا که C# نسبت به Visual Basic خلاصه تر است. حتی برنامه های بزرگ و پیچیده ای که توسط C# نوشته می شود خواناتر، کوتاه و زیبا خواهند بود. برخی از ویژگی های ارائه شده در C# نظیر Unsigned Integer،

Operator Overloading و امنیت بیشتر Type ها، در VB.NET وجود نداشته و این امر می تواند دلیلی بر فراگیر شدن C#.NET نسبت به VB.NET نزد برنامه نویسان با تجربه باشد. برای یادگیری هر یک از زبان های حمایت شده در .NET، می بایست از BCL مربوط به .NET Framework شروع کرد. C# خود صرفاً دارای ۷۷ کلمه کلیدی یا Keyword بوده که برای اکثر برنامه نویسان غریب نخواهند بود. در مقابل BCL، دارای ۴۵۰۰ کلاس و تعداد زیادی متد و خاصیت است که برنامه نویسان C# می توانند از آنها برای انجام عملیات دلخواه خود استفاده کنند.

شاید یکی از مسائل قابل توجه جهت یادگیری این زبان برای برخی از برنامه نویسان حرفه ای عدم وجود برخی از ویژگی ها و امکاناتی باشد که در گذشته و از طریق سایر زبان های استفاده شده، به خدمت گرفته می شدند.

بدون شک یادگیری و تسلط بر زبان C# به منزله کسب یک پتانسیل با ارزش بوده که ثمرات آن برای برنامه نویسان در حال و آینده ای نه چندان دور بیشتر روشن خواهد شد. استاندارد بودن و وجود کتابخانه ای مملو از کلاس این اطمینان را به وجود خواهد آورد که با فراگیری این زبان و کسب مهارت های لازم، به یک توانایی فرا محیطی جدید دست پیدا خواهیم کرد که امکان استفاده از آن بر روی محیط های متفاوت وجود خواهد داشت. ویژگی ها و قابلیت های بی شمار این زبان از جمله دلایل قانع کننده دیگری است که فراگیری آن را توجیه پذیر و منطقی می کند.



فصل دوم

کار با کنترل های کاربردی در محیط Visual Studio

اجرای نرم افزار ۲۰۰۸ Microsoft Visual Studio

جهت اجرای نرم افزار ۲۰۰۸ Visual Studio از منوی Start گزینه All Programs و سپس بر روی پوشه ۲۰۰۸ Microsoft Visual Studio کلیک کنید و در داخل این پوشه بر روی برنامه ۲۰۰۸ Microsoft Visual Studio کلیک کنید، تا برنامه اجرا شود.

آشنایی با محیط نرم افزار ۲۰۰۸ Microsoft Visual Studio

Visual Studio ابزار قدرتمندی است برای ایجاد، مستند سازی و خطایابی برنامه هایی که در زبان های گوناگون .NET نوشته می شوند. وقتی Visual Studio را برای بار اول اجرا می کنیم پنجره ی Start Page باز خواهد شد، این پنجره شامل قسمت های زیر است : Recent Project : پروژه های اخیر را که باز کرده اید می توانید در این قسمت مشاهده کنید.

Getting Started : برای بدست آوردن آخرین مباحث آموزشی راجب Visual Studio می توانید از این بخش استفاده کنید.

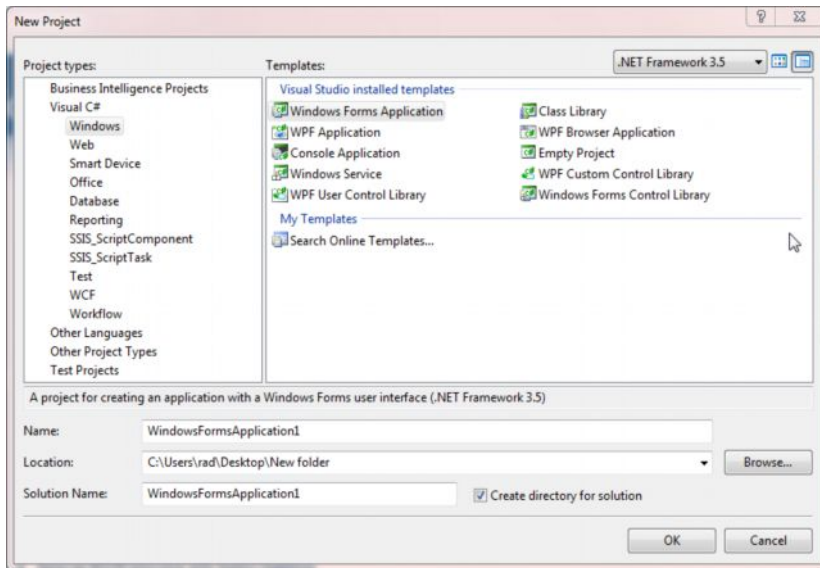
Visual Studio Headlines : برای بدست آوردن آخرین اخبار پیرامون Visual Studio از این بخش می توانید استفاده کنید.

MSDN : برای استفاده از راهنمای MSDN که به صورت بخش بندی شده ارائه می گردد از این بخش استفاده کنید.

MSDN مخفف Microsoft Developer Network می باشد، و در حکم راهنمای آموزشی نرم افزار Visual Studio است. این راهنمای آموزشی به صورت زبان فارسی ارائه گردد؛ و برای استفاده از آن باید در سایت کتابخانه MSDN عضو شوید. تا بتوانید از این فایل آموزشی که حجمی در حدود ۳ گیگابایت دارد استفاده نمایید.

ساخت اولین پروژه

جهت ساخت اولین پروژه خود تحت محیط Visual Studio در بخش Recent Project بر روی گزینه Create کلیک کنید، پنجره New Create باز می شود (پنجره تصویر ۱ - ۲).



تصویر ۱ - ۲

در پنجره New Project و در بخش Project types گزینه Visual C# را کلیک کنید تا گزینه های مربوط به آن باز شود. این بخش همانطور که در پنجره تصویر ۱ - ۲ ملاحظه می کنید دارای گزینه های متعددی است که زمینه های مختلف برنامه نویسی را برای شما در محیط های مختلف فراهم می کند. در بخش Templates هم می توانید انواع بستر های کاری را که برای زمینه کاری خود انتخاب کرده اید مشاهده کنید. در بخش Project type گزینه Windows و از بخش Templates گزینه Windows Forms Application را انتخاب کنید. پنجره New Project دارای بخش های دیگر است که به معرفی هر یک از آنها می پردازیم :

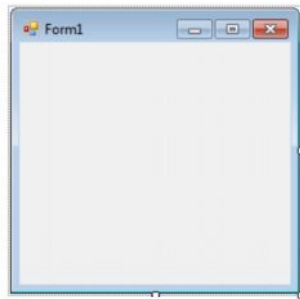
Name : در این بخش می توانیم عنوان پروژه خود را تعیین کنیم.

Location : در این بخش می توانیم محل ذخیره و قرارگیری پروژه خود را بر روی دیسک سخت تعیین کنیم.

Solution Name : می توانیم برای Solution پروژه خودمان عنوانی را تعیین کنیم. Solution وظیفه نگهدار بخش های مختلف پروژه را بر عهده دارد. این پنجره می تواند حاوی اطلاعات قسمت های مختلف یک یا چندین پروژه باشد. و با پسوند Sln ذخیره می شود.

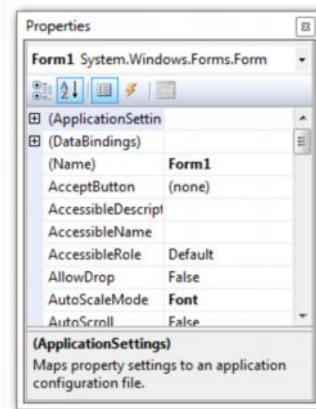
زمانی که با محیط Visual Studio کار کردید خیلی با این بخش که به صورت یک پنجره مجزا ارائه می شود برخورد خواهید داشت. انتخاب گزینه Create directory for solution موجب می شود که برای فایل های بخش Solution شما پوشه جداگانه ای ساخته شود و این اطلاعات در آنجا نگهدای و ذخیره گردند. از بخش Project type گزینه Windows و از بخش Templates گزینه Windows Forms Application را انتخاب کنید، در پایان بر روی دکمه OK کلیک کنید تا یک پروژه جدید از نوع Windows Application ساخته شود. پس از ساخت پروژه محیط Visual Studio به طور کامل نمایش داده می شود که دارای بخش های کاربردی زیادی است، که به معرفی برخی از بخش های مهم آن می پردازیم :

Form : این بخش در وسط پنجره برنامه قرار دارد، در واقع تمام کار ها در این بخش انجام می شود. در این بخش می توانید انواع طرح بندی و کد نویسی را انجام داده و تمام اشیای برنامه بر روی این ابزار قرار می گیرند. این بخش در واقع مانند بوم نقاش است که نقاش بر روی آن طرح و تفکر خود را قرار می دهد (پنجره تصویر ۲ - ۲).



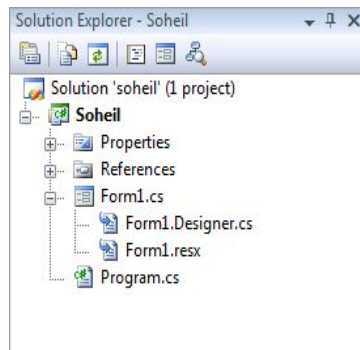
تصویر ۲ - ۲

Properties : این بخش حاوی انواع خصوصیات و رویداد هایی است، که انواع کنترل ها و Form هایی که در محیط Visual Studio قرار دارند می باشند. این خصوصیات و رویداد ها از طریق این پنجره قابل تنظیم است. در صورتی که این پنجره را مشاهده نمی کنید از منوی View گزینه Properties Windows را کلیک کنید، و یا بر روی آیکن این پنجره در نوار ابزار کلیک کنید و یا به طور همزمان دکمه های p , w + Ctrl را از روی صفحه کلید فشار دهید (پنجره تصویر ۳ - ۲).



تصویر ۳ - ۲

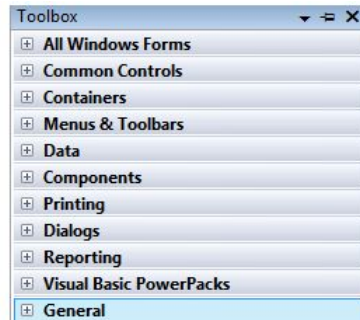
Solution Exploere : همانطور که در پنجره تصویر ۴ - ۲ مشاهده می کنید این پنجره وظیفه نگهداری قسمت های مختلف پروژه را بر عهده دارد.



تصویر ۴ - ۲

Toolbox : این بخش به نوعی جعبه ابزار نرم افزار Visual Studio محسوب می شود. تمامی کنترل های نرم افزار در این بخش قرار دارند. این بخش حاوی کنترلی های کاملاً کاربردی است که در اکثر برنامه های ویندوز با آنها کار کرده ایم. سعی می کنیم در این فصل با بیشتر این کنترل ها کار کنیم تا به صورت عملی نحوه کار این کنترل ها را درک کنید.

همانطور که در پنجره تصویر ۵ - ۲ مشاهده می کنید بخش Toolbox حاوی چندین قسمت می باشد که هر یک از این بخش ها دارای کنترل های خاص خود می باشند و برای کاربرد های خاصی در نظر گرفته شده اند، و بصورت منظم دسته بندی شده اند. در بخش All Windows Forms می توانید تمامی این کنترل ها را مشاهده کنید.



تصویر ۵ - ۲

معرفی Form و خواص آن

اگر که نقاش ها روی بوم نقاشی کار های خود را انجام می دهند، در برنامه های مبتنی بر ویندوز نیز برنامه نویسان تمام کار خود را روی این Form ها انجام می دهند. دقت کنید الان خود محیط Visual Studio مبتنی بر فرم است. کمی در مورد برنامه نویسی شی گرا صحبت کردیم الان این فرم دارای خواص و رفتار هایی است، که سعی می کنیم برخی از آنها را معرفی کنیم. Form را انتخاب کرده و بر روی پنجره Properties کلیک کنید تا لیست خواص مربوط به آن نمایش داده شود :

خاصیت Name : هر شی در محیط Visual Studio و کلا محیط های برنامه نویسی دارای نام است با استفاده از این خاصیت می توانید برای کنترل Form خودتان و یا هر کنترلی که دارید عنوانی را مشخص کنید.

خاصیت RightToLeft : وضعیت فرم را به حالت راست به چپ قرار می دهد که این حالت برای نرم افزار های فارسی مناسب است.

خاصیت ShowIcon : خاصیتی است که نوع آن Boolean است، می توان مقدار آن را روی True یا False تنظیم کرد. اگر روی حالت False قرار گیرد آیکن کنار آن حذف می شود.

خاصیت ShowInTaskbar : این خاصیت نیز دارای دو حالت False و True است. در صورتی که این خاصیت بر روی مقدار True تنظیم شود، Form در نوار وظیفه نمایش داده می شود.

خاصیت StartPosition : این خاصیت دارای چند زیر مجموعه است که به ترتیب زیر است :

Manual : در این حالت می توان به صورت دستی موقعیت قرار گیری Form را تعیین کرد.

CenterScreen : در این حالت Form در وسط صفحه نمایش قرار می گیرد. در این حالت حتی Resolution صفحه را نیز خود نرم افزار Visual Studio تنظیم می کند.

WindowsDefaultLocation : حالت پیش فرض قرارگیری Form است.

WindowsDefaultBound : محلی را که سیستم عامل ویندوز بطور پیش فرض تعیین می کند را مشخص می سازد.

CenterParent : حالتی است که Form دقیقاً در وسط صفحه نمایش قرار می گیرد.

TransparencyKey : با استفاده از این خاصیت می توان وضوح Form را تعیین کرد. با استفاده از این خاصیت و خاصیت Opacity می توان درجه وضوح Form را تعیین کرد. بطور مثال مقدار Opacity را برابر ۵۰٪ و خاصیت TransparencyKey را نیز روی حالت Transparent قرار دهید. همانطور که مشاهده می کنید میزان شفافیت رنگ Form کم شده و این مساله می تواند در زیبا سازی برنامه تجاری و غیره موثر واقع شود.

AutoScroll : این خاصیت دارای دو حالت True و False می باشد. در صورتی که این خاصیت بر روی گزینه True قرار گیرد، زمانی که روی Form از کنترل های زیادی استفاده کنید و اندازه Form نیز در حد مطلوب نباشد به صورت خودکار در حالت Scroll قرار می گیرد.

BackgroundImageLayout : مشخص می کند تصویری که برای زمینه انتخاب کرده اید در چه حالتی قرار بگیرد.

FormBorderStyle : این خاصیت دارای چندین گزینه است که هر کدام عملکرد مخصوص به خود را دارا هستند :

None : باعث می شود تا Form بدون حاشیه و بدون نوار عنوان نمایش داده شود.

FixedSingle : در صورتی که این گزینه انتخاب شده باشد نمی توان در زمان اجرا اندازه Form را تغییر داد.

Fixed3D : حاشیه Form به صورت سه بعدی نمایش داده می شود.

FixedDialog : حاشیه Form به صورت کادر محاوره ای ظاهر می شود. در این حالت Form فاقد ابزار کنترلی در عنوان Form خواهد بود.

Sizeable : حاشیه فرم طوری ظاهر می شود که بتوان اندازه آن را تغییر داد.

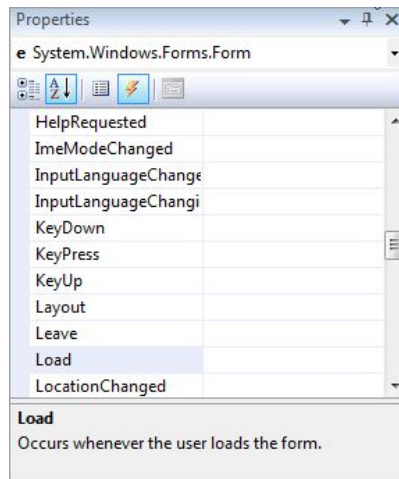
خاصیت Language : با استفاده از این خاصیت می توان زبان کارکرد Form و انواع کنترل هایی را که بر روی آن قرار می گیرند را تعیین کرد.

خاصیت TopMost : در صورتی که مقدار این خاصیت بر روی True تنظیم شود، Form بر روی همه پنجره های کاربردی در زمان اجرا نشان داده خواهد شد.

خاصیت WindowState : می توان اندازه Form را در اولین اجرا تعیین کرد. در صورتی که این خاصیت روی مقدار Normal قرار گیرد Form بطور معمولی نمایش داده می شود. اگر مقدار این خاصیت به Minimized تغییر داده شود Form به اندازه یک آیکن نمایش داده می شود. ولی اگر مقدار این خاصیت روی Maximized قرار گیرد Form در حالت بزرگ نمایش داده می شود.

تعریف رفتارهای Form

یک کلاس Form علاوه بر خاصیت دارای رفتار ها و رویداد هایی نیز هست. برای این منظور در پنجره Properties به قسمت Events کلیک کنید تا انواع رویداد هایی را که برای کنترل Form در نظر گرفته شده است را مشاهده کنید (پنجره تصویر ۶ - ۲).



تصویر ۶ - ۲

همانطور که در پنجره تصویر ۶ - ۲ مشاهده می کنید لیست انواع رویداد ها نمایش داده شده است. حال

به معرفی و کار با برخی از آنها می پردازیم :

رویداد Load : این رویداد لحظه ای است که Form در حال بارگذاری است. بطور مثال می خواهیم با استفاده از تابع MessageBox زمان Load شدن Form یک پیغام نمایش داده شود. روی این رویداد دابل کلیک کنید تا پنجره کد نویسی آن باز شود، سپس دستورات زیر را در آن بنویسید.

```
private void Form1_Load(object sender, EventArgs e)
{
    MessageBox.Show("Hello World! I am a Programmer");
}
```

زمانی که برنامه را اجرا کنید در اولین بار پیغام Hello World! I am a Programmer نمایش داده می شود.

نکته !

جهت اجرا برنامه می توانید از منوی Debug گزینه Start Debugging را اجرا کنید یا در نوار ابزار دکمه Play در نوار ابزار را کلیک کنید و یا دکمه F5 را از صفحه کلید کلیک کنید.

رویداد Click : این رویداد باعث می شود تا در زمانی که کاربر با ماوس بر روی Form کلیک کرد اتفاق مورد نظر برنامه نویس رخ دهد. بر روی این رویداد در پنجره Events کلیک کنید تا وارد بخش برنامه نویسی آن شوید و سپس دستورات زیر را بنویسید.

```
private void Form1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World! I am a Programmer");
}
```

کافی است که برنامه را اجرا کرده و نتیجه را مشاهده کنید. با استفاده از رویداد Click توانستیم کدی را بنویسم که کاربر در زمان اجرا، وقتی بر روی Form کلیک کرد پیغام Hello World! I am a Programmer نمایش داده می شود.

رویداد FormClosing و رویداد FormClosed : در رویداد FormClosing می توان دستورات مورد نظر را در حالی که Form در حال بسته شدن است را نوشت تا اجرا شود. در رویداد FormClosed می توان دستور مورد نظر را در حالی که Form بسته شده است و کار تمام شده است نوشت تا اجرا شود.

روداد `KeyDown` : این رویداد زمانی رخ می دهد که کلیدی از صفحه کلید فشار داده شود. بر روی این رویداد دابل کلیک کنید تا وارد بخش کد نویسی آن شوید.

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    MessageBox.Show("C# is the best language");
}
```

در صورتی که برنامه را اجرا کنید و کلیدی را از صفحه کلید فشار دهید پیغام `C# is the best language` ظاهر می شود. حالا در صورتی که تمایل داشته باشید بدانید چه کلیدی از صفحه کلید فشار داده می شود بهتر است از رویداد `KeyDown` به صورت زیر استفاده کنید.

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    MessageBox.Show("C# is the best language");
    MessageBox.Show(e.KeyCode.ToString());
}
```

کافیست که برنامه را اجرا کرده و نتیجه را مشاهده کنید.

نکته !

رویداد `KeyDown` در `Hot Key` گذاری (کلید تعریف شده برای اجرای سریع دستورات) بسیار مهم است. نرم افزار هایی که `Hot Key` دارند مانند نرم افزار `Visual Studio` با استفاده از رویداد های `KeyDown` و `KeyCode` براحتی می توان برای برنامه ها `Hot Key` تعریف کرد.

رویداد `KeyUp` : مانند رویداد `KeyDown` است ولی عکس آن عمل می کند. در این رویداد زمانی که کلید بالا می آید براحتی دستورات بالا را می توان در رویداد `KeyUp` تایپ کرده و استفاده برد.

رویداد `KeyPress` : این رویداد کاری بالا و پایین آمدن کلید ها ندارد، زمانی که کاربر کلیدی را فشار داد این رویداد اجرا می شود.

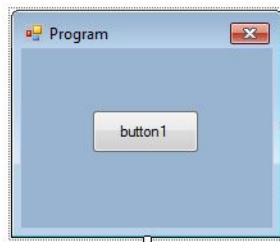
رویداد `DoubleClick` : این رویداد زمانی رخ می دهد که کاربر روی `Form` دوبار کلیک کند.
رویداد `LayOut` : این رویداد زمانی رخ می دهد که کاربر `Form` یا کنترل را ترک کند، یعنی اشاره گر ماوس از روی آن خارج می شود.

رویداد Paint : این رویداد شبیه رویداد Load است، با این تفاوت که اگر Form را بزرگ نمایی و یا کوچک نمایی کنی این رویداد رخ خواهد داد.

تا به اینجا با برخی از رویداد ها و خصوصیات Form ها آشنا شدید. بعضی از این رویداد ها و خواص در اکثر کنترل ها مشترک است، در صورتی که نگران این هستید که تمام این رویداد ها و خصوصیات را یاد نگرفتید، نگران نباشید ما سعی کرده ایم به صورت کاملاً قدم به قدم در تمام بخش ها آنها را توضیح دهیم. سعی کرده ایم تا چیزی را از قلم نیندازیم پس حوصله داشته باشید و با اطمینان ادامه مباحث را دنبال کنید.

آشنایی با کنترل Button

Button یا دکمه، ابزار صدور فرمان توسط استفاده کننده از نرم افزار هاست. خود Form هایی که به صورت روزمره در محیط ویندوز وجود دارند از دکمه برای صدور دستورات در اختیار کاربران قرار داده می شوند. برای دسترسی به این کنترل از Toolbox در Common Controls کنترل Button را به Form خودتان اضافه کنید.



تصویر ۷ - ۲

در حالتی که کنترل Button به صورت انتخاب شده می باشد، روی Properties کلیک کنید تا پنجره خصوصیات آن باز شود. حالا به معرفی برخی از خصوصیات آن می پردازیم :

خاصیت Text : یکی از مهمترین خاصیت های این کنترل و اکثر کنترل های نرم افزار Visual Studio می باشد. این خاصیت در واقع راهنمای کاربر است و به او در انجام عملیات کمک می کند. ما خاصیت Text کنترل Button خود را Calc قرار می دهیم.

خاصیت TextAlign : برای تنظیم موقعیت قرار گرفتن متن درون کنترل Button استفاده می شود.

خاصیت `AutoSize` : با استفاده از این خاصیت می توان در صورتی که روی حالت `True` تنظیم شود، اندازه کنترل را با توجه به خاصیت `Text` تنظیم کرد.

خاصیت `BackgroundImageLayout` : در صورتی که این گزینه روی حالت `Zoom` یا `Stretch` قرار گیرد تصویر هم اندازه دکمه خواهد شد. اما حالت `Center` برای قرار گیری تصویر در وسط کنترل استفاده می شود.

خاصیت `Dock` : این خاصیت در همه کنترل ها وجود دارد یعنی طوری در صفحه قرار می گیرد که اگر کنترل دیگری هم قرار بگیرد بعد از این کنترل قرار خواهد گرفت.

خاصیت `FlatAppearance` : اگر خاصیت `FlatStyle` برابر با `False` باشد نمای حاشیه و رنگ های مورد استفاده برای حالت ماوس و حالت کنترل را مشخص می کند.

کنترل `Button` دارای یک رویداد بسیار مهم به نام `Click` است در روی `Form` بر روی کنترل `button1` دابل کلیک کنید تا رویداد `Click` آن باز شود.

```
private void button1_Click(object sender, EventArgs e)
{
}
```

حالا می خواهیم در رویداد `Click` این کنترل کدی را بنویسیم تا زمانی که کاربر روی دکمه کلیک کرد رنگ زمینه و عنوان `Form` تغییر یابد.

```
private void button1_Click(object sender, EventArgs e)
{
    this.BackColor = Color.Red;
    this.Text = "C#.NET FOR ALL";
}
```

با استفاده از کد نویسی می توان کار های جالبی را انجام داد. برنامه نویسی مبتنی بر `Design` را فراموش کنید همه چیز را با کد نویسی انجام دهید تا مدیریت شما در نرم افزار بیشتر شود.

آشنایی با کنترل `TextBox`

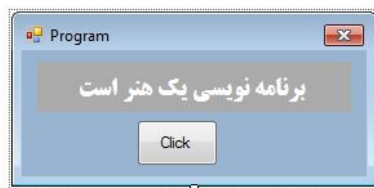
کنترل `TextBox` یا جعبه متن یکی از کنترل مهم در برنامه نویسی سطح بالا می باشد. وظیفه آن دریافت اطلاعات از کاربران است، در حقیقت مانند درگاهی برای گرفتن اطلاعات از کاربران است. در برنامه های مبتنی بر ویندوز، وب، موبایل و حتی در نرم افزار هایی که برای خود پرداز

های بانک ها استفاده می شود با این کنترل مواجه شده و کار کرده اید. از بخش Common Control در Toolbox کنترل TextBox را به Form خودتان اضافه کنید. حالا این کنترل را انتخاب کرده و به پنجره Properties آن توجه کنید می خواهیم برخی از خصوصیات آن را بررسی کنیم :

خاصیت Text : یکی از خواص مهم این کنترل خاصیت Text آن است. هر عنوانی را که در این خاصیت قرار دهید، در بخش Text جعبه متن نمایش داده خواهد شد. می خواهیم با کمک این خاصیت دستوری را بنویسیم تا متنی را که درون کنترل TextBox قرار دارد خوانده و به صورت پیغام نمایش دهیم. به این منظور یک کنترل button بر روی Form قرار داده و خاصیت Text آن را برابر Click قرار دهید، سپس روی آن دو بار دابل کلیک کنید تا وارد قسمت کد نویسی آن شوید، آنگاه دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(textBox1.Text);
}
```

دستورات بالا را تایپ کرده و برنامه را اجرا کنید. در TextBox در زمان اجرا جمله دلخواه را تایپ می کنیم. ما جمله (برنامه نویسی یک هنر است) را تایپ کرده ایم، حالا روی دکمه Click کلیک کنید تا نتیجه را مشاهده کنید.



تصویر ۸ - ۲

خاصیت ReadOnly : گاهی اوقات شما نیاز دارید اجازه ندهید تا کسی درون جعبه متن تان چیزی را تایپ کند ولی درون جعبه متن، متنی وجود داشته باشد. به طور مثال هشدار یا پیغامی را به صورت ثابت بیان کند و به کاربران نشان دهد. در صورتی که مقدار خاصیت ReadOnly را برابر True قرار دهید در زمان اجرا هیچ کس نمی تواند درون آن چیزی تایپ کند و یا نوشته ای را که ما برای آن تعیین کرده ایم تغییر دهد. حال شاید این سوال به ذهنتان خطور کند که

چرا جعبه متن من به صورت خاموش درآمد، خیلی ساده با استفاده از خاصیت BackColor می توانید رنگ زمینه جعبه متن خودتان را تغییر دهید.

خاصیت PasswordChar: در صورتی که نیاز داشته باشید تا از جعبه متن برای موارد خاص مانند دریافت رمز عبور از کاربر استفاده کنید، یا اصلاً در برنامه قسمت هایی را دارید که می خواهید به صورت کاراکترهای خاص ذخیره شوند تا متن تاپی کاربر معلوم نباشد می توانید از این خاصیت استفاده کنید. در صورتی که مقدار این خاصیت را برابر * قرار دهید در زمان تایپ در جعبه متن علامت * تایپ خواهد شد.

خاصیت MultiLine: اگر از جعبه متن به صورت تک خطی استفاده شود کاربر مجاز است تا ۲۰۴۸ کاراکتر را درون آن تایپ کند. وقتی که مقدار خاصیت MultiLine را برابر مقدار True قرار دهید جعبه متن به یک جعبه متن چند خطی تبدیل خواهد شد، کاربر در این حالت می تواند تا ۳۲ کیلوبایت اطلاعات متنی را درون آن جای دهد.

خاصیت ScrollBars: با استفاده از این خاصیت می توان تعیین کرد آیا کنترل دارای نوار جابجایی باشد یا خیر. این خاصیت دارای مقادیر مختلفی است که به شرح کوتاهی راجب آن ها می پردازیم:

مقدار None: کنترل هیچ نوار جابجایی نباید داشته باشد.

مقدار Horizontal: کنترل فقط نوار جابجایی افقی دارد.

مقدار Vertical: کنترل فقط نوار جابجایی عمودی دارد.

مقدار Both: کنترل نوارهای جابجایی عمودی و افقی دارد.

در صورتی که مقدار خاصیت MultiLine برابر True باشد امکان انجام این تغییرات وجود دارد.

خاصیت MaxLength: حداکثر طول متنی را که کاربر می تواند تایپ کند را تعیین می کند بطور مثال در صورتی که بخواهید کاربر ۵ کاراکتر را وارد کند بهتر است مقدار این خاصیت را برابر عدد ۵ قرار دهید البته مقدار پیش فرض این خاصیت برابر ۳۲۷۶۷ است.

خاصیت Accepts: این خاصیت تعیین می کند که آیا در زمان اجرا مواقعی که کاربر کلید Enter را فشار داد مکان نما یک خط به پایین برود یا خیر؟

خاصیت AcceptsTab : این خاصیت تعیین می کند در صورتی که کاربر کلید Tab را فشار داد، آیا مکان نما به Tab بعدی منتقل شود یا به کنترل بعدی. در صورتی که مقدار این خاصیت روی True تنظیم شود با فشار دادن کلید Tab مکان نما به محل Tab بعدی می رود. خاصیت CharacterCasing : روش دریافت کاراکتر ها را تعیین می کند. این خاصیت دارای چند مقدار به صورت زیر می باشد :

مقدار Normal : برای دریافت کاراکتر ها بصورت کوچک و بزرگ استفاده می شود.

مقدار Upper : برای دریافت حروف بزرگ بکار می رود.

مقدار Lower : برای دریافت حروف کوچک بکار می رود.

کنترل جعبه متن نیز مانند سایر کنترل دارای رویداد های کاربردی فراوانی است ولی مهمترین رویدادی که معمولاً کاربران در برنامه نویسی های بانک اطلاعات با آن روبرو می شوند رویداد TextChanged می باشد این رویداد زمانی رخ می دهد که متن درون جعبه متن تغییر داده شود.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    Form1.ActiveForm.Text = textBox1.Text;
}
```

دستور بالا موجب می شود تا در زمان اجرا آنچه را که به عنوان متن تایی در جعبه متن وارد می کنید در نوار عنوان Form فعال نمایش داده شود.

کنترل textBox دارای متد های فراوانی نیز است که به معرفی برخی از آنها می پردازیم :

متد Clear : محتویات داخل جعبه متن را پاک می کند.

متد Copy : متن داخل جعبه متن را در حافظه موقت کپی می کند.

متد Paste : متنی را که در حافظه موقت است در مکان فعلی کنترل جعبه متن می چسباند. دستور زیر محتویات textBox1 را در textBox2 کپی می کند.

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Copy();
    textBox2.Paste();
}
```

متد SelectAll : محتویات کنترل جعبه متن را انتخاب می کند.

متد AppendText : متنی را به محتویات کنترل جعبه متن اضافه می کند. بطور مثال دستور زیر محتویات کنترل textBox2 را به انتهای کنترل textBox1 اضافه می کند.

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.AppendText(textBox2.Text);
}
```

متد Select : برای انتخاب محتویات داخل کنترل جعبه متن بکار می رود. به طور مثال دستور زیر، محتویات textBox1 را از سومین کاراکتر به طول دو کاراکتر انتخاب می کند.

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Select(3, 2);
}
```

متد Undo : عملکرد آخرین فرمان انجام شده بر روی کنترل جعبه متن را لغو می کند.

بررسی برخی از خواص عمومی کنترل ها

یک Form بدون کنترل های آن هیچ ارزشی ندارد. تمامی کنترل ها دارای خواص و رویداد های مختص خود هستند. البته برخی از متد و خواص بین تمامی این ابزار ها به صورت مشترک می باشد. قصد داریم به معرفی برخی از خواصی که در پنجره Properties در اکثر کنترل ها یکسان پردازیم :

خاصیت AllowDrop : تعیین می کند که آیا کنترل شما قابلیت پذیرش داده را از طریق Drag&Drop به کاربر دهد یا خیر؟

خاصیت Anchor : این خاصیت لبه یک object را نسبت به لبه های Form تعیین می کند. برای مثال من دوست دارم کنترل button من طوری رفتار کند که با کوچک و بزرگ شدن سطح Form همیشه اندازه آن نسبت به گوشه سمت راست و پایین فرم ثابت بماند. با استفاده از حالت هایی که این خاصیت ارائه می دهد می توانید تنظیمات دلخواه خود را اعمال کنید.

خاصیت ContextMenuStrip : با استفاده از این خاصیت می توانید کنترل میانبر مورد نظر را برای نمایش در زمان کلیک راست مشخص کنید.

خاصیت Locked : کنترل به صورت فعال نمایش داده می شود ولی کاربر حق استفاده از آن را ندارد.

خاصیت TabStop : می تواند جلوی فوکوس را بگیرد زمانی که کاربر از کلید Tab استفاده می کند.

برخی خاصیت ها مانند BackColor و یا Font و ... نیز همانطور که از نامشان پیداست برای کاربران آشنا و آسان می باشد.

بررسی برخی از متد های عمومی کنترل ها

متد BringToFront : این امکان را در اختیار شما قرار می دهد که کنترل جاری را روی بقیه کنترل های موجود در سطح Form خودتان قرار دهید.

متد Contains : تعیین می کند که آیا کنترل جاری، کنترل دیگری را در خود قرار دهد یا خیر؟ اصولاً از این متد برای کنترل هایی که نقش Container را دارند استفاده می شود.

متد CreateControl : این متد این امکان را می دهد که درون کنترل جاری کنترل دیگری را به عنوان کنترل فرزند بوجود بیاوریم.

متد CreateGraphics : به شما امکان کار با System.Drawing.Graphics را می دهد.

متد DrawToBitmap : این متد امکان ترسیم یک تصویر Bitmap را در اختیار برنامه قرار می دهد.

آشنایی با کنترل Label

یکی از کنترل های مفید در نرم افزار Visual Studio کنترل Label است. این کنترل در تمام نسخه های نرم افزار Visual Studio ارائه شده است. این کنترل برای نمایش متن هایی بکار می رود که اصلاً تغییر نمی کنند. برای راهنمایی یا چاپ پیامی خاص مناسب است و در اکثر پروژه و یا کلا در تمام پروژه ها از این کنترل استفاده می شود. در حالی که این کنترل روی Form در حالت انتخاب است بر روی پنجره Properties کلیک کنید تا لیست خصوصیات آن ظاهر شود. سعی می کنیم تا برخی از خصوصیات مهم این کنترل را مورد بررسی قرار دهیم :

خاصیت Image : تصویری را تعیین می کند که باید در زمینه این کنترل نمایش داده شود.

خاصیت ImageList : لیست تصاویری را مشخص می کند که می تواند در خاصیت Image قرار گیرند.

خاصیت `ImageIndex`: شماره تصویری را از خاصیت `ImageList` تعیین می کند که باید در خاصیت `Image` قرار گیرد.

خاصیت `UseMnemonic`: تعیین می کند آیا می توان در متن کنترل از کاراکتر `&` استفاده کرد یا خیر. به طور پیش فرض می توان از کاراکتر `&` در متن کنترل استفاده کرد. اگر قبل از کاراکتری، علامت `&` قرار گیرد زیر آن کاراکتر خط کشیده می شود. در حقیقت این کاراکتر به عنوان یک کاراکتر کلید دسترسی معرفی می شود.

خاصیت `AutoEllipsis`: امکان اداره کردن خودکار متنی را فراهم می کند که از عرض کنترل `Label` بیشتر است.

دستور `MessageBox`

تا به اینجا در برخی از مواقع از دستور `MessageBox` استفاده کردیم و تا حدودی آشنا هستید. بعضی از مواقع لازم است که شما پیغام کوتاهی را برای کاربر ارسال کنید و یا سوالی را از کاربر پرسید. در پاسخ به سوال لازم است که کاربر جواب `Yes` یا `No` یا `OK` و غیره را ارسال کند.

حال ببینیم به چه نحوی می توانیم از تابع `MessageBox` در زبان برنامه نویسی `C#` استفاده کنیم. از دستور `MessageBox` به ۲۱ شکل متفاوت می توانیم استفاده کنیم. ساده ترین شکل استفاده از این تابع به صورت زیر می باشد:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Iran");
}
```

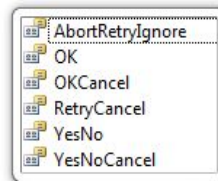
شما می توانید یک بخش دیگر نیز به تابع `MessageBox` اضافه کنید، مورد اول تعیین عنوان پنجره، و دیگری تعیین متن داخلی.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Ardebil", "IRAN");
}
```

در دستورات بالا، عنوان پنجره `IRAN` و متن داخلی آن برابر `Hello Ardebil` است. حالت دیگری که می تواند برای شما امکانات کاملتری را فراهم نماید این است که بتوانید نوشته، عنوان و نوع

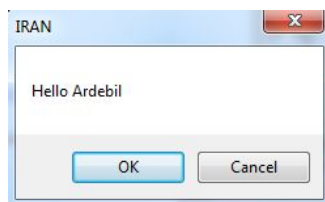
دکمه های کادر را مشخص کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Ardebil", "IRAN", MessageBoxButtons.
```



```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Ardebil", "IRAN", MessageBoxButtons.OKCancel);
}
```

حال اگر برنامه را اجرا کنید خروجی مشابه پنجره تصویر ۹ - ۲ خواهید داشت.

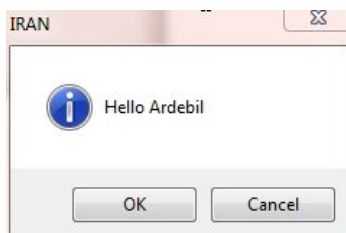


تصویر ۹ - ۲

حالت بعدی که می توانید از تابع MessageBox به صورت کامل و جامع تر استفاده کنید، این است که می توانید برای تابع MessageBox خودتان آیکن نیز تعریف نمایید. دستورات زیر را در نظر بگیرید :

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Ardebil", "IRAN", MessageBoxButtons.OKCancel,
        MessageBoxIcon.Information);
}
```

حال کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.



تصویر ۱۰ - ۲

همانطور که در پنجره تصویر ۱۰ - ۲ ملاحظه می کنید، یک آیکن برای متن مان ظاهر گردیده است. حتی شما می توانید بعد از استفاده از تابع MessageBox وضعیت دکمه کلیک شده توسط کاربر را در برنامه خودتان به سادگی کنترل کنید. حالت دیگری که می توانید از آن استفاده کنید حالتی است که به شما امکان استفاده از دکمه پیش فرض را می دهد. به همین ترتیب می توانید روی تک تک حالت های ممکن کار کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result1 = MessageBox.Show("Is Dot Net Perls awesome?",
    "Important Question",
    MessageBoxButtons.YesNo);
    //
    // Dialog box with question icon. [4]
    //
    DialogResult result2 = MessageBox.Show("Is Dot Net Perls awesome?",
    "Important Query",
    MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question);
    //
    // Dialog box with question icon and default button. [5]
    //
    DialogResult result3 = MessageBox.Show("Is Visual Basic awesome?",
    "The Question",
    MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2);
    //
    // Test the results of the previous three dialogs. [6]
    //
    if (result1 == DialogResult.Yes &&
    result2 == DialogResult.Yes &&
    result3 == DialogResult.No)
    {
        MessageBox.Show("You answered yes, yes and no.");
    }
}
```

مثال ۱ - ۲: برنامه ای بنویسید که دو عدد را از کاربر دریافت کرده و چهار عمل اصلی را بر روی آن انجام دهد؟

حل : Form ای را مطابق پنجره تصویر ۱۱ - ۲ طراحی کنید.



تصویر ۱۱ - ۲

حالا بر روی دکمه + دابل کلیک کنید تا وارد قسمت کد نویسی آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void ADD_Click(object sender, EventArgs e)
{
    int a,b,c;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = a + b;
    textBox3.Text = c.ToString();
}
```

حالا بر روی دکمه * دابل کلیک کنید تا وارد قسمت کد نویسی آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void MUL_Click(object sender, EventArgs e)
{
    int a,b,c;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = a * b;
    textBox3.Text = c.ToString();
}
```

حالا بر روی دکمه - دابل کلیک کنید تا وارد قسمت کد نویسی آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void SUB_Click(object sender, EventArgs e)
{
    int a,b,c;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = a - b;
    textBox3.Text = c.ToString();
}
```

حالا بر روی دکمه / دابل کلیک کنید تا وارد قسمت کد نویسی آن شوید، سپس دستورات زیر ا بنویسید.

```
private void DIV_Click(object sender, EventArgs e)
{
    int a,b,c;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = a / b;
    textBox3.Text = c.ToString();
}
```

همانطور که در کد های هر چهار قسمت مشاهده می کنید، در اولین خط سه متغیر به نام های a , b , c تعریف کرده ایم. در خط های بعدی مقدار رشته ای را که از دو کنترل $textBox1$ و $textBox2$ می خواند تبدیل به مقدار عددی کرده و در متغیر های a و b قرار می دهد. این کار با استفاده از دستور $Convert.ToInt32$ انجام می شود. در خط بعدی نیز همان طور که در هر چهار قسمت مشاهده می کنید عملیات های محاسباتی جمع، ضرب، تفریق و تقسیم نوشته شده است. در خط آخر نیز مقدار c که نتیجه عملیات ها در هر چهار قسمت است در آن ذخیره شده است. سپس با استفاده از تابع $ToString()$ به مقدار رشته ای تبدیل شده و مقدار خروجی در کنترل $textBox3$ نمایش داده می شود.

مثال ۲-۲: برنامه ای بنویسید که درجه حرارت شهری را برحسب فارنهایت دریافت کرده، سپس دمای آن شهر را به درجه سلسیوس نمایش دهد؟

رابطه تبدیل درجه فارنهایت به سانتیگراد: $C = 5 / 9 (F - 32)$

حل : Form ای را مطابق پنجره تصویر ۱۲ - ۲ طراحی کنید.

تصویر ۱۲ - ۲

حال بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، دستورات زیر را بنویسید.

```
private void Calc_Click(object sender, EventArgs e)
{
    float f, c;
    f = Convert.ToInt32(textBox1.Text);
    c = 5 / 9 * (f - 32);
    textBox3.Text = c.ToString();
}
```

حالا بر روی دکمه Close دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستور زیر را بنویسید.

```
private void Close_Click(object sender, EventArgs e)
{
    Close();
}
```

مثال ۳ - ۲: فرض کنید حقوق یک کارمند، S ریال تعیین شده است و ماهانه ۷ درصد بابت بیمه و ۴ درصد بابت وام مسکن از حقوق وی کسر می شود. برنامه ای بنویسید که حقوق کارمندی را دریافت و حقوق خالص پرداختی وی را محاسبه نماید.
حل : Form ای را مطابق پنجره تصویر ۱۳ - ۲ طراحی کنید.

تصویر ۱۳ - ۲

بر روی دکمه محاسبه دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void Calc_Click_1(object sender, EventArgs e)
{
    float s, b, m, p;
    s = Convert.ToInt32(textBox1.Text);
    b = s * 7 / 100;
    m = s * 4 / 100;
    p = s - b - m;
    textBox2.Text = p.ToString();
}
```

بر روی دکمه Close دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید. متد Close() موجب خروج کامل از برنامه می شود.

```
private void Close_Click(object sender, EventArgs e)
{
    Close();
}
```

دستوراتی که برای این برنامه نوشتیم با توجه به مطالبی که تا به الان بیان شده زیاد سخت نیست. بنابراین تجزیه و تحلیل این بخش از دستورات را بر عهده خواننده می گذاریم. در صورتی که یک کارمند ماهیانه ۴۰۰۰۰۰۰ ریال درآمد داشته باشد، با توجه به میزان کسر بیمه و وام، میزان دریافتی وی ۳۵۶۰۰۰۰ خواهد بود.

آشنایی با کنترل RichTextBox

کنترل دیگری که برای کار با متون وجود دارد کنترل RichTextBox است. این کنترل مانند یک جعبه متن چند خطی با قابلیت های منحصر بفرد است. شما در جعبه متن نمی توانید به صورت دلخواه بخش های مختلف یک متن را با رنگ ها و حالت های مختلف به نمایش در بیاورید. ولی در RichTextBox این امکان وجود دارد که به سادگی بتوانید متن های دلخواه خود را با Format و قالب های دلخواه به نمایش در بیاورید. حال این کنترل را بر روی Form خود اضافه کنید. در حالی که این کنترل در حالت انتخاب است پنجره Properties را کلیک کنید تا باز شود، در این پنجره می توانید لیست خصوصیات آن را مشاهده کنید. قصد داریم برخی از خصوصیات این کنترل را معرفی کنیم :

خاصیت RightMargin : این خاصیت این امکان را به شما می دهد که بتوانید فاصله از سمت راست را نسبت به حاشیه کنترل کنید.

خاصیت ZoomFactor : میزان فاکتور بزرگ نمایی نوشته های موجود درون کنترل را مشخص کنید.

خصوصیات دیگری که این کنترل دارد که در پنجره Properties قابل مشاهده نیست و بایستی آنها را در زمان کد نویسی استفاده کرد. که برخی از آنها عبارتند از :

`richTextBox.SelectRtf` : می تواند ساختار متن انتخاب شده و قالب متن انتخاب شده را برگرداند و یا مشخص کند.

`richTextBox.SelectedText` : می تواند متن موجود در بخش انتخاب شده درون کنترل را برگرداند و یا تغییر دهد.

`richTextBox.SelectionBackColor` : امکان انتخاب یا دریافت رنگ ناحیه انتخابی درون متن موجود را فراهم می کند.

`richTextBox.SelectionColor` : رنگ نوشته شده در بخش انتخابی را بر می گرداند.

`richTextBox.SelectionFont` : نوع فونت را در بخش انتخابی تعیین می کند و نیز امکان قالب بندی قسمت های مختلف متن را برای شما فراهم می آورد.

این کنترل نیز رویداد های مختلفی و پرکاربردی را دارد ولی یکی از پرکاربردترین آنها رویداد `LinkClicked` می باشد. با استفاده از این رویداد می توانید به تمام بخش ها و متون مانند `Hyperlink` ها دسترسی داشته باشید و با برنامه نویسی بر روی آنها اعمال مختلفی را انجام دهید.

این کنترل نیز همانند سایر کنترل ها دارای متد های کاربردی که مهمترین آنها عبارتند از :
`LoadFile` : برای ارسال اطلاعات از یک فایل به کنترل `RichTextBox` استفاده می شود.
و به صورت زیر به کار می رود :

```
richTextBox1.LoadFile(string path);
```

`path` رشته ای است که نام فایل و مسیری را تعیین می کند که باید اطلاعات از آن خوانده شود.

متد `SaveFile` : اطلاعات موجود در کنترل را در فایلی ذخیره می کند. و به صورت زیر به کار می رود :

```
richTextBox1.SaveFile(string path);
```

می خواهیم با متد `SaveFile` برنامه ای بنویسیم که متون موجود در کنترل `richTextBox1` را به صورت یک فایل متنی با پسوند `rtf` در دایرِی C ذخیره کند. یک `Form` مانند پنجره تصویر

۱۴ - طراحی کنید، سپس کنترل RichTextBox را از Toolbox و بخش Common Control به Form اضافه کنید.



تصویر ۱۴ - ۲

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن تایپ کنید.

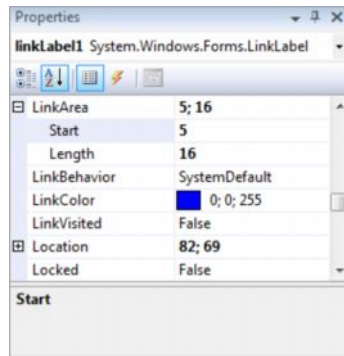
```
private void Save_Click(object sender, EventArgs e)
{
    richTextBox1.SaveFile("E:\\Text.rtf");
}
```

حالا کافی است که برنامه را اجرا کرده و متن دلخواه خود را در کنترل richTextBox1 تایپ کنید. سپس دکمه Click را کلیک کنید.

کنترل LinkLabel

این کنترل می تواند مشابه یک کنترل Label عمل کند و نوشته یا عنوانی را برای کاربران نشان دهد. ولی با این تفاوت که می تواند مشابه پیوند های Hyper Link های اینترنتی، پیوندی را با بخش های مختلف برنامه و صفحات وب برقرار کند. این کنترل دقیقا خواصی مشابه Hyper Link های اینترنتی دارد، یعنی رنگ آن آبی است و زمانی که ماوس روی آن قرار می گیرد به رنگ قرمز تغییر می یابد و زمانی که یک بار از آن استفاده شود به رنگ بنفش تبدیل می شود. برای اضافه کردن آن از Toolbox و از بخش Common Controls کنترل LinkLabel را می توانید به Form خود اضافه کنید. می خواهیم از این کنترل برای ارسال اطلاعات به یک پست الکترونیکی استفاده کنیم. خاصیت Text این کنترل را به Send Information تغییر می دهیم. یکی از خواص جالبی که این کنترل دارد این است که می توانید محدوده لینک مورد نظر در Label جاری را مشخص کنید، خاصیت LinkArea این

امکان را برای شما فراهم می آورد. در پنجره Properties این خاصیت را پیدا کرده و علامت + را کلیک نمایید و در گزینه Start عدد ۵ را وارد کنید.



تصویر ۱۵ - ۲

الان فقط کلمه Information به صورت Hyper Link نشان داده می شود. رویداد مهم این کنترل رویداد Link است این کنترل یک خصوصیت جالب به نام Visit دارد که وضعیت مشاهده یک لینک را مشخص می کند. در صورتی که خاصیت LinkVisited روی مقدار False قرار داشته باشد یعنی تا به حال کلیک روی آن انجام نشده است. و زمانی که روی حالت True قرار دارد یعنی یک بار لینک کلیک شده است. وقتی که در کنترل LinkLabel رویداد LinkClicked رخ می دهد، لازم است که خاصیت LinkVisted برابر با مقدار زیر باشد.

```
private void Click_Click(object sender, EventArgs e)
{
    linkLabel1.LinkVisited = true;
    System.Diagnostics.Process.Start("www.Gmail.com");
}
```

با این دستورات سایت Gmail برای ما باز می شود.

کنترل CheckBox

بطور حتم این کنترل را بار ها در محیط ویندوز و در برنامه های مختلف تجربه کرده اید. از کنترل CheckBox برای گرفتن تائید پارامتر از کاربر و مشخص کردن چندین گزینه مختلف توسط کاربر استفاده می شود. این کنترل امکان انتخاب دو حالت فعال و غیر فعال را به کاربر می دهد. در صورتی که کاربر بخواهد این کنترل را فعال کند باید روی این کنترل کلیک کند

تا علامت $\sqrt{}$ ظاهر شود و با کلیک دوباره آن را از حالت انتخاب خارج کند. از این کنترل می توانید به هر تعداد که دوست داشته باشید روی Form خودتان قرار دهید، این کنترل نیز مثل سایر کنترل ها دارای خواصی است که با کلیک روی پنجره Properties می توانید آنها را مشاهده کنید :

خاصیت Text : برای تعیین عنوان کنترل CheckBox استفاده می شود و برای راهنمایی کاربر به کار می رود.

خاصیت Checked : این خاصیت بررسی می کند که آیا کنترل در حالت انتخاب است یا خیر؟ اگر در حالت True باشد کنترل به صورت انتخاب شده در می آید.

خاصیت CheckState : این خاصیت مشابه خاصیت Checked است. اگر این خاصیت برابر مقدار Unchecked باشد کنترل CheckBox غیر فعال است ولی اگر مقدار این خاصیت برابر Checked باشد کنترل در حالت فعال است. و در صورتی که این خاصیت برابر مقدار Indeterminate باشد علامت تیک به صورت رنگ خاکستری در کنترل CheckBox نمایش داده می شود، یعنی مقدار CheckBox نامعلوم است. برای درک این خاصیت به مثال زیر توجه کنید :

یک پروژه جدید بسازید و در پنجره Form یک کنترل TextBox قرار دهید و خاصیت MultiLine آن را برابر مقدار True قرار دهید، و توضیحاتی را مطابق آنچه که در پنجره تصویر ۱۵ - ۲ مشاهده می کنید تایپ کنید. خاصیت ReadOnly این کنترل را برابر مقدار True قرار دهید.

قصد داریم یکی از این توضیحاتی که در برخی از صفحات اینترنتی و برنامه های کاربردی وجود دارد ایجاد کنیم. یک کنترل CheckBox روی Form قرار داده و در خاصیت Text این کنترل عبارت (من با این شرایط موافق هستم) را می نویسیم. یک کنترل Button نیز روی Form قرار داده و خاصیت Text این کنترل را نیز برابر عبارت (ادامه) تعیین می کنیم. خاصیت Enable آن را نیز برابر با مقدار False قرار می دهیم.



تصویر ۱۶ - ۲

همان طور که در پنجره تصویر ۱۵ - ۲ ملاحظه می کنید، تمام مراحل طراحی Form مانند پنجره تصویر ۱۵ - ۲ خواهد بود.

خاصیت ThreeState: تعیین می کند آیا CheckBox می تواند سه وضعیت تعیین شده در خاصیت CheckedState را قبول کند یا خیر.

یکی از مهمترین رویداد های کنترل CheckBox رویداد CheckedChanged است. این رویداد وقتی رخ می دهد که خاصیت Checked تغییر کند. در پنجره Properties و در بخش Events روی رویداد CheckedChanged دابل کلیک کنید تا پنجره کدنویسی آن باز شود، سپس دستورات زیر را در آن بنویسید.

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        button1.Enabled = true;
    }
    else
    {
        button1.Enabled = false;
    }
}
```

حالا برنامه را اجرا کرده و نتیجه را مشاهده کنید.

رویداد CheckStateChecked: زمانی که خاصیت Checked تغییر می کند این رویداد فعال می شود.

مثال ۴ - ۲: برنامه ای بنویسید که روز های هفته را با استفاده از کنترل های CheckBox نمایش دهد. با انتخاب هر یک از CheckBox ها نام روز و شماره روز هفته را نمایش می دهد، و

با ورود یک عدد (شماره روز هفته)، نام روز هفته را نمایش می دهد و CheckBox مربوط به آن را فعال می کند.

حل : ابتدا یک پروژه از نوع ویندوزی ساخته و Form آن را مانند پنجره تصویر ۱۱۷ - ۲ طراحی کنید.



تصویر ۱۷ - ۲

حال در رویداد CheckedChange هر یک از CheckBox ها دستورات زیر را تایپ می کنیم :

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "شنبه";
    textBox1.Text = "1";
}

private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "یکشنبه";
    textBox1.Text = "2";
}

private void checkBox3_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "دوشنبه";
    textBox1.Text = "3";
}

private void checkBox4_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "سه شنبه";
    textBox1.Text = "4";
}

private void checkBox5_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "چهارشنبه";
    textBox1.Text = "5";
}
```

```
private void checkBox6_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "پنج شنبه";
    textBox1.Text = "6";
}

private void checkBox7_CheckedChanged(object sender, EventArgs e)
{
    label1.Text = "جمعه";
    textBox1.Text = "7";
}
```

حال بر روی دکمه Click دابل کلیک کنید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    int a;
    a = Convert.ToInt32(textBox1.Text);
    switch (a)
    {
        case 1:
            label1.Text = "شنبه";
            checkBox1.Checked = true;
            break;
        case 2:
            label1.Text = "یکشنبه";
            checkBox2.Checked = true;
            break;
        case 3:
            label1.Text = "دوشنبه";
            checkBox3.Checked = true;
            break;
        case 4:
            label1.Text = "سه شنبه";
            checkBox4.Checked = true;
            break;
        case 5:
            label1.Text = "چهارشنبه";
            checkBox5.Checked = true;
            break;
        case 6:
            label1.Text = "پنج شنبه";
            checkBox6.Checked = true;
            break;
        case 7:
            label1.Text = "جمعه";
            checkBox7.Checked = true;
            break;
    }
}
```

هدف از این برنامه آشنایی بیشتر خوانندگان عزیز با کنترل CheckBox و دستور switch بود. در فصل قبل با ساختار switch آشنا شدید، در این بخش با یک مثال عملی این دستور شرح داده شده است. حال کافی است کلید F5 را از صفحه کلید فشار دهید تا برنامه اجرا گردد.

عملیات روی رشته ها

در زبان برنامه نویسی C#.NET برای ذخیره کردن مقادیر مختلف، نوع های داده ای متفاوتی وجود دارد. در فصل قبل نیز برخی از نوع های داده ای را مورد بررسی قرار دادیم. به خوبی می دانید که اگر به کامپیوتر اطلاعات درست بدهید جواب درست دریافت خواهید کرد و عکس این مطلب نیز صادق است. برای مثال برای ذخیره اعداد صحیح از نوع داده ای `int` استفاده می شود.

در این بخش قصد داریم با رشته ها آشنا شویم. رشته مجموعه ای از کاراکتر هاست که ابتدا و انتهای آن بوسیله علامت نقل قول " مشخص می شود مانند "Ali". در C# برای تعریف رشته ها از کلمه کلیدی `string` استفاده می شود.

شما هم در آینده و در پروژه های برنامه نویسی خودتان نیاز خواهید داشت که اطلاعات را به صورت رشته ذخیره کنید. برای مثال ذخیره نام و نام خانوادگی در یک دفترچه تلفن. این نکته را به خاطر داشته باشید، در صورتی که نخواهید بر روی نوع داده های عددی عملیات ریاضی انجام دهید می توانید اعداد را نیز به صورت رشته ذخیره کنید. برای درک بهتر لازم است که به سراغ کار عملی برویم یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۸ - ۲ طراحی کنید :



تصویر ۱۸ - ۲

بر روی دکمه `Click` دابل کلیک کنید تا وارد رویداد `Click` این کنترل شوید، سپس دستورات زیر را تایپ کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    string StrName;
    StrName = textBox1.Text;
    var StrFamily = textBox2.Text;
    var StrTel = textBox3.Text;
    MessageBox.Show("نام = " + StrName);
    MessageBox.Show("فامیلی = " + StrFamily);
    MessageBox.Show("تلفن = " + StrTel);
    Application.Exit();
}
```

در این برنامه در نخستین قدم یک متغیر رشته ای با نام StrName جهت نگهداری نام تعریف کرده ایم، در خط بعدی از این متغیر استفاده شده است و مقدار دریافتی از کنترل textBox1 را به این کنترل نسبت داده ایم. در خطوط بعدی از روش جدید مقدار دهی در C# استفاده کرده ایم، در این روش از کلمه کلیدی var برای تعریف متغیر ها استفاده شده که به کمک آن برنامه نویسان قادر خواهند بود متغیر های محلی خود را بدون ذکر صریح نوع آنها تعریف کنند. نوع متغیر پس از اولین اعلان تا اتمام حوزه تعریف آن تغییر نخواهد کرد و هر گونه تلاش برای تغییر نوع آن با خطا مواجه خواهد شد. توجه کنید که استفاده از var تنها در تعریف متغیر های محلی امکان پذیر است در اعلان متغیر ها به صورت سراسری پارامتر های توابع و مقادیر بازگشتی آنها نمی توانند از کلمه کلیدی var استفاده کنند.

با استفاده از کلمه کلیدی var متغیر های StrFamily و StrTel را در زمان تعریف با استفاده از کنترل های textBox2 و textBox3 مقدار دهی کرده ایم. در خطوط بعدی از دستور MessageBox برای نمایش مقادیر متغیر StrFamily، StrName و StrTel استفاده شده که در نهایت با اجرای دستور Application.Exit() برنامه بسته می شود. برای این کار از متد Exit() در کلاس Application استفاده کرده ایم.

در مجموع عملیات های زیادی را می توان بر روی رشته ها انجام داد برای مثال یکی از مواردی که حتما مورد نیاز خواهد بود به دست آوردن طول رشته، زیر رشته، جایگزینی رشته، و قالب بندی رشته است.

یک پروژه جدید ویندوزی ساخته و یک Form مانند پنجره تصویر ۱۹ - ۲ طراحی کنید. چهار کنترل Button، یک کنترل textbox و یک کنترل label روی فرم قرار دهید.



تصویر ۱۹ - ۲

روی دکمه (طول رشته) دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید :

```
private void Length_Click(object sender, EventArgs e)
{
    string strInput;
    strInput = textBox1.Text;
    MessageBox.Show(strInput.Length + "تعداد کاراکتر ها", "رشته ها");
}
```

زمانی که با یک متغیر رشته ای مواجه هستیم می توانیم از خاصیت Length آن برای دریافت تعداد حروف رشته استفاده کنیم، این خاصیت مقداری را از نوع عدد صحیح به عنوان طول رشته بر می گرداند. توجه کنید که کامپیوتر هر کاراکتری از قبیل فضا های خالی و علامت ها را در محاسبه طول رشته به حساب می آورد. حالا بر روی دکمه (زیر رشته) دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید :

```
private void Substring_Click(object sender, EventArgs e)
{
    var strInput = textBox1.Text;
    MessageBox.Show(strInput.Substring(0, 5), "رشته");
    MessageBox.Show(strInput.Substring(5, 3), "رشته ها");
    MessageBox.Show(strInput.Substring(strInput.Length - 5), "رشته ها");
}
```

در خط اول با استفاده از کلمه کلیدی var متغیر strInput را تعریف کرده ایم و در همان جا با استفاده از مقدار textBox1 آن را مقدار دهی کرده ایم. در خط بعدی از متد Substring استفاده شده است، ممکن است که در برنامه بخواهید به جای کار با تمام رشته با قسمتی از آن کار کنید. به طور مثال می خواهید از یک مجموعه از کاراکتر ها که در اول رشته و یا در آخر آن آمده اند استفاده کنید، این مجموعه کاراکتر ها که ممکن است از هر جای رشته شروع شوند و به هر جایی در رشته ختم شوند، که آن را زیر رشته می نامیم.

متد Substring به شما این امکان را می دهد تا هر قسمتی از یک مجموعه کاراکتر را از هم جدا کنید. این متد به دو روش می تواند فراخوانی شود روش اول این است که شماره کاراکتر اول و تعداد کاراکتر هایی که نیاز دارید را به تابع بدهید. به دو دستور MessageBox اول و دوم توجه کنید که دو پارامتر را از ورودی دریافت می کند، اولی شماره شروع کاراکتر و دومی تعداد کاراکتر های درخواست می باشد. و روش دوم این است که وقتی تابع را فراخوانی می کنید فقط یک پارامتر را برای آن مشخص می کنید.

به دستورات استفاده شده در MessageBox آخر توجه کنید، این پارامتر به تابع می گوید که از مکان مشخص شده شروع کند و تمام کاراکتر های سمت راست آن را جدا کند. بنابراین با استفاده از خاصیت Length به تابع Substring اعلام می کنید که از ۵ کاراکتر مانده به انتهای رشته شروع کند و تمام کاراکتر های باقیمانده را برگرداند.

حالا بر روی دکمه (جایگزینی) دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید :

```
private void button3_Replace(object sender, EventArgs e)
{
    var strInput = textBox1.Text;
    string strNewData;
    strNewData = strInput.Replace("برنامه نویس", "کاربر");
    MessageBox.Show(strNewData, "رشته ها");
}
```

متد Replace دو رشته را دریافت می کند و در متن به دنبال رشته اول می گردد، سپس در هر قسمتی از متن که رشته اول را پیدا کرد آن را با رشته دوم جایگزین می کند. بعد از اینکه رشته اول در تمام متن با رشته دوم جایگزین شد عبارت جدید به شما برگردانده می شود و می توانید آن را نمایش دهید.

حال بر روی دکمه (قالب بندی) دابل کلیک کنید تا وارد رویداد Click آن شوید. قالب بندی رشته ها بسیار اهمیت دارد و در نمایش خروجی بسیار تاثیر گذار است. خروجی هر نرم افزار مثل خمیر بازی بایستی به هر شکلی که برنامه نویس می خواهد درآید و مهم تر از همه خواسته مشتری را برآورده سازد.

```
private void Formatting_Click(object sender, EventArgs e)
{
    double dblNumber = 13;
    dblNumber = 13 / 7;
    MessageBox.Show("بدون قالب بندی خاص: " + dblNumber, "Floating Points");
    MessageBox.Show("با قالب بندی خاص: " + string.Format("{0:n3}", dblNumber), "Floating Points");
}
```

همانطور که در دستورات بالا مشاهده می کنید، در خط اول متغیری از نوع double با مقدار ۱۳ تعریف شده است. در خط بعدی این مقدار بر ۷ تقسیم می شود در این حالت تعداد اعشار آن بسیار زیاد می شود می توان مقدار اعشار آن را کنترل کرد و در یک قالب مناسب نمایش داد. در دستور MessageBox خط آخر با استفاده از تابع string.Format به شما اجازه می دهد متن و اعداد خود را با قالب مناسب نمایش دهید.

تنها نکته مهمی که این تابع دارد پارامتر اول آن است که مشخص می کند خروجی تابع باید در چه قالبی باشد برای فراخوانی متد string.Format بایستی دو پارامتر به آن ارسال کنیم، پارامتر اول قالب رشته ای است که می خواهید از متد دریافت کنید پارامتر دوم dblNumber مقداری است که می خواهید قالب بندی کنید. عدد صفر در پارامتر اول مشخص می کند که در حال مشخص کردن اولین قالب پارامتر بعد از پارامتر حاضر یا همان dblNumber هستیم. قسمتی که بعد از دو نقطه آمده است مشخص می کند که به چه صورتی می خواهیم رشته را قالب بندی کنیم. در اینجا به صورت n۳ استفاده کرده ایم که به معنای یک رقم با ممیز شناور و سه رقم اعشار است. در این حالت عدد را با دو رقم اعشار داریم.

کنترل RadioButton

کنترل RadioButton مشابه کنترل CheckBox می باشد. اما یک تفاوت دارد و آن این است که فقط یکی از آیتم ها را می توان در زمان اجرا انتخاب کرد. اصولاً از RadioButton ها برای پرسیدن یک سوال مانند تعیین جنیست و یا انتخاب یک گزینه خاص استفاده می شود. بیشتر خواص این کنترل مشابه خواص کنترل CheckBox می باشد.

کنترل GroupBox

از این کنترل می توان برای گروه بندی مجموعه ای از کنترل ها استفاده کرد. این کنترل مثل سایر کنترل های محیط Visual Studio دارای یک خصیصه خاص به نام text است که می

تواند عنوانی را برای کنترل های خودش در نظر بگیرد. این کنترل در بخش Containers در پنجره Toolbox قرار دارد.

مثال ۵-۲: می خواهیم برنامه ای بنویسیم که با انتخاب یکی از کنترل های RadioButton، آیتم انتخاب شده در یک کنترل Label نمایش داده شود.

حل: ابتدا یک پروژه جدید از نوع ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۲۰ - ۲ طراحی کنید.



تصویر ۲۰ - ۲

حال رویداد ChackedChanged کنترل RadioButton\ یعنی C#.NET دستورات زیر را بنویسید.

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    // Executed when any radio button is changed.
    // ... It is wired up to every single radio button.
    // Search for first radio button in GroupBox.
    string result1 = null;
    foreach (Control control in this.groupBox1.Controls)
    {
        if (control is RadioButton)
        {
            RadioButton radio = control as RadioButton;
            if (radio.Checked)
            {
                result1 = radio.Text;
            }
        }
    }
    // Search second GroupBox.
    string result2 = null;
    foreach (Control control in this.groupBox2.Controls)
    {
        if (control is RadioButton)
        {
            RadioButton radio = control as RadioButton;
            if (radio.Checked)
```

```

    {
        result2 = radio.Text;
    }
}
label1.Text = result1 + " " + result2;
}

```

حال اگر برنامه را اجرا کنید می توانید نتیجه را مشاهده کنید.

نکته !

قبل از بررسی دستورات می توانید برای خوانایی و اشکال زدایی برنامه های خودتان برای هر یک از سطر های دستورات خودتان شماره تعیین کنید. برای این منظور از منوی Tools گزینه Options و سپس گزینه Text Editor را کلیک کنید، و از آنجا گزینه Display و سپس گزینه Line Numbers را کلیک کنید و در نهایت روی OK کلیک کنید تا خطوط دستورات دارای شماره گذاری باشد. این کار لزومی ندارد ولی جهت خوانایی و اشکال زدایی می تواند به برنامه نویس کمک کند.

مثال ۶-۲: برنامه ای بنویسید که هزینه حمل کالا به وسیله وسایل نقلیه مختلف را محاسبه کند. به این منظور کاربر وزن کالا و مسافت حمل را به همراه روش حمل آن معین می کند و سپس هزینه حمل کالا محاسبه شده، در اختیار وی قرار می گیرد. روش های حمل کالا و هزینه هر یک به ازای هر کیلومتر جابجایی در جدول ۱-۲ آمده است و برای محاسبه هزینه حمل کالا از فرمول زیر استفاده می شود.

هزینه به ازای یک کیلومتر جابجایی * مسافت جابجایی * وزن کالا = هزینه حمل

هزینه حمل به ریال	روش حمل کالا
۱۰۰۰	اتوبوس
۱۵۰۰۰	قطار
۱۴۰۰۰	کشتی
۲۰۰۰۰	هواپیما

جدول ۱-۲

حل : ابتدا یک پروژه از نوع ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۲۱-۲ طراحی کنید.

تصویر ۲۱ - ۲

سه عدد کنترل Label دو عدد کنترل TextBox دو عدد کنترل GroupBox که هر کدام شامل ۴ کنترل RadioButton می باشد که خاصیت Text هر کدام را می توانید مطابق پنجره تصویر ۲۱ - ۲ طراحی کنید.

خاصیت Checked دو کنترل RadioButton های مربوط به (اتوبوس) و (گرم) را برابر True قرار دهید. ابتدا دو متغیر به نام های sngw و intrtrans به صورت Public تعریف کنید محل تعریف متغیر های Public زیر کلاس Public Partial می باشد.

```
public partial class Form1 : Form
{
    int intrtrans;
    float sngw;
```

بعد از تعریف متغیر ها نوبت به نوشتن دستورات مربوط به رویداد Click هر یک از RadioButton هاست.

```
private void bus_Click(object sender, EventArgs e)
{
    intrtrans = 0;
}

private void train_Click(object sender, EventArgs e)
{
    intrtrans = 1;
}

private void ship_Click(object sender, EventArgs e)
{
    intrtrans = 2;
}
```

```
private void plane_Click(object sender, EventArgs e)
{
    inttrans = 3;
}

private void gram_Click(object sender, EventArgs e)
{
    sngw = 0;
}

private void kiligram_Click(object sender, EventArgs e)
{
    sngw = 1;
}

private void ton_Click(object sender, EventArgs e)
{
    sngw = 1000;
}

private void millionton_Click(object sender, EventArgs e)
{
    sngw = 1000000;
}
```

بر روی دکمه Click دابل کلیک کنید تا رویداد Click آن باز شود، سپس دستورات زیر را در آن بنویسید.

```
private void compute_Click(object sender, EventArgs e)
{
    double sngp;
    double weight, length;
    weight = Convert.ToInt32(textBox1.Text);
    length = Convert.ToInt32(textBox2.Text);
    sngp = sngw*weight*length;
    switch (inttrans)
    {
        case 0:
            sngp = sngp * 10000;
            break;
        case 1:
            sngp = sngp * 15000;
            break;
        case 2:
            sngp = sngp * 14000;
            break;
        case 3:
            sngp = sngp * 20000;
            break;
    }
    MessageBox.Show("میزان پرداختی " + sngp.ToString() + " و", "مجموع پرداختی");
}
```

برنامه را اجرا کرده و نتیجه را با مقادیر مختلف بررسی کنید.

نکته !

یکی از ویژگی های منحصر بفرد نرم افزار Visual Studio که باعث می شود خطا های دستوری در یک برنامه به حداقل برسد ویژگی تکمیل خودکار متن یا Intelecence است. به وسیله این ویژگی زمانی که می خواهید دستوری را بنویسید کادری برای شما باز می شود و براساس چندین فاکتور مختلف از قبیل حروفی که وارد کرده اید، دستورات قبلی، و یک سری اطلاعات دیگر، عبارتی را برای قرار دادن در آن مکان پیشنهاد می کند به این وسیله می توانید به سرعت و بدون اینکه چیزی را به خاطر بسپارید نام اعضای کلاس ها، نام ساختار ها، و یا فضای نام هایی که با آن کار می کنید در برنامه وارد کنید. Intelecence اسم هر عضو از کلاس را نمایش می دهد، در سمت چپ هر عنصر یک آیکن قرار دارد که نوع این عضو را مشخص می کند آیکن ها و انواع آن ها به صورت جدول ۲ - ۲ است.

آیکن	عنوان
	کلمه کلیدی
	متد
	خاصیت
	کلاس
	شمارنده
	ساختار
	رابط
	اشاره گر متد
	فضای نام
	رهنمود
	فیلد
	رویداد

جدول ۲ - ۲

گزینه هایی که در جدول ۲ - ۲

تا به حال با برخی از

وجود دارد کار کرده ایم، با گذشت فصل هایی از کتاب با تمام این بخش ها کار کرده و آشنا خواهید شد.

کنترل ListBox

بعضی از مواقع لازم می شود تا لیست بلند بالایی را برای انتخاب در اختیار کاربران خود قرار دهید، منطقی نیست که از RadioButton ها یا CheckBox ها استفاده کنید. در این حالت کنترل دیگری به نام LsitBox وجود دارد که می توانید به آسانی از این کنترل استفاده کنید، این کنترل می تواند لیست بزرگی از آیتم های مختلف را برای انتخاب در اختیار برنامه نویسان قرار دهد. این کنترل در پنجره Toolbox در بخش Common Controls قرار دارد، و به سادگی می توانید این کنترل را بر روی Form خودتان قرار داده و استفاده کنید. این کنترل نیز دارای خصوصیات متعددی است که در روند های برنامه نویسی به برنامه نویس کمک شایانی می کند :

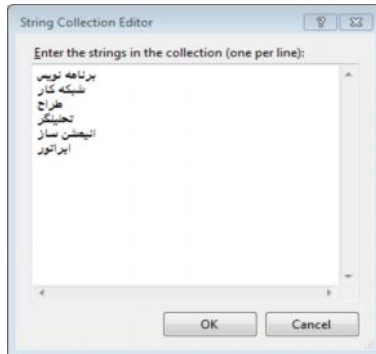
خاصیت ColumnWidth : این خاصیت در رابطه با استفاده از کنترل ListBox به صورت چند ستونی است که می توان پهنای هر ستون را تعیین کرد.

خاصیت HorizontalExtent : این خاصیت در رابطه با تنظیم میزان جابجایی افقی به کار می رود.

خاصیت HorizontalScrollbar : وضعیت نمایش یا عدم نمایش نوار جابجایی افقی را تعیین می کند.

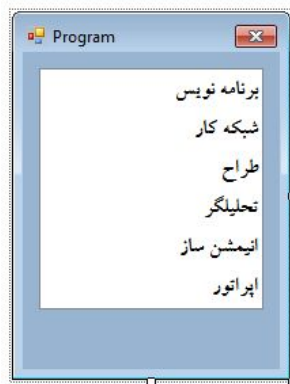
خاصیت ItemHeight : با استفاده از این خاصیت می توان ارتفاع هر آیتم در جعبه لیست را تعیین کرد.

خاصیت Items : می تواند مانند یک کلوکسیون، آیتم های مورد نیاز شما را در خود جای دهد روی این خاصیت دابل کلیک کنید تا پنجره آن باز شود، سپس مقادیر زیر را وارد کنید (پنجره تصویر ۲۲ - ۲).



تصویر ۲۲ - ۲

پس از ورود مقادیر مورد نظر روی دکمه OK کلیک کنید.



تصویر ۲۳ - ۲

همانطور که در پنجره تصویر ۲۳ - ۲ مشاهده می کنید، آیتم های مورد نظر به کنترل ListBox اضافه شده است.

خاصیت MultiColumn : تعیین می کند که آیا کنترل شما می تواند دارای چندین ستون باشد یا خیر؟

خاصیت SelectionMode : تعیین می کند که آیا کاربر امکان انتخاب یک یا چند گزینه را در اختیار داشته باشد یا خیر؟ و یا اصلاً حق انتخاب هیچ گزینه ای را در لیست ندارد.

خاصیت ScrollAlwaysVisible : وضعیت نمایش نوار جابجایی عمودی را بصورت دائم تعیین می کند.

خاصیت Sorted: به شما این امکان را می دهد که آیتم های موجود در ListBox خود را به صورت مرتب شده نشان دهید.

خاصیت SelectedIndex: شماره آیتمی را تعیین می کند که باید انتخاب شود. شماره گزینه ها از صفر شروع می شود.

خاصیت SelectedItems: این خاصیت آیتم هایی را تعیین می کند که باید انتخاب شوند.
خاصیت SelectedValue: مقداری را تعیین می کند که گزینه مربوط به آن باید انتخاب شود.
حال به بررسی برخی از رویداد مهم این کنترل می پردازیم:

رویداد DrawItem: این رویداد زمانی رخ می دهد که آیتمی در حال نمایش باشد.
رویداد SelectedIndexChanged: این رویداد زمانی رخ می دهد که کاربر باعث بوجود آمدن تغییر در index آیتم انتخابی می شود. می خواهیم از این رویداد به نحوی استفاده کنیم که وقتی کاربر آیتمی را از ListBox انتخاب می کند این آیتم به عنوان Form جاری انتخاب شود در Form_Load دستورات زیر را بنویسید:

```
private void Form1_Load(object sender, EventArgs e)
{
    listBox1.Items.Add("سهیل");
    listBox1.Items.Add("مسعود");
    listBox1.Items.Add("مهدی");
    listBox1.Items.Add("وحید");
}
```

با دستورات بالا گزینه های سهیل، مسعود، مهدی و وحید به کنترل listBox1 ما اضافه می شوند. در این دستورات برای اضافه کردن آیتم های مورد نظرمان از متد Add در خاصیت Items استفاده کردیم. حالا در رویداد SelectedIndexChanged دستورات زیر را بنویسید:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.Text = Convert.ToString(listBox1.SelectedItem);
}
```

با دستورات بالا هر آیتمی را که از کنترل listBox1 در زمان اجرا انتخاب کنید به عنوان عنوان فرم نمایش داده خواهد شد. بعضی از مواقع نیز لازم است تا شما تک آیتم های جعبه متن خود را توسط یک حلقه بررسی کنید برای این کار لازم است که تعداد آنها را

داشته باشید. به راحتی می توانید تعداد آنها را مشخص کرده و در عنوان Form خودتان نمایش دهید.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.Text = Convert.ToString(listBox1.Items.Count);
}
```

در صورتی که برنامه را اجرا کنید به راحتی می توانید تعداد آیتم های موجود در کنترل listBox1 را مشاهده کنید. برای این کار از متد Count در خاصیت Items استفاده کرده ایم. با دو متد Add و Count آشنا شدید حال قصد داریم تا برخی از متد های پرکاربرد دیگر را نیز در این بخش برای شما معرفی کنیم :

متد Insert : برای اضافه کردن یک گزینه در یک مکان خاص در کنترل listBox به کار می رود.

```
listBox1.Items.Insert(4, "محمد رضا");
```

دستور بالا آیتم (محمد رضا) را به عنوان چهارمین گزینه به لیست آیتم های کنترل listBox اضافه می کند.

متد Remove : برای پاک کردن گزینه مورد نظر از کنترل listBox به کار می رود.

```
listBox1.Items.Remove("بهنام");
```

در دستورات بالا آیتم (بهنام) از لیست کنترل های listBox حذف می شود.

متد IndexOf : مقداری را دریافت کرده و شماره اندیس آن را نشان می دهد.

متد RemoveAt : شماره آیتمی را دریافت کرده و آن را حذف می کند.

```
listBox1.Items.RemoveAt(4);
```

۴ شماره index ای است که باید حذف شود.

متد GetType : نوع آیتم های داخل کنترل listBox را بر می گرداند.

```
متغیر = listBox1.Items.GetType();
```

مثال : با استفاده از حلقه های do...while و while و کنترل listBox برنامه ای بنویسید که اعداد تصادفی در بازه ۰ تا ۲۷ را تولید نماید؟

حل : ابتدا یک پروژه از نوع ویندوزی ایجاد کرده، سپس Form آن را مانند پنجره تصویر ۲۴ - ۲ طراحی

کنید.



تصویر ۲۴ - ۲

خاصیت HorizontalScrollbar کنترل ListBox را بر روی مقدار True قرار دهید، سپس دستورات زیر را در رویداد Click دکمه Click بنویسید.

```
private void Click_Click(object sender, EventArgs e)
{
    Random ObjRandom = new Random();
    int intRandomNumber = 0;
    listBox1.Items.Clear();
    do
    {
        intRandomNumber = ObjRandom.Next(28);
        listBox1.Items.Add(intRandomNumber);
    }
    while (intRandomNumber != 10);
}
```

حال برنامه را اجرا کرده و نتیجه را بررسی کنید.

در این مثال در خط اول ابتدا یک متغیر به نام ObjRandom از کلاس Random ایجاد کردیم، این متغیر توابع مورد نیاز برای تولید اعداد تصادفی را در اختیار شما قرار می دهد. به نحوه تعریف متغیر ObjRandom توجه کنید، همانطور که می دانید زمانی که یک متغیر ایجاد شد بایستی مقدار اولیه آن را مشخص کرد و سپس از آن استفاده کرد. برای بعضی متغیر ها همانند اعداد صحیح به راحتی می توان مقدار اولیه مشخص کرد اما برخی از متغیر ها را نمی توان مانند متغیر های از نوع اعداد صحیح مقدار دهی کرد. برای مقدار دهی به این نوع متغیر که به صورت یک شیء شناخته می شوند از کلمه کلیدی new استفاده می کنیم در این حالت خود کلاس برای شیء یک مقدار اولیه ایجاد کرده و آن را به شیء جدید نسبت می دهد. متغیر بعدی مورد استفاده متغیر intRandomNumber می باشد که وظیفه نگهداری اعداد

تصادفی تولید شده توسط شیء ObjRandom را بر عهده دارد. در خط بعدی متد Clear() برای پاک کردن محتویات کنترل ListBox به کار می رود. سپس برنامه وارد حلقه do می شود و تا تولید ۱۰ عدد به اجرای خود ادامه می دهد. همانطور که می دانید در حلقه do ابتدا یک بار دستورات اجرا می شوند و سپس شرط حلقه بررسی می شود با هر بار اجرای حلقه عدد تصادفی جدیدی تولید و مقدار آن در متغیر intRandomNumber ذخیره می گردد. برای تولید عدد تصادفی از تابع Next() در شیء ObjRandom استفاده شده که یک پارامتر به عنوان بزرگترین عددی که می تواند تولید کند دریافت می کند. سپس مشخص می کند که تابع باید عدد تصادفی در بازه ۰ تا ۲۷ تولید کرده و در متغیر intRandomNumber قرار دهد. در ادامه با استفاده از متد Add عدد تولید شده توسط تابع Random به لیست کنترل ListBox اضافه می شود و در نهایت در حلقه بررسی می شود که آیا عدد تولید شده برابر با ۱۰ بوده است یا خیر؟ اگر عدد تولید شده مخالف ۱۰ باشد برنامه به ابتدای دستورات حلقه do بر می گردد و بار دیگر دستورات داخل حلقه اجرا می شوند. در غیر این صورت از حلقه خارج شده و به اولین خط بعد از حلقه می رود.

به این نکته توجه کنید که دستورات در این حلقه قبل از بررسی شرط حتما یک بار اجرا می شوند. نیازی نیست که به متغیر intRandomNumber مقدار اولیه اختصاص داد زیرا این متغیر در مرحله اولی که دستورات حلقه اجرا می شوند مقدار می گیرد. حال می خواهیم این برنامه را عینا با استفاده از حلقه while بنویسیم تا به طور عملی تفاوت این دو ساختار را با هم ببینید :

```
private void Click_Click(object sender, EventArgs e)
{
    Random ObjRandom = new Random();
    int intRandomNumber = 0;
    listBox1.Items.Clear();
    while (intRandomNumber != 10)
    {
        intRandomNumber = ObjRandom.Next(28);
        listBox1.Items.Add(intRandomNumber);
    }
}
```

عملکرد حلقه while نیز همانند حلقه do است، با این تفاوت که شرط حلقه در ابتدای حلقه بررسی می شود، سپس در صورت درست بودن آن، دستورات داخل حلقه اجرا می شوند.

مثال ۷ - ۲: برنامه ای بنویسید که یک عدد را از کاربر دریافت کرده و نشان دهد که آیا عدد مزبور یک عدد اول است یا خیر؟

ابتدا یک پروژه جدید از نوع ویندوزی ایجاد کرده و سپس Form آن را مطابق تصویر ۲۵ - ۲ طراحی کنید :



تصویر ۲۵ - ۲

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید :

```
private void Compute_Click(object sender, EventArgs e)
{
    int a = 2;
    var num = int.Parse(textBox1.Text);
    while (a < num)
    {
        if (num % a == 0)
        {
            MessageBox.Show("این عدد اول نیست");
            break;
        }
        a++;
        if (num == a)
        {
            MessageBox.Show("این عدد اول است");
            break;
        }
    }
}
```

در برنامه از یک کنترل TextBox و یک کنترل Label و Button مطابق پنجره تصویر ۲۳ - ۲ استفاده کرده ایم. در بخش کدنویسی این برنامه از دو متغیر استفاده شده که متغیر a برای نگهداری اعداد تقسیم شونده، که مقدار اولیه آن ۲ می باشد. و متغیر num که برای نگهداری اعداد ورودی کاربر در نظر گرفته شده است. برای تبدیل یک رشته که شامل عدد است به یک عدد صحیح باید از تابع Parse در کلاس int استفاده کنیم. این تابع یک رشته را که شامل یک

عدد است دریافت کرده و عدد معادل آن را بر می گردانند. اگر رشته ای که به این تابع فرستاده می شود از نوع داده عددی نباشد تابع با خطا مواجه شده و اجرای برنامه متوقف می شود. شرط حلقه شامل کوچک بودن متغیر a نسبت به متغیر num می باشد، در صورتی که مقدار باقیمانده تقسیم num بر a مساوی صفر باشد در پیغامی نشان می دهد که این عدد اول نیست. با استفاده از دستور `break` از حلقه خارج می شویم و در یک بررسی دیگر اگر مقدار متغیر num برابر متغیر a بود با چاپ پیغامی اعلام می کنیم که عدد مورد نظر اول است. دوباره با استفاده از دستور `break` از حلقه خارج می شویم.

آشنایی با دستور `break`

دستور `break` موجب خروج از هر حلقه ای می شود. در این حالت زمانی که کامپایلر به این دستور رسید به اولین خط بعد از حلقه می رود و آن را اجرا می کند. اگر برنامه ای که نوشته اید شامل چندین حلقه تو در تو می باشد این دستور موجب خروج از داخلی ترین حلقه تکرار می شود.

کنترل `CheckBox`

این کنترل لیستی از `CheckBox` ها را در اختیار کاربران قرار می دهد. این کنترل را می توانید از پنجره `Toolbox` و بخش `Common Controls` به `Form` خودتان اضافه کنید. تقریباً بیشتر پارامترها و خصوصیات این کنترل، مشابه کنترل `ListBox` می باشد فقط تفاوت در این است که کاربر در این کنترل می تواند یک آیتم را ابتدا انتخاب کرده و سپس آن را فعال کند. خاصیت `OnClick` : این خاصیت زمانی که در حالت `True` قرار بگیرد این امکان را در اختیار کاربران قرار می دهد که با یک بار کلیک روی آیتم مورد نظر خود، همزمان آیتم مورد نظر خود را علامت دار و در حالت انتخابی قرار دهد. خاصیت `IntegralHeight` : با استفاده از این خاصیت می توان تعیین کرد که آیا لیست بصورت کامل ارائه شود یا خیر. خاصیت `SelectionMode` : این خاصیت حالت های مختلف انتخاب را به صورت یک انتخابی نه چند انتخابی در اختیار کاربر قرار می دهد.

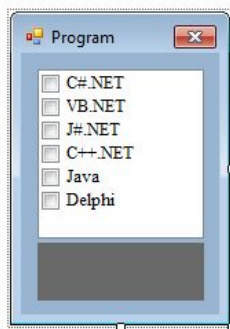
خاصیت ThreeDCheckBoxes : دو حالت Flat, Normal را برای CheckBox های داخل این کنترل تعیین می کند.

مهمترین رویداد های این کنترل نیز مشابه کنترل ListBox رویداد SelectedIndexChanged و SelectValueChange است. رویداد مهم دیگر ItemCheck است، این رویداد وقتی رخ می دهد که وضعیت Checked بودن یک آیتم در لیست آیتم های این کنترل تغییر کند. دو متد پر کاربرد در این کنترل متد SetItemChecked که جهت تعیین مقدار عنصر خاصی به کار می رود. و متد SetItemCheckState که وضعیت فعال بودن یک عنصر خاص را تعیین می کند.

```
private void Form1_Load(object sender, EventArgs e)
{
    checkedListBox1.SetItemChecked(1, true);
}
```

مثال ۸ - ۲: می خواهیم برنامه ای بنویسیم که کاربر با انتخاب یکی از آیتم های موجود در کنترل CheckedListBox، عنوان آیتم انتخاب شده در یک کنترل Label نمایش داده شود. حل : ابتدا یک پروژه جدید از نوع ویندوزی ساخته، سپس Form آن را مطابق پنجره تصویر ۲۶ - ۲ طراحی کنید.

با استفاده از خاصیت Items مقادیر زیر را در کنترل CheckedListBox قرار دهید.



تصویر ۲۶ - ۲

حال کنترل CheckedListBox را انتخاب کرده و در پنجره Properties و از بخش Events، رویداد SelectedIndexChanged را دابل کلیک کنید تا بخش کد نویسی آن باز شود. سپس دستورات زیر را بنویسید.

```
private void checkedListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    int selected = checkedListBox1.SelectedIndex;
    if (selected != -1)
    {
        label1.Text = checkedListBox1.Items[selected].ToString();
    }
}
```

حال برنامه را اجرا کرده و نتیجه را بررسی کنید.

کنترل ComboBox

کنترل ComboBox یکی از کنترل های پرکاربرد در محیط های برنامه نویسی است، از طریق این کنترل لیستی از اطلاعات در اختیار کاربر قرار می گیرد. کنترل ComboBox در اصل یک TextBox است که دیگر نیاز نیست کاربر چیزی درون آن تایپ کند. برخی از خصوصیات مهم این کنترل عبارتند از :

خاصیت DrawMode : وضعیت منوی بازشوی کنترل مذکور را در حالت های مختلف مشخص می کند.

خاصیت Item : که جهت قرار دادن آیتم های مختلف برای انتخاب کاربر را در اختیار کاربر قرار می دهد.

خاصیت MaxDropDownItems : این خاصیت مشخص می کند که حداکثر چند آیتم در منوی بازشوی کنترل ComboBox می تواند قرار گیرد. در صورتی که مقدار این خاصیت بیش از ۶ بود یک نوار لغزان باز می شود.

خاصیت DropDownStyle : در صورتی که مقدار این خاصیت روی حالت Single قرار گیرد، کنترل ComboBox تبدیل به یک جعبه متن می شود. حالت مناسب دیگر DropDownList می باشد در این وضعیت هیچ کس نمی تواند چیزی را درون ComboBox تایپ کند بلکه بایستی از لیستی که در اختیار دارد انتخاب کند.

برای درک مطالب بیان شده بهتر است که به صورت عملی با این کنترل کار کنیم، پس یک پروژه جدید ویندوزی ساخته و یک Form مانند پنجره تصویر ۲۷ - ۲ طراحی کنید.



تصویر ۲۷ - ۲

بر روی Form دابل کلیک کنید تا وارد Form_Load شوید، سپس دستورات زیر را تایپ کنید :

```
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 1; i <= 10; i++)
    {
        comboBox1.Items.Add(i.ToString());
    }
}
```

وقتی برنامه را اجرا کنید مشاهده خواهید کرد که کنترل ComboBox حاوی اعدادی شده است که در حلقه for تعیین کرده ایم. حال بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را تایپ کنید :

```
private void Click_Click(object sender, EventArgs e)
{
    MessageBox.Show(comboBox1.SelectedItem.ToString());
}
```

حال اگر برنامه را اجرا کنید و روی دکمه Click کلیک کنید آیتمی را که انتخاب شده است به ما نشان می دهد. اما حالت دیگری نیز برای نشان دادن انتخاب ها وجود دارد به دستورات زیر توجه کنید :

```
private void Click_Click(object sender, EventArgs e)
{
    MessageBox.Show(comboBox1.SelectedIndex.ToString());
}
```

وقتی برنامه را اجرا کنید و عدد ۱ را انتخاب کنید index عدد ۱ را نشان خواهد داد. رویداد مهم کنترل ComboBox رویداد SelectedIndexChanged است. البته چون کنترل ComboBox در حقیقت یک جعبه متن کامل است می تواند رویداد TextChanged را در اختیار شما قرار دهد. به نحوه استفاده از رویداد SelectedIndexChanged دقت کنید :

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.Text = comboBox1.Text;
}
```

کنترل ComboBox بیشتر قدرت نمایی می کند وقتی که به بانک اطلاعاتی متصل باشیم ولی در حالت عادی بیشتر از این مطلبی برای گفتن وجود ندارد.

کنترل ListView

از این کنترل برای به نمایش گذاشتن آیتم های مختلف در یک پنجره استفاده می شود. نمونه واقعی آن در محیط ویندوز پنجره My Computer می باشد که در آن می توانید لیست تمام آیتم ها را برحسب وضعیت های مختلف مشاهده کنید و براحتی از تک تک آنها استفاده کنید. با استفاده از کنترل ListView می توانید لیستی از اشیاء را به نمایش در بیاورید این کنترل را می توانید از پنجره Toolbox و بخش Common Controls به Form خودتان اضافه کنید. حال به معرفی برخی از خواص مهم کنترل ListView در پنجره Properties می پردازیم :

خاصیت Activation : نحوه فعال سازی آیتم های موجود در سطح کنترل را تعیین می کند.

خاصیت AutoArrange : آیتم های موجود در کنترل ListView را مرتب می کند.

خاصیت CheckBox : باعث قرار گرفتن یک CheckBox در کنار هر آیتم می شود.

خاصیت Column : امکان ایجاد ستون ها را به کمک پنجره Editor مربوطه فراهم می آورد.

خاصیت GridLines : باعث نمایش خطوط Grid ما بین ستون ها و آیتم ها می شود.

خاصیت Group : به کمک پانل Editor مربوطه امکان گروه بندی آیتم ها فراهم می شود.

خاصیت HeaderStyle : نحوه سبک سرتیر ستون ها را در حالت های مختلف تعیین می کند.

خاصیت HoverSelection : این خصیصه امکان انتخاب آیتم ها را با قرار گرفتن ماوس بر روی آنها در اختیار کاربران قرار می دهد.

خاصیت Items : به کمک CollectionEditor امکان ایجاد آیتم های مختلف در لیست را فراهم می کند.

خاصیت LabelEdit : امکان ویرایش عنوان هر آیتم را در اختیار کاربر قرار می دهد.

خاصیت Image List : LargeImageList مربوط به تصاویر را در حالت بزرگ نمایی فراهم می کند.

خاصیت MultiSelect : امکان انتخاب چند آیتم به صورت همزمان را فراهم می آورد.

خاصیت Scrollable : امکان Scroll کردن آیتم ها را زمانی که تعداد آنها بیشتر از محدوده نمایش ListView است را فراهم می کند.

خاصیت ShowGroup : امکان نمایش آیتم ها را در گروههای مشخص شده تعیین می کند.

خاصیت SmallImageList : لیست تصاویر آیتم ها را مواقعی که در حالت Small View قرار دارد مشخص می کند.

خاصیت Sorting : نحوه مرتب سازی آیتم ها را مشخص می کند.

خاصیت TileSize : اندازه تصویر را مشخص می کند.

خاصیت View : نوع نمایش آیتم ها را در حالت های مختلف فراهم می کند.

یکی از رویداد های مهمی که به کمک آن می توانید رفتار کاربران را با آن تشخیص دهید رویداد ItemSelectionChange است. این رویداد دارای آرگومانی به نام e است که دقیقا برای شما مشخص می کند که کدام آیتم انتخاب شده و چه شرایطی دارد.

کنترل Tooltip

حتما با پنجره های کوچک زرد رنگ در محیط سیستم عامل ویندوز برخورد کرده اید این پنجره ها در حکم راهنمای سریع برای کاربران می باشد به این پنجره های کوچک زرد رنگ اصطلاحا Tooltip می گویند. نرم افزار Visual Studio این امکان را برای برنامه نویسان فراهم کرده است که بتوانند برای بخش های مختلف برنامه خود از این امکان بهره ببرند. Tooltip یک کنترل غیر بصری است از پنجره Toolbox بخش Common Controls می توانید یک نمونه از این کنترل را به Form خودتان اضافه کنید. برخی از خاصیت های مهم این کنترل عبارتند از :

خاصیت Active : امکان فعال یا غیر فعال بودن را در روی Form تعیین می کند.

خاصیت AutomaticDelay : میزان تاخیر زمانی برای به نمایش درآمدن یک Tooltip بر روی یک Object را براساس میلی ثانیه تعیین می کند.

خاصیت AutoPopDelay : میزان تاخیر زمانی یا طول کل مدت زمان نمایش یک Tooltip را مشخص می کند.

خاصیت IsBallon : تعیین می کند آیا Tooltip دارای Ballon باشد یا به صورت ساده نمایش داده شود.

خاصیت ReShowDelay : فاصله زمانی یا تاخیر زمانی را برای نمایش Tooltip برحسب میلی ثانیه تعیین می کند.

خاصیت ShowAlways : باعث نمایش دائمی یک Tooltip بر روی یک Object می شود.

خاصیت TooltipIcon : می توانید برای Tooltip مورد نظرتان یک آیکن تعیین کنید.

خاصیت TooltipTitle : عنوان Tooltip را مشخص می کند.

خاصیت UseAnimation : می توانید از انیمیشن برای نمایش Tooltip مورد نظر خودتان استفاده کنید.

خاصیت UseFading : زمان کار با برنامه، جهت حذف Tooltip به کار می رود.

کنترل Tooltip دارای دو رویداد فوق العاده کاربردی است، یکی از آنها رویداد Draw است

این رویداد موقعی رخ می دهد که Tooltip ترسیم شده و به نمایش در می آید. و رویداد

Popup موقعی رخ می دهد که Tooltip به صورت کامل در معرض دید قرار می گیرد.

کنترل TreeView

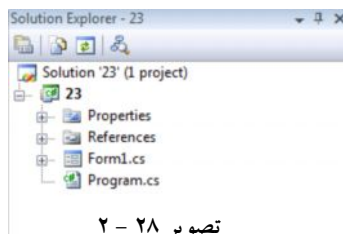
یکی از کنترل های پرکاربرد در محیط های برنامه نویسی کنترل TreeView است. کنترل

TreeViw برای نمایش ساختار های درختی فوق العاده مفید و کاربردی است این کنترل

ساختار های وابسته به هم را به صورت بسیار زیبایی در برنامه ها به تصویر می کشد این

کنترل در واقع، کنترل نمای درختی است نمونه بارز این کنترل پنجره Solution Explorer در

محیط Visual Studio می باشد (پنجره تصویر ۲۸ - ۲).



این کنترل را می توانید از پنجره Toolbox و بخش Common Controls به Form خودتان اضافه کنید.

برخی از خصوصیات مهم این کنترل در پنجره Properties به شرح زیر است :
خاصیت CheckBoxes : تعیین می کند که آیا هر آیتم در نمای درختی دارای جعبه علامت گذاری باشد یا خیر.

خاصیت FullRowSelect : تعیین می کند که آیا یک سطر به صورت کامل انتخاب شود یا خیر.

خاصیت HideSelection : باعث مخفی شدن انتخاب بعد از رفتن فوکوس می شود.
خاصیت HotTracking : در صورتی که این خاصیت با مقدار True مقدار دهی شود هر گره نسبت به حرکت ماوس عکس العمل نشان می دهد و به شکل پر رنگ به نمایش در می آید.
خاصیت Indent : مقدار تورفتگی گره های زیرین را نسبت به گره پدر برحسب Point مشخص می کند.

خاصیت Nodes : به کمک این خاصیت می توانید کلوکسیونی کامل از گره های خودتان را توسط پنجره TreeNode Editor مشخص کنید.

خاصیت Scrollable : به شما امکان Scroll کردن روی ساختار درختی را می دهد.
خاصیت LabelEdit : در صورتی که این خاصیت روی حالت True باشد به کاربر اجازه داده می شود که عنوان هر گره را به دلخواه تغییر دهد و امکان ویرایش گره ها در اختیار کاربر قرار داده می شود.

خاصیت ShowLines : باعث نمایش خطوط در بین آیتم های ساختار درختی می شود.
خاصیت ShowPlusMinus : تعیین می کند آیا گره هایی که دارای زیر مجموعه هستند با علامت + و - در زمان باز و بسته بودن زیر مجموعه ها نمایش داده شوند یا خیر.

خاصیت ShowRootLines : باعث نمایش خطوط مزبور به شاخه اصلی می شوند و در حین اجرا هم ویژگی هایی وجود دارند که به کمک آنها می توانید گره های انتخابی و علامت دار را تشخیص دهید.

برخی از متد های اختصاصی که در زمان کار با نمای درختی می توانید از آنها بهره ببرید به این شرح هستند :

متد Add : برای افزودن گره از متد Add مربوط به خاصیت Nodes می توانید استفاده کنید.
متد Remove : برای حذف یک گره می توان از متد Remove مربوط به خاصیت Nodes استفاده کرد.

متد Clear : برای حذف تمام گره های کنترل می توان از متد Clear مربوط به خاصیت Nodes استفاده کرد.

متد CollapseAll : باعث بسته شدن تمامی گره های موجود در نمای درختی می شود.
متد ExpandAll : باعث باز شدن تمامی گره های موجود روی نمای درختی می شود.
متد GetNodeAt : این امکان را به شما می دهد که یک گره خاص را در نقطه مشخص شده برگردانید.

متد GetNodeCount : می تواند تعداد گره های موجود در نمای درختی را به شما برگرداند.
برخی از رویداد های مهم کنترل TreeView به شرح زیر است :

رویداد AfterCheck : بعد از علامت دار شدن CheckBox مربوط به یک گره رخ می دهد.
رویداد AfterCollapse : بعد از بسته شدن زیر مجموعه های گره رخ می دهد.
رویداد AfterExpand : بعد از باز شدن زیر مجموعه های گره رخ می دهد.
رویداد AfterLabelEdit : بعد از ویرایش عنوان یک گره رخ می دهد.
رویداد AfterSelect : بعد از انتخاب شدن گره روی می دهد و رویداد پیش فرض کنترل TreeView است این رویداد بعد از انجام عمل انتخاب توسط کاربر بوقوع می پیوندد.
رویداد NodeMouseClick : وقتی که روی یک Node کلیک می شود این رویداد رخ می دهد.

وقتی با کنترل TreeView یک گره را بوجود می آورید در حقیقت شما به این ترتیب یک Object را در کلاس Tree بوجود می آورید و این Object به صورت منحصر دارای خواص، متد ها و رویداد هایی است که می توانید آنها را به کار ببرید.

مثال ۹-۲: با استفاده از کنترل TreeView برنامه ای بنویسید که عملیات های زیر را انجام

دهد :

الف : ساختن ریشه

ب : ساختن زیر گره یا فرزند برای ریشه ها

ج : آشنایی با دستور Remove

د : آشنایی با دستور Clear

حل : ابتدا یک پروژه جدید از نوع ویندوزی ساخته، سپس Form آن را مطابق پنجره تصویر ۲۹ - ۲ طراحی کنید. یک کنترل TextBox، یک کنترل TreeView، و ۴ کنترل Button و یک کنترل Label برای طراحی این Form استفاده کنید.



تصویر ۲۹ - ۲

خاصیت LabelEdit کنترل TreeView را برابر True قرار دهید، سپس دستورات زیر در هر یک از دکمه های ریشه، گره، حذف گره، حذف کلیه تایپ کنید.

```
private void Root_Click_1(object sender, EventArgs e)
{
    if (textBox1.Text != "")
        treeView1.Nodes.Add(textBox1.Text);
    else
        MessageBox.Show("لطفا یک نام را انتخاب کنید");
}

private void Node_Click(object sender, EventArgs e)
{
    TreeNode child;
    child = treeView1.SelectedNode;
    if (Node == null)
    {
        MessageBox.Show("لطفا یک گره را انتخاب کنید");
        return;
    }
    if (textBox1.Text == "")
        MessageBox.Show("لطفا یک نام را انتخاب کنید");
    child.Nodes.Add(textBox1.Text);
}
```

```
private void Remove_Click(object sender, EventArgs e)
{
    TreeNode child;
    child = treeView1.SelectedNode;
    if (Node == null)
    {
        MessageBox.Show("لطفا یک گره را انتخاب کنید");
        return;
    }
    child.Remove();
}

private void Clear_Click(object sender, EventArgs e)
{
    TreeNode child;
    child = treeView1.SelectedNode;
    if (Node == null)
    {
        MessageBox.Show("لطفا یک گره را انتخاب کنید");
        return;
    }
    child.Nodes.Clear();
}
```

حال کافی است که برنامه را اجرا کرده و نتیجه دستورات را به صورت واقعی بررسی کنید.

کنترل Timer

این کنترل برای زمانبندی کارها به کار می رود. به طور کلی وظیفه کنترل Timer مشخص کردن یک بازه زمانی خاص برای انجام یک رویداد یا اتفاق خاص است. به طور مثال تعیین می کنیم که در یک بازه زمانی خاص بایستی یک اتفاق رخ دهد یا در یک بازه زمانی خاص از وقوع اتفاقی جلوگیری شود. تمامی این عملیات ها با استفاده از کنترل Timer صورت می گیرد. این کنترل در زمان اجرا روی Form وجود ندارد و بایستی در زمان طراحی و کد نویسی اعمال مورد نظر را روی آن پیاده سازی کرد، این کنترل را می توانید از پنجره Toolbox و بخش All Windows Forms انتخاب کنید.

برخی از خاصیت های مهم کنترل Timer در پنجره Properties به شرح زیر است :

خاصیت InterVal : مدت زمانی را تعیین می کند که بایستی پس از سپری شدن رویداد Tick اجرا شود. مقدار پیش فرض این کنترل روی عدد ۱۰۰ تنظیم شده است. توجه داشته باشید که هر ۱۰۰ واحد برابر یک ثانیه است بطور مثال برای نشان دادن مقدار ۵ ثانیه بایستی این خاصیت برابر عدد ۵۰۰ باشد.

خاصیت Enable : این خاصیت فعال یا غیر فعال بودن کنترل Timer را تعیین می کند. به طور پیش فرض مقدار این کنترل روی False تنظیم شده است، جهت استفاده از این کنترل بایستی مقدار این خاصیت روی True قرار گیرد.

رویداد مهم این کنترل رویداد Tick است که پس از سپری شدن مدت زمانی که در خاصیت InterVal تعیین شده است رخ می دهد.

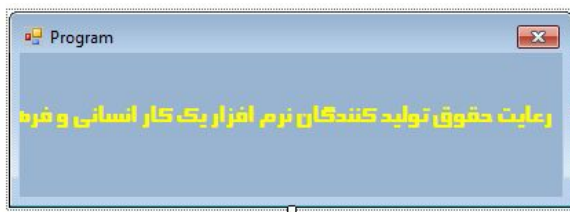
متد Start : شروع به محاسبه زمان تا زمان تعیین شده در خاصیت InterVal عمل می کند.

متد Stop : کار محاسبه زمان را خاتمه می دهد.

مثال ۱۰ - ۲: می خواهیم برنامه ای بنویسیم که متن درون یک کنترل Label را به صورت متحرک به کاربران برنامه نشان دهیم.

حل : ابتدا یک پروژه جدید ویندوزی ساخته و در Form آن یک کنترل Label و یک کنترل Timer به آن اضافه می کنیم. خاصیت Text کنترل Label را برابر مقدار (رعایت حقوق تولید کنندگان نرم افزار یک کار انسانی و فرهنگ است) قرار دهید. سعی کنید خاصیت BackColor کنترل Label را با خاصیت BackColor کنترل Form یکی انتخاب کنید و خاصیت ForeColor کنترل Label را نیز بسته به سلیقه خودتان انتخاب کنید (پنجره تصویر ۳۰ - ۲).

خاصیت Enabel کنترل Timer را برابر True قرار دهید و مقدار خاصیت InterVal را برابر ۵۰۰ قرار دهید.



تصویر ۳۰ - ۲

پس از اینکه تنظیمات گفته شده را انجام دادید بر روی کنترل Label دابل کلیک کنید و در رویداد Click آن دستورات زیر را بنویسید.

```
private void label1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}
```

دوباره به محیط طراحی Form برگشته و بر روی کنترل Time دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Left += 1;
}
```

این دستور تعیین می کند که کنترل Label از مبدا Left یک واحد به سمت جلو حرکت کند. حال برنامه را اجرا کرده و نتیجه را بررسی کنید.

کنترل NumericUpDown

این کنترل به شما امکان می دهد که به کمک آنها یک ساختار ساده جعبه متنی با کمک دکمه های افزایش و کاهش برای تعیین یک مقدار عددی ایجاد کنید.

برخی از خاصیت ها و رویداد های مهم این کنترل به شرح زیر است :

خاصیت DecimalPlaces : تعداد ارقام بعد از نقطه اعشار را تعیین می کند. البته نقطه اعشار نمایش داده نمی شود و در حالت پیش فرض برابر صفر است.

خاصیت Hexadecimal : در صورتی که این خاصیت روی حالت True تنظیم شود اطلاعات در مبنای ۱۶ نمایش داده می شود.

خاصیت Increment : اندازه مورد نظر را برای کاهش یا افزایش عدد در هر بار کلیک روی دکمه عای Up و Down تعیین می کند.

خاصیت InterceptArrowKey : تعیین می کند که آیا کلید های جهت دار مانند دکمه های Up و Down عمل کند یا خیر.

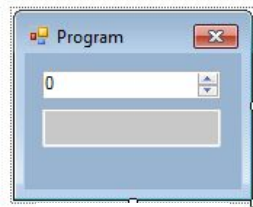
خاصیت Value : در این کنترل به جای خاصیت Text از خاصیت Value استفاده می کنید که به شما مقدار عدد موجود را بر می گرداند.

رویداد ValueChanged : مواقعی که مقدار خاصیت Value تغییر می کند این رویداد به وقوع می پیوندد.

مثال ۱۱ - ۲: برنامه ای بنویسید که عددی را خوانده و تشخیص دهد که این عدد جزء اعداد سری فیبوناچی است یا خیر.

در این برنامه از کنترل NumericUpDown برای خواندن عدد استفاده کنید.

حل : یک پروژه جدید ویندوزی ایجاد کرده و Form آن را مطابق پنجره تصویر ۳۱ - ۲ طراحی کنید. یک کنترل NumericUpDown برای خواندن مقادیر ورودی و یک کنترل TextBox برای نمایش خروجی استفاده کنید.



تصویر ۳۱ - ۲

بر روی کنترل NumericUpDown دابل کلیک کنید، سپس دستورات زیر را تایپ کنید.

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    int f1 = 1, f2 = 1, f3 = 0;
    if (numericUpDown1.Value == 1)
    {
        textBox1.Text="بلی";
        return;
    }
    while (f3 <= numericUpDown1.Value)
    {
        f3 = f1 + f2;
        if (numericUpDown1.Value == f3)
        {
            textBox1.Text = "بلی";
            return;
        }
        f1 = f2;
        f2 = f3;
    }
    textBox1.Text = "نه";
}
```

حال کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

کنترل PictureBox

از این کنترل برای نمایش تصاویر گرافیکی استفاده می شود. برخی از خواص و متد های مهم این کنترل به شرح زیر است :

خاصیت `BackgroundImage` : با استفاده از این خاصیت می توانید تصویر زمینه کنترل خود را تعیین کنید.

خاصیت `Image` : تصویری را تعیین می کند که کنترل `PictureBox` باید آن را نمایش دهد.

خاصیت `SizeMode` : با این خاصیت می توان اندازه تصویر را در کنترل `PictureBox` تعیین کرد. این خاصیت دارای مقادیر زیر است :

`Normal` : تصویر را در اندازه واقعی نمایش می دهد.

`StretchImage` : تصویر متناسب اندازه کنترل `PictureBox` قرار خواهد گرفت.

`AutoSize` : کنترل `PictureBox` به اندازه تصویر تغییر می یابد.

`CenterImage` : تصویر در مرکز کنترل `PictureBox` قرار می گیرد.

متد `Save` : برای ذخیره سازی تصاویر موجود در کنترل `PictureBox` بر روی دیسک سخت استفاده می شود و به صورت زیر به کار می رود :

```
PictureBox ControlName.Image.Save(filename);
```

کنترل `PictureBox` را می توانید از پنجره `Toolbox` و بخش `Common Controls` به `Form` خودتان اضافه کنید.

مثال ۱۲ - ۲ : می خواهیم برنامه ای بنویسیم که تصویر مورد نظر در درایو C کامپیوتر را با کنترل `PictureBox` نمایش دهیم؟

حل : ابتدا یک پروژه جدید از نوع ویندوزی ساخته، سپس `Form` آن را مطابق پنجره تصویر ۳۲ - ۲ طراحی کنید.

خاصیت `Width` و `Height` کنترل `PictureBox` را برابر ۴۰۰ قرار دهید. این دو خاصیت زیر مجموعه های خاصیت `Size` هستند.



تصویر ۳۲ - ۲

بر روی Form دابل کلیک کنید تا وارد رویداد Form1_Load شوید، سپس دستورات زیر را بنویسید.

```
private void Form1_Load(object sender, EventArgs e)
{
    pictureBox1.ImageLocation = ("C:\\1.jpg");
    pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
}
```

با استفاده از عملگر @ می توانیم به آدرس یک فایل دسترسی داشته باشیم.

کنترل ProgressBar

از این کنترل برای نمایش درصد پیشرفت کار ها استفاده می شود. نمونه این کنترل را می توانید در زمان کپی و چسباندن یک فایل دیده اید که میزان پیشرفت کار را به صورت درصد نشان می دهد.

برخی از خاصیت ها و متد های مهم این کنترل عبارتند از :

خاصیت Minimum : کمترین مقدار کنترل را مشخص می کند.

خاصیت Maximum : بیشترین مقدار کنترل را مشخص می کند.

خاصیت Step : گام افزایش پیشرفت کار را تعیین می کند

خاصیت Value : مقدار فعلی پیشرفت کار را تعیین می کند.

خاصیت Increment : موجب افزایش مقدار خاصیت Value می شود.

مثال ۱۳ - ۲: برنامه ای بنویسید که در آن کاربرد کنترل ProgressBar را نشان دهد؟

حل : ابتدا یک پروژه ویندوزی جدید ساخته، سپس Form آن را مشابه پنجره تصویر ۳۳ - ۲ طراحی کنید. بر روی Form یک کنترل Timer، یک کنترل Button، یک کنترل TextBox و یک کنترل Label قرار دهید.



تصویر ۳۳ - ۲

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    timer1.Interval = 10;
    timer1.Enabled=true;
}
```

حال بر روی کنترل Timer۱ دابل کلیک کنید تا وارد رویداد Tick کنترل Timer۱ شوید، سپس دستورات زیر را بنویسید.

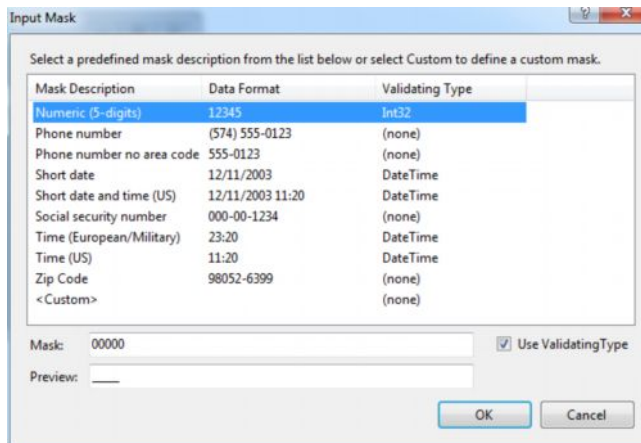
```
private void timer1_Tick(object sender, EventArgs e)
{
    progressBar1.Value += 1;
    textBox1.Text = Convert.ToString(progressBar1.Value) + "%";

    if (progressBar1.Value == 100)
    {
        progressBar1.Value = 0;
        timer1.Enabled = false;
        MessageBox.Show("عملیات با موفقیت تمام شد");
    }
}
```

کنترل MaskedTextBox

این کنترل شبیه کنترل TextBox معمولی است، ولی اجازه ورود هر نوع داده ای را به کاربر نمی دهد. با استفاده از خاصیت Mask می توان نوع داده های ورودی را کنترل کرد. به طور مثال با تنظیم روی حالت نوع داده ای عددی، فقط کاربر می تواند نوع عددی را وارد کند و یا تنظیم این خاصیت روی حالت رشته ای، فقط کاربر اجازه ورود نوع داده ای رشته ای را خواهد داشت. یک نمونه از این کنترل را به Form اضافه کنید و در پنجره Properties روی خاصیت Mask کلیک کنید تا پنجره Input Mask باز شود (پنجره تصویر ۳۴ - ۲).

همانطور که در پنجره تصویر ۳۴ - ۲ مشاهده می کنید انواع نوع های داده ای که کاربر می تواند به عنوان داده وارد کند در این پنجره مشاهده می شود. به طور مثال ما حالت Numeric را روی حالت (digit - ۵) قرار داده ایم، یعنی فقط نوع داده عددی و با حداقل ۵ عدد را می تواند به عنوان ورودی تایپ کند. در زمان اجرا این کنترل فقط نوع داده عددی را قبول خواهد کرد. اگر کاربر روی حروف در صفحه کلید، کلیدی را فشار دهد بی اثر خواهد بود و کنترل مقداری را قبول نخواهد کرد.



تصویر ۳۴ - ۲

کنترل HelpProvider

کنترل HelpProvider یک پنجره کمکی به صورت آنلاین می باشد. برخی از متدهای مفید HelpProvider به صورت زیر است :

متد SetShowHelp : مشخص می کند که آیا راهنمایی برای کنترل مشخصی ارائه شده است .

متد SetHelpString : مشخص کننده رشته متنی راهنما در ارتباط با کنترل های مشخص می باشد.

متد SetHelpNavigator : از دستورات کمکی برای زمانی که فایل های راهنما برای کنترل های مشخص بازیابی می شوند، استفاده می کند.

متد SetHelpKeyword : از واژه کلیدی برای بازیابی فایل های راهنما، در مواقعی که کاربر نیاز دارد، استفاده می کند.

کنترل HelpProvider را می توانید از پنجره Toolbox و بخش All Windows Forms به پروژه های خودتان اضافه کنید. به دستورات زیر دقت کنید.

```
private void Form1_Load(object sender, EventArgs e)
{
    helpProvider1.SetHelpString(textBox1, "Please Enter Your Name :");
}
```

حال اگر برنامه را اجرا کرده و در کنترل textbox در حالت نوشتن قرار گیرید، با زدن کلید F۱ از صفحه کلید برنامه اجرا خواهد شد.

کنترل ErrorProvider

در یک مثال کلی ما نحوه استفاده از این کنترل را توضیح خواهیم داد. و خواهیم دید که چگونه این کنترل یک پیغام خطا مبنی بر عدم تایید کاربر، با توجه به اطلاعات وارد شده را در برنامه نشان می دهد.

یک کنترل ErrorProvider از جعبه ابزار به پروژه خود اضافه کرده و در رویداد TextChanged یک کنترل textbox دستورات زیر را بنویسید.

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    // Determine if the TextBox has a digit character.
    string text = textBox1.Text;
    bool hasDigit = false;
    foreach (char letter in text)
    {
        if (char.IsDigit(letter))
        {
            hasDigit = true;
            break;
        }
    }
    // Call SetError or Clear on the ErrorProvider.
    if (!hasDigit)
    {
        errorProvider1.SetError(textBox1, "Needs to contain a digit");
    }
    else
    {
        errorProvider1.Clear();
    }
}
```

حال اگر برنامه را اجرا کنید یک آیکن خطا نمایش داده می شود و پیغام (Needs to contain a digit) نمایش داده می شود.

با توجه به اینکه هنگام اجرای برنامه و در صورت نمایش خطا برای یک کنترل مخصوص ، کنترل Error Provider با استفاده از آرگومان اول، متد SetError را راه اندازی می کند. و با استفاده از آرگومان دوم هنگامی که نشانگر ماوس بر روی پیغام خطا قرار می گیرد، یک فایل متنی کوچک به عنوان Tooltip را نشان می دهد. یعنی زمانی که کاربر یک مقدار اشتباه را وارد نماید، تصویر یک خطای چشمک زن در Form برای کنترل نمایان می شود و سپس با قرار گرفتن ماوس بر روی پیغام خطا، کادر متنی کوچک تحت عنوان یک Tooltip نمایش داده می شود.

با استفاده از کنترل `ErrorProvider` می توان چندین پیغام را در یک `Form` مشاهده کرد. همچنین می توان خطاهایی که هنگام تنظیم تاریخ رخ می دهند را دید و یا می توان `Blink Style` و `Blink Rate` مربوط به کنترل `ErrorProvider` را تنظیم نمود. و یا حتی می توان از یک تصویر دلخواه به جای تصویر پیش فرض پیغام خطا استفاده کرد.

کنترل `TrackBar`

این کنترل از یک نوار قابل جابجایی تشکیل شده است که کاربر به راحتی می تواند آن را بین دو نقطه جابجا کند.

برخی از خواص مهم این کنترل عبارتند از :

خاصیت `Orientation` : نحوه قرار گیری کنترل را به صورت عمودی یا افقی تعیین می کند.
خاصیت `LargeChanged` : تعیین می کند با حرکت نوار جابجایی با فاصله زیاد تا چه اندازه از مقدار خاصیت `Value` کم یا زیاد شود.
خاصیت `SmallChanged` : تعیین می کند با حرکت نوار جابجایی با فاصله کم تا چه اندازه از مقدار خاصیت `Value` کم یا زیاد شود.
خاصیت `TickStyle` : شیوه خط های `Tick` را کنترل می کند.
متد `SetRange` : خواص `Maximum` و `Minimum` کنترل `TrackBar` را تعیین می کند و به صورت زیر استفاده می شود.

`ControlName.SetRange` (کمترین مقدار و بیشترین مقدار);

یک کنترل `TrackBar` از جعبه ابزار به پروژه خود اضافه کنید، سپس در رویداد `Scroll` آن دستورات زیر را بنویسید.

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    label1.Text = "Value: " + trackBar1.Value;
}
```

دستورات بالا مقدار اولیه کنترل `TrackBar` را نشان می دهد که با کشیدن دکمه لغزان، این مقدار افزایش می یابد.

کنترل Process

این کنترل برای فراخوانی یا اجرای برنامه های گوناگون به کار می رود. این کنترل در جعبه ابزار Toolbox و در قسمت Components قرار دارد. این کنترل دارای خصوصیتی به نام FileName می باشد، نام برنامه ای را که می خواهیم اجرا شود را با پسوند اجرایی آن در این خاصیت وارد می کنیم.

برخی از رویدادهای کنترل Process عبارتند از :

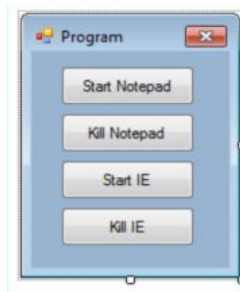
رویداد Start : باعث اجرای کنترل Process شده که در نهایت باعث اجرای برنامه مورد نظر خواهد شد.

رویداد Exited : با این رویداد می توان تعیین کرد که با بسته شدن کنترل Process چه اتفاقی روی دهد.

رویداد Kill : این رویداد کنترل Process را می بندد، و باعث بسته شدن برنامه ای که از طریق این کنترل باز شده می شود.

مثال ۱۴ - ۲: می خواهیم با استفاده از کنترل Process، برنامه ای بنویسیم که بتوان برنامه های IE و Notepad را با آن اجرا کرد.

حل : ابتدا یک پروژه از نوع ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۳۵ - ۲ طراحی کنید.



تصویر ۳۵ - ۲

دو کنترل Process به برنامه اضافه کنید.

کنترل process1 را انتخاب کرده و بر روی پنجره Properties کلیک کنید، سپس خاصیت StartInfo را باز کرده و در خاصیت FileName عبارت notepad.exe را وارد می کنیم.

و خاصیت WorkingDirectory را برابر با محلی که فایل notepad.exe قرار دارد مقدار دهی می کنیم، یعنی (%system%\system۳۲).

بر روی دکمه Start Notepad دابل کلیک کرده و در رویداد Click آن، دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    process1.EnableRaisingEvents = true;
    process1.Start();
}
```

بر روی دکمه Kill Notepad دابل کلیک کرده و در رویداد Click آن، دستورات زیر را بنویسید.

```
private void button2_Click(object sender, EventArgs e)
{
    process1.Kill();
}
```

حال در رویداد Exited مربوط به کنترل process۱ دستورات زیر را بنویسید.

```
private void process1_Exited(object sender, EventArgs e)
{
    MessageBox.Show("Notepad Process Terminated", "Process Terminated");
}
```

با دستورات بالا، موقع بستن برنامه Notepad، پیغام مورد نظر نمایش داده خواهد شد. برای اینکه process۲ را در زمان اجرا پیکربندی کنیم، بایستی ویژگی FileName را که موجب اجرای IE در زمان اجرا می شود را بدست آورد. بر روی دکمه Start IE دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.

```
private void button3_Click(object sender, EventArgs e)
{
    string programfilepath = System.Environment.GetFolderPath
        (Environment.SpecialFolder.ProgramFiles);
    string iepath = System.IO.Path.Combine(programfilepath,
        @"Internet Explorer \ IExplorer.exe");
    process2.EnableRaisingEvents = true;
    process2.StartInfo.FileName = iepath;
    process2.Start();
}
```

حال بر روی دکمه Kill IE دابل کلیک کرده و در رویداد Click آن، دستورات زیر را بنویسید.

```
private void button4_Click(object sender, EventArgs e)
{
    process2.Kill();
}
```

در رویداد Exited کنترل process2 دستورات زیر را بنویسید.

```
private void process2_Exited(object sender, EventArgs e)
{
    MessageBox.Show("IE Process Terminated", "Process Terminated");
}
```

حال کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

کنترل BackgroundWorker

کنترل BackgroundWorker برای ایجاد برنامه های چند نخه به کار می رود. این کنترل به برنامه نویس کمک می کند تا از پرداختن به جزئیات رها شود. با این کنترل فرآیند های طولانی مدت را بدون اینکه تاثیری بر روی رابط کاربری داشته باشد می توان اجرا کرد. این کنترل را می توان از پنجره Toolbox و بخش Components به برنامه ها اضافه کرد.

برخی از خواص و متد های کنترل BackgroundWorker به صورت زیر است :

خاصیت CancellationPending : در صورتی که کاربردی لغو یا حذف شده باشد، با استفاده از این کنترل قابل مشاهده است.

خاصیت IsBusy : مشخص می کند که BackgroundWorker در حال اجرای عملیات می باشد.

خاصیت WorkerReportsProgress : مشخص کننده گزارش های بروز شده برای کنترل BackgroundWorker می باشد.

خاصیت WorkerSupportsCancellation : توسط این خصوصیت می توان به حذفیات غیرهمزمان کنترل BackgroundWorker دست یافت.

دو متد RunWorkerAsync و CancelAsync (برای شروع و خاتمه یک وضعیت) کاربرد دارد.

برای اینکه رویدادی برای BackgroundWorker اتفاق بیفتد، بایستی متد RunWorkerAsync فراخوانی شده باشد.

مثال ۱۵-۲: در برنامه زیر می خواهیم نحوه عملکرد کنترل BackgroundWorker را نشان دهیم.

حل : ابتدا یک پروژه جدید ویندوزی ایجاد کرده و Form آن را مطابق پنجره تصویر ۳۶ - ۲ طراحی کنید.



تصویر ۳۶ - ۲

یک کنترل BackgroundWorker، یک کنترل Button و یک کنترل ProgressBar به Form خودتان اضافه کنید.

بر روی دکمه Click در پنجره تصویر ۳۶ - ۲ دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    backgroundWorker1.RunWorkerAsync();
}
```

حال در رویداد DoWork کنترل backgroundWorker\ بدستورات زیر را بنویسید.

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 1500000; j++) ;
        backgroundWorker1.ReportProgress(i);
    }
}
```

دوباره در رویداد RunWorkCompleted دستورات زیر را بنویسید.

```
private void backgroundWorker1_RunWorkerCompleted(object sender,
    RunWorkerCompletedEventArgs e)
{
    MessageBox.Show("Finished");
    progressBar1.Value = 0;
}
```

در صورت اتمام فرآیند، رویداد RunWorkCompleted رخ می دهد و با یک پیام به کاربر اعلام می کند.

رویداد ProgressChanged در صورت فراخوانی متد ReportProgress مربوط به کنترل

backgroundWorker۱ رخ می دهد. در رویداد ProgressChanged دستورات زیر را بنویسید.

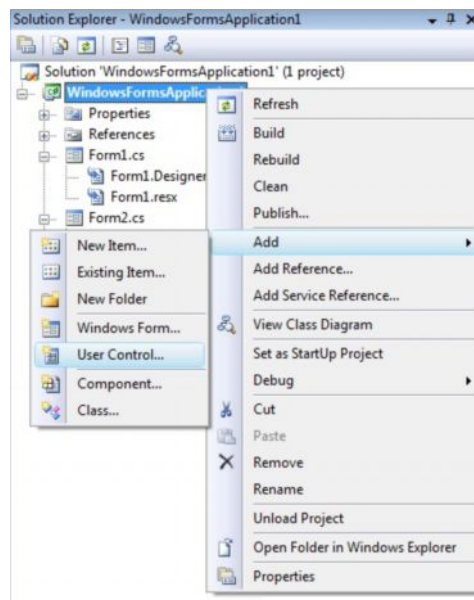
```
private void backgroundWorker1_ProgressChanged(object sender,
    ProgressChangedEventArgs e)
{
    progressBar1.Value = e.ProgressPercentage;
}
```

خاصیت WorkReportProgress کنترل backgroundWorker۱ را برابر با مقدار True قرار دهید.

حال برنامه را اجرا کرده و نتیجه را مشاهده کنید.

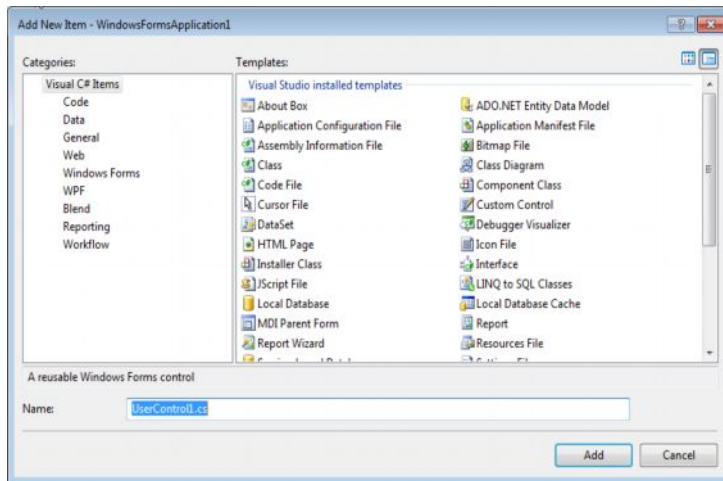
کار با User Control ها

می توان User Control ها را ایجاد و درون آنها اعمال کنترل سنجی و غیره را انجام داده و به هر شکلی که دوست داریم از آنها استفاده کنیم. برای اضافه کردن یک User Control در Solution Explorer روی اسم پروژه راست کلیک کرده گزینه Add و سپس User Control را کلیک کنید (پنجره تصویر ۳۷ - ۲).



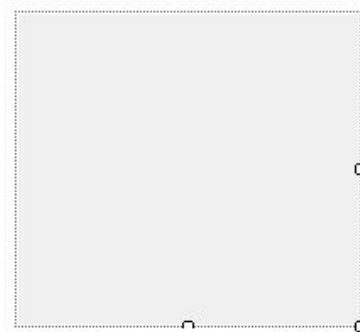
تصویر ۳۷ - ۲

پنجره Add New Item باز می شود (پنجره تصویر ۳۸ - ۲).



تصویر ۳۸ - ۲

در بخش Name در پنجره Add New Item می توانید عنوان User Control خود را وارد کنید. بر روی دکمه Add کلیک کنید تا پنجره زیر نمایش داده شود.



تصویر ۳۹ - ۲

همانطور که در پنجره تصویر ۳۹ - ۲ مشاهده می کنید یک Form خالی بدون نوار عنوان و کادر های کنترلی ظاهر می شود. می توانید روی این Form خالی انواع کنترل های خود را قرار داده و طراحی کنید. نرم افزار Visual Studio به طور مستقیم User Control های طراحی شده توسط برنامه نویسان را به پنجره Toolbox و بخش پایین این پنجره اضافه می کند. از این پس می توانیم این User Control را بر روی Form های خودمان درگ کرده و استفاده کنیم.

معرفی چند کنترل و ترفند کاربردی

در این بخش قصد داریم چند کنترل کاربردی را جهت کار با انواع کنترل ها در محیط ویژوال استودیو بررسی کنیم.

ابزار تلفظ متن : در محیط نرم افزار Visual Studio در قسمت خالی پنجره Toolbox راست کلیک کرده و گزینه Choose Items... را کلیک کنید تا پنجره Choose Toolbox Items ظاهر شود، در این پنجره روی تب COM Components کلیک کنید تا ظاهر شود. در این تب گزینه Text ToSpeech Class را تیک زده و روی دکمه OK کلیک کنید، تا این کنترل به لیست کنترل های پنجره Toolbox اضافه شود. این کنترل به صورت خودکار در لیست کنترل های بخش Dialogs ظاهر خواهد شد. این کنترل شبیه یک دهان است که می توانید با غیر فعال کردن خاصیت Visible آن را غیر قابل مشاهده کنید. جهت استفاده از این کنترل کافی است که آن را روی Form درگ کنید.

دستور زیر موجب می شود تا متنی که درون یک کنترل TextBox قرار دارد تلفظ شود.

```
private void button1_Click(object sender, EventArgs e)
{
    axTextToSpeech1.Speak(textBox1.Text.Trim());
}
```

ابزار باز کردن فایل PDF : دوباره در محیط نرم افزار Visual Studio در قسمت خالی پنجره Toolbox راست کلیک کرده و گزینه Choose Items... را کلیک کنید تا پنجره Choose Toolbox Items ظاهر شود، در این پنجره روی تب COM Components کلیک کنید تا ظاهر شود.

در این تب گزینه Adobe PDF Reader را تیک زده و روی دکمه OK کلیک کنید، تا این کنترل به لیست کنترل های پنجره Toolbox اضافه شود. این کنترل به صورت خودکار در لیست کنترل های بخش Dialogs ظاهر خواهد شد. جهت استفاده از این کنترل کافی است که آن را روی Form درگ کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.Filter = "pdf files (*.pdf) |*.pdf";
    dlg.ShowDialog();
    if (dlg.FileName != null)
    {
        axAcroPDF1.LoadFile(dlg.FileName);
    }
}
```

دستورات بالا را که در رویداد Click یک کنترل Button نوشته شده است در صورت اجرای برنامه، کادری ظاهر خواهد شد و از شما مسیر فایل PDF را خواهد خواست که با انتخاب فایل PDF خودتان می توانید آن را مشاهده کنید.

ابزار اجرای فایل های SWF: مسیر گفته شده در پنجره Toolbox را دنبال کرده ولی این بار در تب COM Components گزینه Shockwave Flash Object را تیک زده و روی OK کلیک کنید، تا این کنترل نیز به لیست کنترل های پنجره Toolbox اضافه شود. این کنترل قابلیت پخش فایل هایی با پسوند swf را داراست. جهت استفاده از این کنترل کافی است که همانند سایر کنترل ها موجود در پنجره Toolbox یک نمونه از آن را به Form درگ کنید. دستور زیر موجب اجرای فایل main.swf می شود. این دستور را می توانید در رویداد Click یک کنترل Button یا رویداد Form_Load بنویسید.

```
axShockwaveFlash1.Movie = Application.StartupPath + @"flash\main.swf";
```

پخش صدا هنگام بار گذاری Form: برای اجرای یک آهنگ کوتاه در برنامه های خود، بطور مثال زمان ورود و یا خروج کاربر به برنامه، ابتدا فایل صوتی را کنار فایل اجرایی کپی کرده، سپس کتابخانه زیر را اضافه کنید:

```
using System.Media;
```

حال کافی است که دستورات زیر را در رویداد مورد نظر خود مانند Form_Load بنویسید.

```
SoundPlayer mySound = new SoundPlayer("music1.wav"); //music
mySound.Play(); //play music
```

توجه کنید که فایل صوتی خود را در پوشه Debug و کنار فایل اجرایی برنامه، در پوشه ای که پروژه خود را ساخته اید، ذخیره کنید.

محدود کردن کنترل TextBox به پذیرش اعداد ۰ تا ۹ : کافی است که در رویداد KeyPress مربوط به کنترل TextBox دستورات زیر را بنویسید. تا از ورود کاراکترهای رشته ای داخل کنترل TextBox جلوگیری شود.

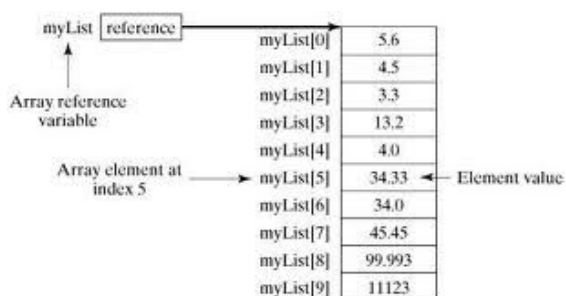
```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar))
    {
        e.Handled = true;
    }
}
```

باز کردن یک سایت در مرورگر IE : دستور زیر موجب باز شدن سایت irib.ir در مرورگر IE می شود.

```
Application.Restart();
```

Restart کردن برنامه : دستورات زیر موجب بسته شدن و اجرای مجدد برنامه می شود.

```
System.Diagnostics.Process.Start
("iexplore.exe", "www.irib.ir");
```



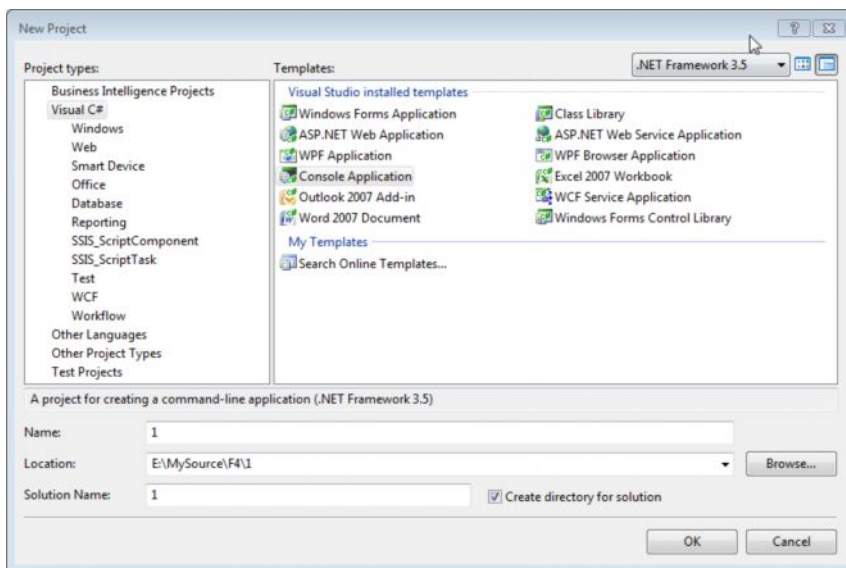
فصل چهارم

بررسی توابع ریاضی، رشته ای و آرایه ها در زبان C#.NET

روش های برنامه نویسی در زبان C#.NET

اگر به یاد داشته باشید در فصل اول گفتیم که نرم افزار Visual Studio دارای محیط های متعددی جهت برنامه نویسی با زبان های برنامه نویسی تحت خود ارائه کرده است. تا به این فصل ما تمام عملیات و پروژه های خود را در محیط Windows انجام می دادیم از این به بعد می خواهیم با محیط جدیدی تحت عنوان Console آشنا شویم در این محیط برخلاف محیط Windows برای نوشتن برنامه ها، از کنترل استفاده نمی شود، بلکه دستوراتی برای انجام کار ها نوشته می شوند. متد های Read و ReadLine برای خواندن اطلاعات از صفحه کلید و متد های Write و WriteLine برای نوشتن اطلاعات در صفحه نمایش به کار می روند.

جهت ساخت یک پروژه تحت کنسول کافی است که نرم افزار Visual Studio را اجرا کرده و از پنجره New Project گزینه Console Application را اجرا کلیک کنید (پنجره تصویر ۱ - ۴).



تصویر ۱ - ۴

بعد از انتخاب گزینه Console Application بر روی دکمه OK کلیک کنید تا بخش کد نویسی آن باز شود.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}

```

همانطور که در بخش دستورات این محیط می توانید مشاهده کنید، تعدادی دستور `using` برای استفاده از فضا های نام `System` و زیر مجموعه های آن قرار گرفته است. فضای نامی `namespace _1` یعنی با پروژه، ایجاد شده است. سپس کلاسی به نام `Program` ایجاد شده است که دارای متدی به نام `Main()` است تمام برنامه ها در `C#` با یک کلاس شروع می شوند. متد `Main()` شروع کننده اجرای برنامه در `C#` است. واژه `static` در متد `Main()` مشخص می کند که بدون نمونه سازی از کلاس می توان آن را اجرا کرد. عبارت `void` تعیین می کند که متد `Main()` هیچ مقداری را بر نمی گرداند. متد `Main()` آرایه ای از رشته هاست که پارامتر های وارد شده از طریق خط فرمان را در خود ذخیره می کند. پارامتر اول خط فرمان در `args[0]`، پارامتر دوم در `args[1]` و ... قرار می گیرد.

چاپ اطلاعات در `Console` : برای چاپ اطلاعات در کنسول از دو دستور `Write()` و `WriteLine()` استفاده می شود. این دستورات متعلق به کلاس `Console` هستند که در `C#` وجود دارند. خروجی این دستورات می تواند دارای فرمت خاصی باشد که با کاراکتر های خاصی مشخص می شوند این کاراکتر ها را کاراکتر های کنترلی می گویند که در جدول ۱ – ۴ مشخص شده اند.

کاراکتر	شرح
\n	مکان نما را به ابتدای خط جدید می برد.
\t	مکان نما را به مکان توقف بعدی می برد.
\r	مکان نما را به ابتدای سطر فعلی می برد.
\\	کاراکتر \ را در متن قرار می دهد.
\"	کاراکتر " را در متن قرار می دهد.

جدول ۱ - ۴

خروجی برنامه های تحت Console در محیط Command Prompt قابل مشاهده است. توجه داشته باشید که ساختار تمام دستورات زبان C# در تمام محیط ها یکسان است و تفاوتی از لحاظ اجرایی و ساختاری ندارد.

مثال زیر، یک برنامه ساده را نشان می دهد که اطلاعاتی را در Console می نویسد :

```
namespace _1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 8, y = 5, a, b;
            a = x++;
            b = ++y;
            System.Console.WriteLine(a);
            System.Console.WriteLine(b);
        }
    }
}
```

حال کافی است که بر روی دکمه Run در نوار ابزار برنامه کلیک کنید تا برنامه اجرا شود. جهت مشاهده خروجی کافی است به پوشه ای که پروژه را در آن ایجاد و ذخیره کرده اید مراجعه کنید.

همانطور که در پنجره تصویر ۲ - ۴ مشاهده می کنید یک فایل که حاوی سورس برنامه و یک پوشه که شامل اجزای پروژه است در اختیار کاربر قرار می گیرد. این فایل و پوشه هم نام با عنوان پروژه که در ابتدای ایجاد آن وارد کرده ایم تولید می شوند. بر روی پوشه دابل کلیک کنید تا وارد آن شوید.



تصویر ۲ - ۴

	bin	5/21/2012 2:05 AM	File folder	
	obj	5/21/2012 2:05 AM	File folder	
	Properties	5/21/2012 2:05 AM	File folder	
	1.csproj	5/21/2012 2:05 AM	Visual C# Project f...	3 KB
	Program.cs	5/21/2012 2:52 AM	Visual C# Source f...	1 KB

تصویر ۳ - ۴

همانطور که در پنجره تصویر ۳ - ۴ مشاهده می کنید اجزای پروژه نمایش داده می شود. کافی است که بر روی پوشه bin دابل کلیک کنید تا محتویات آن باز شود. محتویات پوشه bin نیز یک پوشه دیگر به نام Debug است با کلیک بر روی این پوشه می توانید فایل خروجی برنامه خود را مشاهده کنید.

	1.exe	5/21/2012 2:53 AM	Application	5 KB
	1.pdb	5/21/2012 2:53 AM	Program Debug D...	12 KB
	1.vshost.exe	5/21/2012 2:54 AM	Application	14 KB

تصویر ۴ - ۴

با کلیک بر روی فایل 1.exe می توانید خروجی برنامه خود را مشاهده کنید. در صورتی که در محیط ویندوز مخصوصاً ویندوز های Vista & Seven اجرا کنید این فایل سریع اجرا و بسته می شود. برای مشاهده بهتر خروجی برنامه خود می توانید از محیط Command Prompt استفاده کنید و این مسیر را در این محیط طی کنید تا بتوانید خروجی برنامه خود را مشاهده کنید. یا از منوی Debug گزینه Start Without Debugging استفاده کنید.

نکته!

فایل exe برنامه های نوشته شده تحت محیط ویندوز نیز در مسیر ذکر شده قرار دارند. تنها تفاوتی که بین دستورات Write و Writeline وجود دارد این است که دستور Writeline علاوه بر نمایش خروجی، سطر جاری را رد می کند و دستور چاپ بعدی می تواند در سطر بعدی عمل کند

خواندن اطلاعات از Console

با استفاده از دو دستور Read و ReadLine می توان اطلاعاتی را به صورت رشته ای از ورودی خواند. اطلاعاتی که توسط این متد خوانده می شود، در یک متغیر قرار می گیرد. به دستورات زیر دقت کنید :

```
string str;
str = System.Console.ReadLine();
```

دستور اول، متغیر str را تعریف می کند و دستور دوم رشته ای را از ورودی خوانده در str قرار می دهد. دستورات Read و ReadLine می توانند اعداد را نیز بخوانند، ولی چون اعداد را به صورت رشته ای دریافت می کنند بایستی پس از دریافت آن را به عدد تبدیل کرد. تبدیل رشته به عدد با متد Parse انجام می شود. دستورات زیر را ببینید :

```
int max;
max = int.Parse(System.Console.ReadLine());
```

دستور اول، متغیر str را تعریف می کند و دستور دوم رشته ای را از ورودی خوانده در str قرار می دهد. دستورات Read و ReadLine می توانند اعداد را نیز بخوانند، ولی چون اعداد را به صورت رشته ای دریافت می کنند بایستی پس از دریافت آن را به عدد تبدیل کرد. تبدیل رشته به عدد با متد Parse انجام می شود. دستورات زیر را ببینید :

```
int max;
max = int.Parse(System.Console.ReadLine());
```

مثال ۱-۴: برنامه ای تحت محیط Console بنویسید که دو عدد را از کاربر دریافت کرده و عملیات جمع و ضرب را بر روی این دو عدد انجام دهد؟

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و سپس دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    int a, b, sum, mul;
    Console.Write("Please Enter Number1 : ");
    a = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter number2 :");
    b = Convert.ToInt32(Console.ReadLine());
    sum = a + b;
    mul = a * b;
    Console.WriteLine("{0}{1}\n{2}{3}", "Sum = ", sum, "Mul = ", mul);
}
```

به خط آخر برنامه بالا یعنی دستور WriteLine توجه کنید، این دستور دارای ۵ پارامتر است و حاصل جمع و ضرب را به خروجی می برد. پارامتر اول که با "{۰}\n{۲}{۳}" مشخص می شود رشته ای است که فرمت خروجی را مشخص می کند. {۰} مشخص می کند که اولین پارامتر (غیر از این پارامتر که رشته فرمت است) باید به خروجی برود که "Sum =" است. {۱} مشخص می کند که پارامتر دوم باید به خروجی برود که sum است. مقدار \n مشخص می کند که پس از چاپ پارامتر دوم، سطر فعلی باید رد شود. مقدار {۲} مشخص می کند که پارامتر سوم یعنی "Mul =" باید به خروجی برود و مقدار {۳} مشخص می کند که پارامتر چهارم یعنی mul باید چاپ شود. حال برای اجرا کافی است که از منوی Debug گزینه Start Without Debugging را کلیک کنید تا خروجی برنامه را به راحتی مشاهده کنید و نتیجه را بررسی کنید.

مثال ۲ - ۴: برنامه ای بنویسید که یک عدد را از کاربر گرفته و اعلام کند که عدد مزبور یک عدد تام است یا خیر؟

تعریف: عدد کامل (تام) عددی است که با مجموع مقسوم علیه های خود، به جز خودش، برابر باشد. از معروفترین آنها $6=1+2+3$ و $28=1+2+4+7+14$ هستند.

حل: ابتدا یک پروژه جدید از نوع Console ایجاد کرده و سپس دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    Console.WriteLine("Please Enter a Number:");
    int n = Convert.ToInt32(Console.ReadLine());
    int sum = 0;
    for (int i = 1; i < n; i++)
    {
        if (n % i == 0)
            sum += i;
    }
    if (sum == n)
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
    Console.ReadLine();
}
```

مثال ۳ - ۴: برنامه ای بنویسید که جدول ضرب $10 * 10$ را در خروجی چاپ کند؟
 حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده، سپس دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    int i = 1;
    while (i <= 10)
    {
        int j = 1;
        string line = "";
        while (j <= 10)
        {
            line += (i * j).ToString() + " ";
            j++;
        }
        Console.WriteLine(line);
        i++;
    }
}
```

توابع ریاضی در زبان C#.NET

توابع جهت انجام اعمال ریاضی، کاراکتری، رشته ای و غیره در تمام زبان های برنامه نویسی وجود دارد. در اعمال ریاضی می توان تولید اعداد تصادفی، محاسبه قدر مطلق اعداد و لگاریتم و سایر اعمال محاسباتی ریاضی را انجام داد. الگوی اکثر توابع در فایل Math.h قرار دارد.

تابع `Abs()` : این تابع برای محاسبه قدر مطلق اعداد صحیح مورد استفاده قرار می گیرد. اگر آرگومان این تابع یک عدد منفی باشد نتیجه حاصل از تابع یک عدد مثبت است و اگر آرگومان تابع، مثبت یا صفر باشد نتیجه، یک عدد مثبت یا صفر خواهد بود. تابع `Abs()` با انواع داده های عددی `single` , `int` , `double` , `decimal` قابل استفاده است. برنامه زیر نحوه استفاده از تابع `Abs()` را به خوبی نشان می دهد.

```
static void Main(string[] args)
{
    int value1 = -1000;
    int value2 = 20;
    int abs1 = Math.Abs(value1);
    int abs2 = Math.Abs(value2);
    Console.WriteLine(value1);
    Console.WriteLine(abs1);
    Console.WriteLine(value2);
    Console.WriteLine(abs2);
    double value3 = -100.123;
    double value4 = 20.20;
    double abs3 = Math.Abs(value3);
    double abs4 = Math.Abs(value4);
    Console.WriteLine(value3);
    Console.WriteLine(abs3);
    Console.WriteLine(value4);
    Console.WriteLine(abs4);
}
```

تابع $\text{Acos}()$: تابع $\text{Acos}()$ برای محاسبه آرک کسینوس یک عدد مورد استفاده قرار می گیرد. مقادیری را که این تابع قبول می کند در بازه -1 تا 1 است و نتیجه حاصل به صورت رادیان و در بازه صفر تا عدد P است.

تابع $\text{Asin}()$: تابع $\text{Asin}()$ برای محاسبه آرک سینوس یک عدد به کار می رود. ورودی این تابع در بازه -1 تا 1 است و نتیجه حاصل به صورت رادیان و در بازه $P/2$ و $P/2$ است. تابع $\text{Atan}()$: تابع $\text{Atan}()$ برای محاسبه آرک تانژانت یک عدد به کار می رود. مقادیری که توسط این تابع محاسبه می شوند به صورت رادیان و در بازه $P/2$ و $P/2$ است.

تابع $\text{Cos}()$: تابع $\text{Cos}()$ برای محاسبه کسینوس یک زاویه بر حسب رادیان به کار می رود. تابع $\text{Sin}()$: برای محاسبه سینوس یک زاویه بر حسب رادیان به کار می رود.

تابع $\text{Tan}()$: برای محاسبه تانژانت یک زاویه که بر حسب رادیان می باشد به کار می رود.

تابع $\text{Cosh}()$: تابع $\text{Cosh}()$ برای محاسبه کسینوس هیپربولیک یک عدد به کار می رود.

تابع $\text{Sinh}()$: برای محاسبه سینوس هیپربولیک یک زاویه بر حسب رادیان به کار می رود.

تابع $\text{Tanh}()$: تابع $\text{Tanh}()$ برای محاسبه تانژانت هیپربولیک یک زاویه بر حسب رادیان به کار می رود.

در برنامه زیر می توانید کاربرد این توابع را مشاهده کنید.

```
static void Main(string[] args)
{
    Console.WriteLine(Math.Acos(0)); // Inverse cosine
    Console.WriteLine(Math.Cos(2)); // Cosine
    Console.WriteLine(Math.Cosh(5)); // Hyperbolic cosine

    Console.WriteLine(Math.Asin(0.2)); // Inverse sine
    Console.WriteLine(Math.Sin(2)); // Sine
    Console.WriteLine(Math.Sinh(-5)); // Hyperbolic sine

    Console.WriteLine(Math.Atan(-5)); // Inverse tangent
    Console.WriteLine(Math.Tan(1)); // Tangent
    Console.WriteLine(Math.Tanh(0.1));
}
```

تابع $\text{Atan2}()$: تابع $\text{Atan2}()$ دو پارامتر را دریافت کرده، اولین پارامتر را بر دومین پارامتر تقسیم کرده و آرک تانژانت نتیجه حاصل را محاسبه می کند. الگوی تابع زیر به صورت زیر می باشد:

```
double Atan2 (double y , double x)
```

تابع $\text{Atan2}()$ دو پارامتر به نام y و x دارد که آرک تانژانت y/x را محاسبه می کند. تابع $\text{Ceiling}()$: تابع $\text{Ceiling}()$ کوچکترین عدد صحیح بزرگتر یا مساوی با یک عدد را که به عنوان پارامتر ورودی آن می باشد محاسبه می کند و توانایی کار با نوع داده های decimal و double را داراست. به عنوان مثال اگر ۱.۰۲ پارامتر ورودی این تابع باشد نتیجه حاصل برابر با ۲ و اگر پارامتر ورودی تابع برابر ۱.۰۲- باشد نتیجه حاصل برابر با ۱- خواهد بود. برنامه زیر عملکرد تابع $\text{Ceiling}()$ را نشان می دهد.

```
static void Main(string[] args)
{
    double value1 = 123.456;
    double ceiling1 = Math.Ceiling(value1);

    // Get ceiling of decimal value.
    decimal value2 = 456.789M;
    decimal ceiling2 = Math.Ceiling(value2);

    // Get ceiling of negative value.
    double value3 = -100.5;
    double ceiling3 = Math.Ceiling(value3);

    // Write values.
    Console.WriteLine(value1);
    Console.WriteLine(ceiling1);
    Console.WriteLine(value2);
    Console.WriteLine(ceiling2);
    Console.WriteLine(value3);
    Console.WriteLine(ceiling3);
}
```

تابع `Exp()`: تابع `Exp()` برای محاسبه توانی از e (پایه لگاریتم طبیعی) مورد استفاده قرار می گیرد.

```
static void Main(string[] args)
{
    double a = Math.Exp(2);
    double b = Math.Pow(Math.E, 2);
    // Write results.
    Console.WriteLine(a);
    Console.WriteLine(b);
}
```

تابع `Floor()`: تابع `Floor()` بزرگترین مقدار صحیح کوچکتر یا مساوی یک عدد را که به صورت `double` یا `decimal` نمایش داده می شود محاسبه می کند. برنامه زیر عملکرد این تابع را نشان می دهد.

```
static void Main(string[] args)
{
    double value1 = 123.456;
    double value2 = 123.987;
    // Take floors of these values.
    double floor1 = Math.Floor(value1);
    double floor2 = Math.Floor(value2);
    // Write first value and floor.
    Console.WriteLine(value1);
    Console.WriteLine(floor1);
    // Write second value and floor.
    Console.WriteLine(value2);
    Console.WriteLine(floor2);
}
```

تابع `Pow()`: تابع `Pow()` توان های یک مبنا را محاسبه می کند. نتیجه حاصل از تابع، عبارت `base exp` است. اگر مبنا صفر باشد و یا توان منفی یا صفر باشد این تابع عمل نخواهد کرد. همچنین اگر مبنا منفی باشد و توان (`exp`) از نوع صحیح نباشد نتیجه ای نخواهد داد. برنامه زیر نحوه عملکرد این تابع را نشان می دهد.

```
static void Main(string[] args)
{
    // Compute the squares of the first parameters to Math.Pow.
    double value1 = Math.Pow(2, 2);
    double value2 = Math.Pow(3, 2);
    // Write results to the screen.
    Console.WriteLine(value1);
    Console.WriteLine(value2);
}
```

تابع `Sqrt()`: تابع `Sqrt()` جذر یک عدد مثبت را محاسبه می کند. برنامه زیر عملکرد این تابع را نشان می دهد.

```
static void Main(string[] args)
{
    double a = Math.Sqrt(4);
    Console.WriteLine(a);
}
```

تابع `Log()` و `Log10()`: تابع `Log()` لگاریتم طبیعی یک عدد مثبت را محاسبه می کند (پایه لگاریتم طبیعی عدد $e = 2.718281828$ است). تابع `Log10()` لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می کند برخلاف تابع `Log()` که لگاریتم مبنای e اعداد مثبت را محاسبه می کند. برنامه زیر نحوه استفاده از این توابع را نشان می دهد.

```
static void Main(string[] args)
{
    double a = Math.Log(1);
    Console.WriteLine(a);
    double b = Math.Log10(1000);
    Console.WriteLine(b);
    double c = Math.Log(1000, 10);
    Console.WriteLine(c);
}
```

تابع `Sign()`: علامت عدد تعیین شده در پارامتر ورودی را بر می گرداند. و توانایی کار با نوع داده های `decimal`, `double`, `int`, `byte`, `single` را داراست. خروجی این تابع برای اعداد مثبت عدد یک و برای اعداد منفی عدد منفی یک است. برنامه روبرو نحوه استفاده از این تابع را بیان می کند.

```
static void Main(string[] args)
{
    int a = Math.Sign(-5);
    int b = Math.Sign(5);
    int c = Math.Sign(0);
    Console.WriteLine(a);
    Console.WriteLine(b);
    Console.WriteLine(c);
    a = Math.Sign(-5.5);
    b = Math.Sign(5.5);
    Console.WriteLine(a);
    Console.WriteLine(b);
}
```

تابع `Truncate()`: جزء صحیح یک عدد اعشاری را بر می گرداند. این تابع توانایی کار با نوع داده ای `decimal` و `double` را داراست. برنامه زیر نحوه استفاده از این تابع را نشان می دهد.

```
static void Main(string[] args)
{
    decimal a = 1.223M;
    double b = 2.913;

    a = Math.Truncate(a);
    b = Math.Truncate(b);

    Console.WriteLine(a);
    Console.WriteLine(b);
}
```

تابع `Round()`: این تابع برای گرد کردن اعداد اعشاری استفاده می شود و در حالت های مختلفی به کار می رود. که به بررسی آنها می پردازیم:

۱- `Round(decimal)`: یک مقدار `decimal` را به نزدیکترین مقدار یک انتگرال گرد می کند.
 ۲- `Round(double)`: یک مقدار `double` را با دقت دو برابر اعشار و نزدیکترین مقدار به انتگرال گرد می کند.

۳- `Round(Decimal,int۳۲)`: یک مقدار `decimal` را به یک مقدار مشخص از ارقام کسری گرد می کند.

۴- `Round(decimal,MidpointRounding)`: گرد کردن یک مقدار با نقطه اعشار دقت مضاعف به یک مقدار صحیح.

۵- `Round(decimal,int۳۲,MidpointRounding)`: یک مقدار `decimal` را به تعداد مشخصی از ارقام کسری گرد می کند.

۶- `Round(double,MidpointRounding)`: یک مقدار `double` را به تعداد مشخصی از ارقام کسری گرد می کند.

تابع `Round` در واقع در دو حالت عملیات گرد کردن را انجام می دهد.

۱- عدد یا پارامتر ورودی را با توجه به پارامتر دوم گرد می کند.

۲- عدد یا پارامتر دریافتی را گرد می کند.

برنامه زیر نحوه استفاده از این تابع را در حالت های مختلف نشان می دهد.

```
static void Main(string[] args)
{
    // Round double type in three ways.
    double before1 = 123.45;
    double after1 = Math.Round(before1, 1,
        MidpointRounding.AwayFromZero); // Rounds "up"
    double after2 = Math.Round(before1, 1,
        MidpointRounding.ToEven); // Rounds to even
    double after3 = Math.Round(before1);
    Console.WriteLine(before1); // Original
    Console.WriteLine(after1);
    Console.WriteLine(after2);
    Console.WriteLine(after3);
    // Round decimal type.
    decimal before2 = 125.101M;
    decimal after4 = Math.Round(before2);
    decimal after5 = Math.Round(before2, 1);
    Console.WriteLine(before2); // Original
    Console.WriteLine(after4);
    Console.WriteLine(after5);
}
```

تابع Min() و Max() : تابع Min() کوچکترین مقدار را از بین پارامتر های ورودی بر می گرداند. و تابع Max() بزرگترین مقدار را از بین پارامتر های ورودی بر می گرداند. این دو تابع توانایی کار با انواع داده های عددی را دارا هستند. برنامه زیر نحوه استفاده از این توابع

را نشان می دهد.

```
static void Main(string[] args)
{
    // Use Math.Min on positive integers.
    int value1 = 4;
    int value2 = 8;
    int minimum1 = Math.Min(value1, value2);
    Console.WriteLine(minimum1);
    // Use Math.Min on negative and positive integers.
    int value3 = -1000;
    int value4 = 100;
    int minimum2 = Math.Min(value3, value4);
    Console.WriteLine(minimum2);
}
```

تابع DivRem() : تابع DivRem() خارج قسمت و باقیمانده یک عدد صحیح را بر می گرداند. برنامه زیر نحوه استفاده از این تابع را نشان می دهد.

```
static void Main(string[] args)
{
    const int a = 1000;
    const int b = 300;
    int result;
    int quotient = Math.DivRem(a, b, out result);
    Console.WriteLine(quotient);
    Console.WriteLine(result);
}
```

آرایه چیست؟

در فصل اول مروری کوتاه بر مفهوم آرایه ها داشتیم و مختصری با آن آشنا شدیم. آرایه دنباله ای از عناصر است. تمام عناصر آرایه در یک قطعه پیوسته از حافظه قرار می گیرند و با استفاده از اندیس های عددی مورد دستیابی قرار می گیرند. آرایه زمانی سودمند است که برنامه باید با یک مجموعه منظم از ارقام مشابه کار کند که در این صورت بسیار انعطاف پذیرتر و پر قدرتمندتر از ایجاد چندین متغیر است. تاکنون برای ذخیره داده ها از متغیر هایی از نوع های مختلف مانند int استفاده کردیم که هر کدام یک خانه از حافظه را اشغال می کردند. همیشه تعریف این گونه متغیر ها جوابگوی نیاز برنامه نویسی نیست. برای مثال، فرض کنید می خواهیم ۱۰ عدد صحیح را در حافظه نگهداری کنیم یک روش این است که ۱۰ متغیر از نوع صحیح تعریف کنیم و هر مقدار را در یک متغیر قرار دهیم. شاید این روش برای ده عدد مطلوب باشد، ولی اگر بخواهیم ۱۰۰ عدد صحیح را ذخیره کنیم، آیا تعریف ۱۰۰ متغیر در برنامه کار مناسبی است؟ در این گونه موارد، باید از متغیر های دیگر به نام متغیر های اندیس دار یا آرایه استفاده کرد. در این مثال، برای ۵۰۰ عدد صحیح فقط یک نام انتخاب می کنیم و هر مقدار را یک عنصر می نامیم. زمانی که در برنامه آرایه تعریف می کنید در واقع متغیری ایجاد می کنید که قابلیت نگهداری بیش از یک عنصر را داراست.

آرایه متداول ترین و ساده ترین ساختمان داده ای است که تقریباً در تمام زبان های برنامه سازی وجود دارد. در زبان C#.NET تمام آرایه هایی که ایجاد می شوند از کلاس انتزاعی Array مشتق می شوند. کلاس Array مجموعه ای از متدها را برای اجرای کار هایی مانند مرتب سازی و جستجو را فراهم می کند.

یکی دیگر از روش های استفاده از آرایه در زبان C#.NET، استفاده از کلاس ArrayList است. ArrayList، آرایه ای است که در صورت نیاز به عناصر بیشتر، اندازه آن به طور خودکار افزایش می یابد. در مواردی که نمی توانید اندازه آرایه را بطور دقیق تعیین کنید، یا وقتی اندازه آرایه در طول اجرای برنامه تغییر کند، استفاده از ArrayList به جای آرایه، مناسب تر است. در ادامه آرایه ها و ArrayList را بررسی می کنیم.

تصویر ۵ - ۴ یک آرایه به نام X را نشان می دهد که شامل ۵ عنصر است.

X[۰]	X[۱]	X[۲]	X[۳]	X[۴]

تصویر ۵ - ۴

در زبان C# اندیس آرایه، از صفر شروع می شود و عناصر آرایه در محل متوالی حافظه و تحت قوانین خاصی ذخیره می شوند. به کمک این قوانین می توان در هر یک از محل، اطلاعاتی را قرار داد و به هر یک از عناصر آرایه دسترسی داشت.

سوالی که در اینجا به ذهن می رسد این است که چرا اندیس ها از صفر شروع می شوند؟ همان طور که می دانید یک متغیر آدرس مکانی از حافظه کامپیوتر است، هنگامی که برای دسترسی به یک عنصر از آرایه اندیس آن را مشخص کنید. زبان C# این اندیس را در اندازه یکی از عناصر آرایه ضرب می کند و حاصل را با آدرس آرایه جمع می کند تا به آدرس عنصر مشخص شده برسد. آدرس شروع یک آرایه هم در حقیقت آدرس اولین عنصر آن آرایه است. بنابراین اولین عنصر آرایه به اندازه ضربدر اندازه عنصر از نقطه شروع فاصله دارد. دومین عنصر به اندازه یک ضربدر اندازه عنصر از شروع آرایه فاصله دارد و به همین ترتیب ادامه پیدا می کند.

همیشه در زمان استفاده از آرایه ها به نکات زیر توجه کنید :

۱- آرایه هایی با یک اندیس را آرایه یک بعدی، آرایه دارای دو اندیس را آرایه دو بعدی و به طور کلی آرایه دارای n اندیس را آرایه n بعدی می نامند.

۲- نامگذاری آرایه ها از قوانین نامگذاری متغیر ها تبعیت می کند.

۳- زبان C# حدود آرایه را کنترل می کند. بطور مثال، آرایه ای به نام X با ۱۰ عنصر را در نظر بگیرید. این آرایه را نمی توان با اندیس بزرگتر از ۹ استفاده کرد. در این مورد، کامپایلر زبان C# در زمان اجرا خطایی را اعلان می کند. این موضوع را چک کردن مرز ها می گویند.

آرایه های یک بعدی در زبان C#.NET

آرایه یک بعدی مجموعه ای از عناصر همگن است که در مکان های متوالی حافظه ذخیره می شوند. داده های موجود در آرایه می توانند از نوع داده اولیه، یا هر نوعی باشند که توسط برنامه نویس تعریف می شوند. می توان گفت که داده های آرایه، اشیا هستند. خود آرایه ها در C# نیز شیء هستند، زیرا از کلاس Array مشتق می شوند. چون آرایه، نمونه اعلان شده کلاس Array است، در هنگام استفاده از آرایه می توانید از تمام متدها و خواص این کلاس استفاده کنید.

تعریف آرایه در زبان C# در دو مرحله صورت می گیرد :

- ۱- اعلان آرایه : در اعلان آرایه، نوع عناصر آن مشخص می شود.
- ۲- تخصیص حافظه به آرایه : ولی در تخصیص حافظه به آرایه، تعداد عناصر آرایه مشخص می گردد.

این دو مرحله را می توان در یک دستور و یا دو دستور جداگانه نوشت. تخصیص حافظه به آرایه را ایجاد نمونه هایی از آرایه می گویند.

آرایه های یک بعدی به صورت زیر اعلان می شوند :

Type [] Name;

Type یکی از انواع متداول در زبان C# است، مثل int.

Name : نام آرایه است.

مثال زیر را ببینید :

```
int [] x;
```

این دستور، آرایه ای به نام x را از نوع int اعلان می کند.

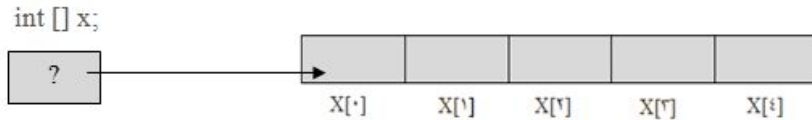
```
int [] x;
```

?

پس از اعلان آرایه می توان نمونه ای از آرایه را ایجاد کرد. در نمونه آرایه، تعداد عناصر آرایه مشخص می شود :

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

همانطور که مشاهده می کنید، برای ایجاد نمونه ای از آرایه، از عملگر new استفاده می شود.



تصویر ۶ - ۴

همانطور که در تصویر ۶ - ۴ مشاهده می کنید این دستور آرایه ای به نام x را با طول ۵ تعریف می کند. بخش اعلان و بخش تخصیص آرایه را می توان در یک دستور انجام داد.

```
int [] x = new int [۵];
```

در یک دستور می توان چندین آرایه را تعریف کرد :

```
double [] array۱ = new double [۱۰];
```

```
array۲ = new double [۲۰];
```

توجه داشته باشید که کروشه ها ([])، عملگر محسوب می شوند. تقدم [] مثل تقدم پرانتز هاست.

برای دستیابی به عناصر آرایه (جهت مقدار دادن به آن یا بازیابی مقداری از آن)، از اندیس استفاده می شود. اندیس آرایه ها در C# از ۰ تا n - ۱ است.

وقتی آرایه ای را تعریف می کنید C# به صورت خودکار مقادیر اولیه پیش فرضی را برای عناصر آن در نظر می گیرد. این مقادیر اولیه، به نوع عناصر آرایه بستگی دارد، که عبارتند از :

عناصر آرایه ی از نوع عددی، با صفر مقدار اولیه می گیرد.

عناصر آرایه ی از نوع bool، با false مقدار اولیه می گیرد.

عناصر آرایه از نوع ارجاع، با Null تهی مقدار اولیه می گیرد.

با توجه به نکات مطرح شده، آرایه x که قبلا تعریف شده است، به صورت زیر نمایش داده می شود :

	۰	۱	۲	۳	۴
x	۰	۰	۰	۰	۰

اگر بخواهید به عناصری از آرایه مقدار جدیدی بدهید، به صورت زیر عمل کنید :

```
x [۰] = ۶ ;
```

```
x [۳] = ۹ ;
```

عناصر آرایه را می توان در هنگام اجرای برنامه تعیین کرد. به عنوان مثال، فرض کنید یک کنترل TextBox در برنامه وجود دارد که عددی را به عنوان طول آرایه دریافت می کند. در برنامه، آرایه ای به این طول ایجاد می شود :

```
int len = convert.ToInt32(textBox1.Text);
int [] a = new int [len];
```

تعیین مقادیر اولیه برای عناصر آرایه

همانطور که گفتیم وقتی آرایه ای را تعریف می کنید، بسته به نوع عناصر آن، مقادیری در عناصر آرایه قرار می گیرد. در C# می توان در هنگام تعریف آرایه، مقادیر اولیه دلخواهی را برای عناصر در نظر گرفت. برای این کار، به صورت زیر عمل کنید :

```
{مقادیر} [طول] نوع = new نام [] نوع;
```

مقادیری که برای عناصر آرایه تعیین می شوند، با کما از هم جدا می شوند. دستور زیر را در نظر بگیرید :

```
int [] a = new int [5] {۲, ۳, ۴, ۵, ۶};
```

این دستور آرایه ای به نام a را با ۵ عنصر تعریف کرده و مقادیر ۲، ۳، ۴، ۵، ۶ را به عناصر نسبت می دهد. این آرایه به صورت زیر نمایش داده می شود.

	۰	۱	۲	۳	۴
a	۲	۳	۴	۵	۶

در هنگام مقدار دهی اولیه به آرایه، تعداد مقادیر باید دقیقا برابر با تعداد عناصر آرایه باشد و گرنه در زمان اجرای برنامه خطای کامپایلری رخ خواهد داد زیرا تعداد مقادیری که باید به عناصر آرایه تخصیص یابد، بیشتر از تعداد عناصر آرایه است.

عناصر آرایه را به روش دیگری نیز می توان مقدار اولیه داد:

```
{مقادیر} = نام آرایه [] نوع آرایه;
```

به عنوان مثال دستورات زیر را در نظر بگیرید :

```
int [] b = {۱,۲,۳,۴,۵};
```

این دستور، آرایه ای به طول ۵ را ایجاد می کند که مقادیر عناصر آن به شرح زیر است :

	۰	۱	۲	۳	۴
b	۱	۲	۳	۴	۵

در این روش به آرایه ای که از قبل تعریف شده است نمی توان مقدار اولیه داد. به عنوان مثال

دستورات زیر را در نظر بگیرید :

```
int [] x;
x = {۱,۲,۳}
```

این دستورات، خطای کامپایلری ایجاد می کنند، زیرا نمی توان به آرایه ای که از قبل تعریف شده است به این روش مقدار داد. برای این منظور باید به صورت زیر عمل کنید. دستورات

زیر آرایه ای با ۳ عنصر و مقادیر ۱, ۲, ۳ تعریف می کند :

```
int x;
x = new int [۳] {۱,۲,۳};
```

دستیابی به عناصر آرایه

برای دستیابی به یک عنصر آرایه، ابتدا اسم متغیر و بعد اندیس عنصر مورد نظر را در داخل کروشه بنویسید. اندیس های آرایه از صفر شروع می شوند. اندیس عنصر اول آرایه، صفر است و اندیس ۱ به عنصر دوم اشاره دارد و الی آخر.

[اندیس آرایه] نام آرایه

دستورات زیر را در نظر بگیرید :

```
a [۲] = ۲;
textBox۱.Text = a [۳]. ToString();
```

دستور اول، مقدار ۲ را در سومین عنصر آرایه (اندیس شماره ۲) قرار می دهد و دستور دوم، مقدار چهارمین عنصر آرایه را به خاصیت Text کنترل textBox۱ نسبت می دهد.

اگر از اندیسی استفاده کنید که کوچکتر از صفر و یا بزرگتر مساوی طول آرایه باشد، کامپایلر یک خطای IndexOutOfRangeException ایجاد می کند. پس در زبان C# مرز ها چک می

شوند. دستورات زیر را مشاهده کنید :

```
int [] a = {۹,۷,۳};
a [۴] = ۲;
```

با اجرای این دستورات خطای زمان اجرا رخ می دهد، زیرا در این دستور اندیس آرایه می تواند مقادیر ۰، ۱، ۲ باشد پس مقدار ۴ را نمی توان در اندیس آرایه a استفاده کرد.

دستورات زیر را در نظر بگیرید :

```
int [] x = new int [4];
```

```
for (int i = 0 ; i < 4 ; i++)
```

```
x[i] = i * i ;
```

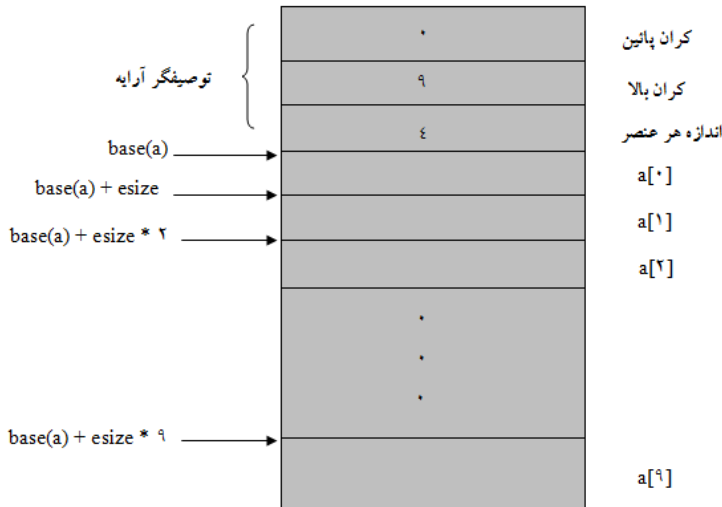
این دستورات، آرایه ای با ۴ عنصر تعریف کرده و در عناصر آن مقادیر ۰، ۱، ۴، ۹ را قرار می دهد.

پیاده سازی آرایه های یک بعدی

آرایه های یک بعدی را به آسانی می توان پیاده سازی کرد (در حافظه نمایش داد). هر آرایه دارای توصیفگری است که در ساده ترین حالت، کران پائین و کران بالای آرایه را مشخص می کند. در زبان C# کران پائین صفر و کران بالا $n - 1$ است. دستور زیر را مشاهده کنید :

```
int [] a = new int [10];
```

این آرایه را می توان مانند تصویر ۷ - ۴ نمایش داد. در این تصویر فرض شد که هر یک از عناصر آرایه که از نوع int هستند، چهار بایت از حافظه را اشغال می کنند. آدرس شروع اولین عنصر آرایه a را $base(a)$ می نامیم. اگر فرض کنیم اندازه هر عنصر آرایه esize باشد، عنصر $a[0]$ در مکان $base(a)$ ، عنصر $a[1]$ در مکان $base(a) + esize$ و عنصر $a[2]$ در مکان $base(a) + 2 * esize$ قرار می گیرد و غیره. در توصیفگر آرایه، کران پائین و بالا و اندازه عناصر آرایه قرار گرفته است.



تصویر ۷ - ۴

آرایه های یک بعدی در پردازش لیست مورد استفاده قرار می گیرند. لیست مجموعه ای از مقادیر همونوع است. چون تمام مقادیر از یک نوع هستند، به آسانی می توان لیست را در آرایه ذخیره کرد. دو عملیات مهمی که معمولاً بر روی آرایه های یک بعدی (لیست ها) انجام می شوند عبارتند از :

مرتب سازی عناصر آرایه

جستجوی عنصری در آرایه

در زبان C#، کلاس Array دارای دو متد Static به نام های Sort و BinarySearch است که اولی برای مرتب سازی آرایه ها و دومی برای جستجوی دودویی در آرایه مورد استفاده قرار می گیرد. متد Sort را به صورت زیر می توان به کار برد :

```
Array.Sort(array);
```

array آرایه ای است که باید به طور صعودی مرتب شود. اگر آرایه را مرتب صعودی کردید و خواستید آن را به صورت نزولی مرتب کنید، می توانید از متد Reverse کلاس Array برای این کار استفاده کنید. متد BinarySearch به صورت زیر به کار می رود.

```
BinarySearch (array , value);
```

این متد مقدار value را در آرایه array جستجو می کند. اگر value در آرایه وجود داشته باشد، اندیس محل وجود آن، و گرنه یک مقدار منفی را بر می گرداند.

متد و خواص مربوط به آرایه ها

دیدید که برای دستیابی به عناصر آرایه برای مقدار دادن یا بازیابی مقادیر آنها، از اندیس استفاده می شود. اما دو متد در آرایه وجود دارند که می توانند برای مقدار دادن به عناصر آرایه و بازیابی مقادیر عناصر آرایه به کار روند.

متد SetValue : این متد به صورت زیر به کار می رود.

SetValue (value , index);

به عنوان مثال دستور زیر، عنصری با اندیس ۲ را در آرایه x برابر با ۵۰ قرار می دهد.

x.SetValue (۵۰ , ۲);

متد GetValue : این متد به صورت زیر به کار می رود :

GetValue (index);

به عنوان مثال دستور زیر، مقدار عنصری با اندیس ۲ را در متغیر y قرار می دهد :

int y = System.Convert.ToInt32 (x.GetValue(۲));

چون متد GetValue مقدار عنصر را به صورت یک Object بر می گرداند، با متد ToInt32 به

یک مقدار صحیح تبدیل شده و در متغیر y قرار می گیرد.

متد(GetValueUpperBound(dimension): این متد، کران بالای بعدی از آرایه را که با dimension

مشخص می شود تعیین می کند. در آرایه یک بعدی، به جای dimension باید صفر را قرار

دهیم. دستورات زیر را ببینید :

```
int sum = ۰;
```

```
for (int i = ۰ ; i < x.GetValueUpperBound (۰) ; i++)
```

```
sum += system.convert.ToInt32 (x.GetValue (i));
```

دستور اول متغیر sum را از نوع int تعریف کرده مقدار صفر را در آن قرار می دهد. دستور

for متغیر i را اعلان کرده مقدار اولیه آن را ۰ و مقدار نهایی آن را x.GetValueUpperBound(۰)

تعیین می کند که کران بالای آرایه x است. دستور آخر، شیء برگردانده شده توسط متد

GetValue را به مقدار صحیح تبدیل کرده با Sum جمع می کند. در واقع، در اینجا مجموع تمام عناصر آرایه x محاسبه می شود.

متد GetLength: این متد، تعداد عناصر موجود در یک بعد از آرایه را بر می گرداند و به صورت زیر به کار می رود:

GetLength (dimension);

dimension شماره بعدی از آرایه است. بدیهی است که در آرایه های یک بعدی، به جای dimension باید ۰ را قرار دهیم که به معنای این است که می خواهیم تعداد عناصر موجود در آرایه یک بعدی را محاسبه کنیم. اگر k یک متغیر صحیح باشد در آرایه یک بعدی a، دو دستور زیر معادل هستند:

```
k = a.length;
k = a.GetLength (۰);
```

متد GetType: نوع آرایه را بر می گرداند و به صورت زیر به کار می رود.

object.GetType();

به عنوان مثال، اجرای دستور زیر را در نظر بگیرید:

Console.WriteLine (x.GetType());

این دستور، System.Int32[] را بر می گرداند که نشان می دهد x آرایه یک بعدی از مقادیر صحیح است. زمانی که یقین ندارید نوع آرایه چیست، مثلاً وقتی که آرایه به عنوان پارامتر به متدی ارسال می شود، این متد مفید واقع خواهد شد. همان طور که دیدید متد GetType نه تنها نوع آرایه را بر می گرداند، بلکه با آن می توان پی برد که شیء مورد نظر آرایه است (با []) در خروجی بالا).

خاصیت Length: این خاصیت، تعداد کل عناصر آرایه را مشخص می کند.

arrayName.Length;

به عنوان مثال دستورات زیر را در نظر بگیرید:

```
int [] a = {۱,۲,۳,۴};
int i = a.Length;
```

این دستور، مقدار ۴ را در متغیر i قرار می دهد. زیرا، خاصیت Length تعداد عناصر آرایه را تعیین می کند.

خاصیت Rank : این خاصیت، تعداد ابعاد آرایه را مشخص می کند. دستور زیر، تعداد ابعاد آرایه یک بعدی a را در خروجی چاپ می کند که ۱ است.

```
Console.WriteLine (x.Rank);
```

دستیابی به عناصر یک آرایه با استفاده از یک حلقه

با استفاده از یک حلقه for می توانید در عناصر یک آرایه گردش کنید. کد زیر، مقدار عناصر آرایه را در کنسول می نویسد :

```
int [] a = {۹,۳,۷,۲};

for (int i = ۰ ; i !=a.Length; i++)
{
    int a = a[i];
    Console.WriteLine(a);
}
```

البته ممکن است این گردش در بین عناصر را به اشتباه انجام دهید. ممکن است به جای اندیس صفر، از اندیس ۱ شروع کنید و در نتیجه عنصر اول را از دست بدهید. ممکن است در شرط خاتمه از عملگر $\leq$ استفاده کنید که در این صورت یک `IndexOutOfRangeException` ایجاد خواهد شد. ممکن است فراموش کنید که اندیس را افزایش دهید که در این صورت حلقه for هرگز به پایان نمی رسد. خوشبختانه، یک راه بهتر وجود دارد که این مشکلات را حل می کند و آن هم استفاده از دستور foreach است. در فصل اول در مورد این دستور و ساختار آن حرف زدیم و از تکرار آن در این بخش خودداری می کنیم. می خواهیم مثال قبل را با دستور foreach

بنویسیم :

```
int [] a = {9,3,7,2};
foreach (int a in a)
{
    Console.WriteLine (a);
}
```

بعضی از برنامه نویسان ترجیح می دهند که به جای عملگر $<$ از عملگر $\leq$ در شرط خاتمه استفاده کنند. در اینجا، اندیس، نشان دهنده تعداد ارقامی است که در کنسول نوشته می شود. در شروع حلقه، مقدار اندیس صفر است. به این معنا که هیچ رقمی در کنسول نوشته نمی شود. در پایان حلقه، اندیس هنوز هم باید نشان دهنده تعداد رقم های نوشته شده در کنسول باشد. اگر از عملگر $\leq$ استفاده کنید، بدون نگاه کردن به بدنه حلقه دقیقاً می دانید که وقتی حلقه تمام می شود، مقدار اندیس چند خواهد بود. این مقدار در پایان حلقه `a.Length` خواهد بود. زیرا اگر مساوی این مقدار نبود حلقه هنوز اجرا می شد. ولی اگر از عملگر $\leq$ استفاده

کنید، فقط می توانید نتیجه گیری کنید که در پایان حلقه، مقدار اندیس بزرگتر یا مساوی a.Length است.

دستور foreach یک متغیر تکرار (در این مثال a int) تعریف می کند که به صورت خودکار مقدار هر عنصر آرایه را یکی بعد از دیگری به خود می گیرد (in یک کلمه کلیدی است). این روش، هدف کد را صریح تر بیان می کند. دستور foreach برای گردش در بین تمام عناصر یک آرایه، روش بهتری است. ولی در بعضی موارد مجبورید که از دستور for استفاده کنید :

دستور foreach در کل عناصر آرایه گردش می کند. اگر بخواهید که فقط در یک قسمت خاص از آرایه گردش کنید (مثلا نیمه اول) یا برخی از عناصر را نادیده بگیرید (برای مثال، سه عنصر در میان) استفاده از دستور for ساده تر است.

دستور foreach همیشه از اندیس صفر به سمت اندیس ۱- Length حرکت می کند. اگر بخواهید که به سمت عقب حرکت کنید استفاده از دستور for ساده تر است.

اگر بدنه حلقه بخواهد که به جای مقدار عنصر، اندیس عنصر را بداند، باید از دستور for استفاده کنید.

اگر بخواهید که عناصر آرایه را تغییر دهید، باید از دستور for استفاده کنید، زیرا متغیر حلقه برای دستور foreach یک متغیر فقط _خواندنی است.

مثال ۴-۴: یک آرایه رشته با طول ۵ تعریف کرده و مقدار خانه های این آرایه را با اسامی دوستان خود پر کرده و عنصر صفرم را درون یک textbox نمایش دهید؟

حل : یک پروژه جدید ویندوزی ساخته، Form آن را مطابق پنجره تصویر ۸-۴ طراحی کنید.



تصویر ۸-۴

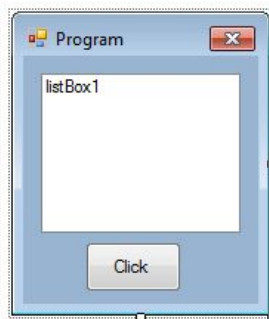
حال بر روی دکمه Click دابل کلیک کرده و دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    string[] strFriends = new string[5];
    strFriends[0] = "مرتضی سلیمان پور";
    strFriends[1] = "حسین سلطانی سیاپوش";
    strFriends[2] = "سجاد بینانمین";
    strFriends[3] = "محسن نجاریاشی";
    textBox1.Text = strFriends[0];
}
```

حال اگر برنامه را اجرا کنید مقدار موجود در خانه صفرم آرایه که برابر (مرتضی سلیمان پور) است را به کاربر نمایش می دهد.

مثال ۵ - ۴: یک آرایه رشته ای به طول ۱۰۰ تعریف کنید و سپس آن را مقدار دهی کرده و درون یک کنترل ListBox نمایش دهید؟

یک پروژه ویندوزی جدید ساخته و Form آن را مطابق پنجره تصویر ۹ - ۴ طراحی کنید.



تصویر ۹ - ۴

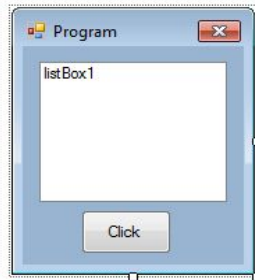
بر روی دکمه Click دابل کلیک کرده و دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    string[] str = new string[1000];
    for (int j = 0; j < 100; j++)
    {
        str[j] = "برنامه نویسی" + j;
    }
    for (int j = 0; j < 100; j++)
    {
        listBox1.Items.Add(str[j]);
    }
}
```

مثال ۶-۴: یک آرایه از نوع int با طول ۱۰۰ تعریف کرده و خانه ۲۷ و ۲۸ و ۴۵ و ۶۰ را

مقدار دهی کرده و با هم جمع و ضرب کنید؟

یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۰-۴ طراحی کنید.



تصویر ۱۰-۴

حال بر روی دکمه Click دابل کلیک کرده و دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    int sum , mul;
    int[] calc = new int[100];
    calc[27] = 2;
    calc[28] = 8;
    calc[45] = 10;
    calc[42] = 11;
    mul = calc[27] * calc[28];
    sum = calc[45] + calc[42];
    textBox1.Text = Convert.ToString(mul);
    textBox2.Text = Convert.ToString(sum);
}
```

مثال ۷-۴: آرایه ای از نوع رشته ای و به طول ۶ تعریف کرده و مقدار خانه های آن را با

اسامی دوستان خود پر کرده و با استفاده از یک حلقه foreach این مقادیر را در کنترل

ListBox نمایش دهید؟

یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۱-۴ طراحی کنید.



تصویر ۱۱-۴

از روی صفحه کلیک دکمه F7 را فشار دهید تا وارد بخش کد نویسی شوید. سپس در بخش public partial دستورات زیر را تایپ کنید :

```
public partial class Form1 : Form
{
    string[] strFriends = new string[6];
```

بر روی Form دابل کلیک کنید تا وارد رویداد Form_Load شوید، سپس دستورات زیر را تایپ کنید :

```
private void Form1_Load(object sender, EventArgs e)
{
    strFriends[0] = "مرتضی سلیمان پور";
    strFriends[1] = "مجاد بینا نمین";
    strFriends[2] = "محسن نجار بافی";
    strFriends[3] = "مهیل داوری";
    strFriends[4] = "رحمان قهرمانی";
    strFriends[5] = "محمد سروی";
}
```

حال بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

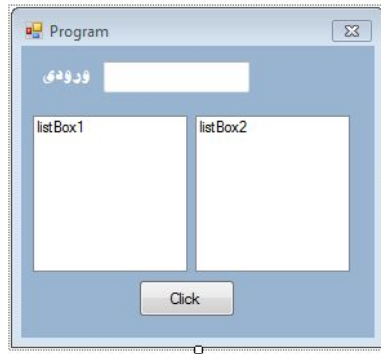
```
private void button1_Click(object sender, EventArgs e)
{
    foreach (string strName in strFriends)
        listBox1.Items.Add(strName);
}
```

حال برنامه را اجرا کرده و نتیجه را بررسی کنید.

در این برنامه ما آرایه را خارج از متد های درون یک کلاس تعریف کردیم بنابراین در تمامی متد های آن کلاس قابل دسترس خواهد بود. حلقه foreach در بین عناصر یک آرایه از رشته ها حرکت می کند. در اینجا یک متغیر کنترل کننده، هم نوع با عناصر آرایه تعریف کردیم حلقه foreach از عنصر صفرم آرایه شروع می کند و تا رسیدن به آخرین عنصر در آرایه، بین تمام عناصر جابجا می شود. در هر بار تکرار حلقه می توانید از عنصری که در متغیر کنترل کننده قرار گرفته است استفاده کنید. توجه کنید عناصر به همان ترتیبی که در آرایه وارد شده اند داخل لیست قرار می گیرند این مورد به این دلیل است که حلقه foreach از عنصر صفرم تا آخرین عنصر آرایه را به همان ترتیبی که تعریف شده اند طی می کند.

مثال ۸ - ۴: برنامه ای بنویسید که ۵ عدد را خوانده از آخرین عدد به اولین عدد نمایش دهد؟

یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۲ - ۴ طراحی کنید.



تصویر ۱۲ - ۴

در بخش `public partial` دستورات زیر را بنویسید.

```
public partial class Form1 : Form
{
    const int count = 5;
    int[] a = new int[count];
    int i = 0;
```

حال به رویداد `KeyPress` مربوط به کنترل `textBox1` رفته و دستورات زیر را در آن تایپ کنید.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    char ch;
    ch = e.KeyChar;
    if (ch == '\r')
    {
        a[i] = Convert.ToInt32(textBox1.Text);
        i++;
        if (i < count)
        {
            textBox1.Text = "";
            label1.Text = "عدد" + Convert.ToString(i + 1, 10);
        }
        else
        {
            i = 0;
            label1.Text = "عدد" + Convert.ToString(i + 1, 10);
            MessageBox.Show("لطفا بر روی دکمه کلیک کنید");
            button1.Enabled = true;
        }
    }
}
```

این دستورات برای خواندن تعدادی عدد به کار می رود. تعداد عددی که باید خوانده شود با ثابت count مشخص می شود. پس از وارد کردن هر عدد، برای وارد کردن عدد بعدی باید کلید Enter را فشار داد. کاراکتر '\r' کلید Enter را مشخص می کند. حال بر روی دکمه Click موجود بر روی Form دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    listBox2.Items.Clear();
    foreach (int j in a)
        listBox1.Items.Add(j.ToString());
    for (int j = a.Length - 1; j >= 0; j--)
        listBox2.Items.Add(a[j].ToString());
    button1.Enabled = false;
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال ۹-۴: برنامه ای بنویسید که تعداد ۱۰ عدد را از ورودی خوانده و در آرایه ای قرار می دهد و سپس معدل آنها را حساب می کند؟
حل: ابتدا یک پروژه جدید از نوع Console ایجاد کرده و دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    Console.WriteLine("Please Inset 10 Numbers to compute them average :");
    int[] data = new int [10];
    int i;
    for ( i = 0; i < 10; i++)
    {
        Console.WriteLine("Num =",i+1);
        data[i]= int.Parse(System.Console.ReadLine());
    }
    int sum=0;
    for( i=0;i<10;i++)
        sum+=data[i];
    float ave;
    ave=sum/10;
    Console.WriteLine("Result =");
    Console.WriteLine(ave);
}
```

مثال ۱۰ - ۴: برنامه ای بنویسید که با دریافت تاریخ یک روز سال جاری، تعداد روز های گذشته از ابتدای سال را مشخص می کند. این برنامه برای سال های کسبیه جواب اشتباه می دهد. حل : یک پروژه جدید از نوع Console ایجاد کرده، سپس دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    int month, day, total_days;
    int[] day_per_month = new int[12] {31,31,31,31,31,31,
        30,30,30,30,30,29};
    Console.WriteLine("Enter month (1 to 12) : ");
    month=int.Parse(System.Console.ReadLine());
    if (month <1 || month>12)
    {
        Console.WriteLine("Wrong input! Press any key to end.");
    }
    Console.WriteLine("Enter day (1 to 31) : ");
    day=int.Parse(System.Console.ReadLine());
    total_days=day;
    for(int i=0;i<month-1;i++)
        total_days+=day_per_month[i];
    Console.WriteLine("Total days from start of year is :");
    Console.WriteLine(total_days);
}
```

مثال ۱۱ - ۴: برنامه ای که تعداد ۵ عدد را از ورودی خوانده، سپس آنها را به ترتیب معکوس در آرایه دیگری قرار داده و نتیجه را به خروجی می برد. چون تمام اعداد باید خوانده شوند تا یکی یکی از آخرین عدد به اولین عدد به خروجی بروند باید در آرایه ای نگهداری شوند. حلقه تکرار اول، عناصر آرایه x را می خواند. حلقه تکرار دوم، که اندیس آن از ۴ تا صفر است عناصر آرایه x را از آخرین عنصر به اولین عنصر، به آرایه y منتقل می کند. به همین دلیل متغیر j که به عنوان اندیس آرایه y است، از صفر شروع می شود و هر بار یک واحد به آن اضافه می گردد. پس اندیس آرایه x از ۴ به صفر کاهش می یابد و اندیس آرایه y از صفر به ۴ افزایش می یابد.

حل : ابتدا یک پروژه جدید از نوع Console ساخته و دستورات زیر را در آن تایپ کنید.

```
static void Main(string[] args)
{
    int[] x = new int[5];
    int[] y = new int[5];
    int i,j;
    for (i = 0; i < 5; i++)
    {
        Console.WriteLine("Enter number :=", i + 1);
        x[i] = int.Parse(System.Console.ReadLine());
    }
    j = 0;
    Console.WriteLine("Number in inverse : =");
    for (i = 4; i >= 0; i--)
    {
        y[j] = x[i];
        Console.WriteLine(y[j]);
        j++;
    }
}
```

مثال ۱۲ - ۴: برنامه ای بنویسید که تعداد ۱۰ عدد صحیح را از ورودی می خواند و ابتدا اعداد منفی و سپس اعداد مثبت را به خروجی می برد. تعداد اعداد مثبت و منفی را نیز مشخص می کند.

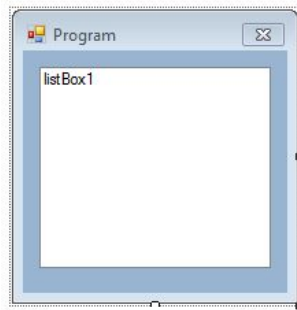
حل : ابتدا یک پروژه از نوع Console ایجاد کرده و دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    const int n = 10;
    int[] arr = new int[n];
    int i, c1 = 0, c2 = 0;
    Console.WriteLine("Enter numbers and press Enter : ");
    for (i = 0; i < n; i++)
        arr[i] = int.Parse(System.Console.ReadLine());
    Console.WriteLine("negatives area :");
    for(i=0;i<n;i++)
        if (arr[i] < 0)
        {
            Console.WriteLine(arr[i]);
            c1++;
        }
    Console.WriteLine("positives are :");
    for(i=0;i<n;i++)
        if (arr[i] > 0)
        {
            Console.WriteLine(arr[i]);
            c2++;
        }
    Console.WriteLine("number of negative =");
    Console.WriteLine(c1);
    Console.WriteLine("number of positive =");
    Console.WriteLine(c2);
}
```

مثال ۱۳ - ۴: برنامه ای بنویسید که یک تاس را ۲۰۰ بار پرتاب کند و تعیین کند که اعداد ۱، ۲، ۳، ۴، ۵، ۶ چند بار تکرار شده اند (برای تولید پرتاب تاس از یک تابع تصادفی که عددی بین ۱ تا ۶ تولید می کند استفاده کنید).

یک پروژه جدید از نوع ویندوز ایجاد کرده و Form آن را مطابق پنجره تصویر ۱۳ - ۴ طراحی کنید.

حالا بر روی Form دابل کلیک کرده و در رویداد Form\Load دستورات زیر را بنویسید.



تصویر ۱۳ - ۴

```
private void Form1_Load(object sender, EventArgs e)
{
    int[] a = new int[6];
    Random r1 = new Random();
    Random r = new Random(r1.Next());
    for (int i = 1; i <= 200; i++)
    {
        int n = r1.Next(7);
        while (n == 0) n = r1.Next(7);
        a[n - 1]++;
    }
    for (int i = 0; i < 6; i++)
        listBox1.Items.Add((i + 1).ToString() + "تکرار" + a[i].ToString());
}
```

مرتب سازی آرایه ها و انواع آن

اعمال مرتب سازی از عمومی ترین اعمالی است که در برنامه نویسی انجام می شود. در بسیاری از موارد، مرتب سازی از ضروریات برنامه نویسی است. به عنوان مثال فرض کنید، دفترچه تلفنی دارید که حاوی تلفن ۵۰۰ نفر است و می خواهید شماره تلفن یک نفر را در دفترچه تلفن بیابید. اگر دفترچه تلفن براساس نام افراد و بر حسب حروف الفبا مرتب باشد به

راحتی می توانید نام فرد و در نتیجه شماره تلفن وی را بیابید. اما اگر دفترچه تلفن براساس نام افراد مرتب نباشد، پیدا کردن شماره تلفن یک نفر از بین ۵۰۰ نفر کار دشواری است. از این موارد، بسیار زیاد است.

یکی از ویژگی های دیگر آرایه ها، پدیده مرتب سازی یا Sorting است. در برنامه نویسی واقعی معمولاً لازم است تا اطلاعات دریافت شده و یا نتیجه محاسبات را به صورت مرتب شده در اختیار کاربران قرار دهید. به همین دلیل مرتب سازی درصد زیادی از کار کامپیوتر را تشکیل می دهد.

تا به امروز روش های متعددی جهت مرتب کردن داده های موجود در یک آرایه طراحی و معرفی شده اند که هر یک نقاط ضعف و قدرتی نیز دارند. از انواع روش های معروف در مرتب سازی اطلاعات می توان به روش های (مرتب سازی حبابی، مرتب سازی Shell، مرتب سازی سریع، مرتب سازی درجی، مرتب سازی انتخابی، مرتب سازی ادغام) اشاره نمود.

مرتب سازی حبابی : این روش ساده ترین و کندترین روش مرتب سازی است و هر عضو از آرایه با اعضای بعدی آرایه مقایسه می شود و در صورت نیاز، عمل تعویض صورت می گیرد. در این روش چون هر عضو با سایر اعضای بعد از آن مقایسه می شود در آرایه های بزرگ زمان زیادی جهت مرتب شدن داده ها مصرف می شود بنابراین توصیه می شود تا در صورت کوچک بودن ابعاد یک آرایه از این روش استفاده کنید. از مزایای این روش به سادگی آن می توان اشاره کرد. به این روش، روش تعویض استاندارد (Standard Exchange) نیز می گویند. این روش شامل چند مرحله است، که در هر مرحله یک عنصر از لیست به طور قطع در محل مناسب خود قرار می گیرد. این مرتب سازی از آن رو که برای کار با عناصر آن ها را با یکدیگر می سنجد، یک مرتب سازی سنجشی است. با فرض داشتن n عضو در لیست، در بدترین حالت $\frac{n(n-1)}{2}$ عمل لازم خواهد بود.

اجازه بدهید یک آرایه از عددهای (۵، ۱، ۴، ۲، ۸) اختیار کنیم و آن را به ترتیب صعودی با استفاده از مرتب سازی حبابی مرتب کنیم. در هر مرحله عناصری که در حال مقایسه شدن با یکدیگر هستند نشان داده شده اند :

مرحله اول :

$(8, 2, 4, 1, 5) \Rightarrow (8, 2, 5, 4, 1)$

در اینجا الگوریتم دو عنصر اول را مقایسه، و جابجا می‌کند.

$(8, 2, 5, 4, 1) \Rightarrow (8, 2, 5, 1, 4)$

$(8, 2, 5, 1, 4) \Rightarrow (8, 5, 2, 1, 4)$

$(8, 5, 2, 1, 4) \Rightarrow (8, 5, 2, 4, 1)$

مرحله دوم:

$(8, 5, 2, 4, 1) \Rightarrow (8, 5, 2, 1, 4)$

$(8, 5, 2, 1, 4) \Rightarrow (8, 5, 2, 1, 4)$

$(8, 5, 2, 1, 4) \Rightarrow (8, 5, 2, 1, 4)$

$(8, 5, 2, 1, 4) \Rightarrow (8, 5, 2, 1, 4)$

در نهایت آرایه مرتب شده است و الگوریتم می‌تواند پایان پذیرد. شبه کد مرتب سازی حبابی به صورت زیر است :

```
procedure bubbleSort(A: list of sortable items) defined as:
do
  swapped:= false
  for each i in 0 to length(A) - 2 inclusive do:
    if A[ i ] > A[ i + 1 ] then
      swap(A[ i ], A[ i + 1 ])
      swapped:= true
    end if
  end for
  while swapped
end procedure
```

کارایی مرتب سازی حبابی با رعایت شرایطی می‌تواند افزایش قابل ملاحظه‌ای داشته باشد. اول این که توجه داشته باشید بعد از هر مقایسه (و احتمالاً جابجایی) در هر پیمایش، بزرگ‌ترین عنصری که از آن عبور می‌کند در آخرین موقعیت پیمایش شده قرار خواهد گرفت. از این رو بعد از اولین پیمایش بزرگ‌ترین عنصر آرایه در آخرین خانه آن خواهد بود. این یعنی با داشتن لیستی با اندازه n ، پس از اولین پیمایش، n امین عنصر در مکان نهایی اش خواهد بود. بنا بر استقرا بقیه $n - 1$ عنصر باقیمانده به همین صورت مرتب می‌شوند که بعد از پیمایش دوم، $n - 1$ امین عنصر در جای نهایی خودش خواهد بود، و الی آخر. پس طول هر پیمایش می‌تواند به

اندازه یک مرحله کم تر از پیمایش قبل از خودش باشد و به جای آنکه هر بار تمامی عناصر را تا انتهای آرایه پیماییم، عناصری که در موقعیت پایانی خود هستند و به هیچ عنوان جابجا نمی شوند چشم پوشی کنیم.

```
procedure bubbleSort(A: list of sortable items) defined as:
  n:= length(A)
  do
    swapped:= false
    n:= n - 1
    for each i in 0 to n - 1 inclusive do:
      if A[ i ] > A[ i + 1 ] then
        swap(A[ i ], A[ i + 1 ])
        swapped:= true
      end if
    end for
  while swapped
end procedure
```

مثال ۱۴ - ۴: برنامه ای بنویسید که از کاربر به تعداد دلخواه عدد دریافت کرده و سپس

اعداد ورودی را با استفاده از مرتب سازی حبابی مرتب کند؟

حل : ابتدا یک پروژه جدید از نوع Console ساخته، سپس دستورات زیر را در آن تایپ کنید.

```
static void Main(string[] args)
{
    int[] SortArray = new int[200];
    Console.WriteLine("Program for Bubble Sort");
    Console.WriteLine("Enter number of elements of Sorting array?");
    int NumberCount = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine(" ");
    Console.WriteLine("\nEnter integer Sorting array elements \n");
    for (int i = 0; i < NumberCount; i++)
    {
        SortArray[i] = Convert.ToInt32(Console.ReadLine());
    }
    int Secondlimit = NumberCount - 1;
    for (int q = 0; q < NumberCount - 1; q++)
    {
        for (int j = 0; j < Secondlimit - q; j++)
        {
            if (SortArray[j] > SortArray[j + 1])
            {
                int k = SortArray[j];
                SortArray[j] = SortArray[j + 1];
                SortArray[j + 1] = k;
            }
        }
    }
    Console.WriteLine(" ");
    Console.WriteLine("Bubble Sort Results: ");
    for (int j = 0; j < NumberCount; j++)
        Console.WriteLine(SortArray[j]);
}
```

مرتب سازی Shell : این روش مرتب سازی نسبت به روش حبابی از عملکرد بهتری برخوردار است. در این روش از یک بازه برای مقایسه عناصر موجود در آرایه استفاده می شود تا هر مقدار، در جایگاه تقریبی خود قرار بگیرد. مثلاً اگر ابعاد آرایه ۲۰ باشد، ابتدا بازه ۱۰ یعنی نصف داده ها در نظر گرفته می شود و عنصر اول با یازدهم و عنصر دوم با دوازدهم و ... عنصر دهم با بیستم مقایسه شده و در صورت نیاز جابجا خواهد شد. در مرحله بعد مقدار بازه دوباره نصف می شود و این بار عناصر اول با ششم و دوم با هفتم و الی آخر، این کار تا بازه با مقدار یک ادامه می یابد تا تمام عناصر به صورت صعودی یا نزولی مرتب شوند. نام این الگوریتم از مخترع آن گرفته شده است.

این روش که در چند مرحله انجام می شود در هر مرحله لیست موجود به مجموعه های مجزایی از عناصر افزای می شود و هر کدام از این قسمت ها معمولاً به روش درج ساده مرتب می شوند.

در هر مرحله برای تقسیم بندی لیست به مجموعه هایی مجزا از پارامتر مشخصی مانند k استفاده می شود که پارامتر «افزایش» نامیده می شود.

به عنوان مثال اگر $k = 3$ انتخاب شود، لیست مورد نظر به ۳ مجموعه مجزا به صورت زیر

تقسیم می شود :

$$S1 = \{ a[0], a[3], a[6], \dots \}$$

$$S2 = \{ a[1], a[4], a[7], \dots \}$$

$$S3 = \{ a[2], a[5], a[8], \dots \}$$

برای اینکه عنصر i ام از $j(s)$ یک آرایه را بدست آوریم از فرمول زیر استفاده می کنیم :

$$a[(i-1) \cdot k + j - 1]$$

به عنوان مثال عنصر دوم از $S3$ برابر است با :

$$a[(2-1) \cdot 3 + 3 - 1] = a[5]$$

به عنوان تمرین می خواهیم e, b, c, a, d, f را مرتب کنیم.

F	d	a	c	b	e	شروع :
C	b	a	f	d	e	مرحله اول :
A	b	c	d	e	f	مرحله دوم :
A	b	c	d	e	f	نتیجه :

در مرحله اول : داده های با فاصله ۳ از یکدیگر، مقایسه و مرتب شده، در مرحله دوم داده ها با فاصله ۲ از یکدیگر، مقایسه و مرتب می شوند و در مرحله دوم داده ها با فاصله یک از یکدیگر مقایسه و مرتب می شوند.

منظور از فاصله سه این است که عنصر اول با عنصر چهارم ($3 + 1$)، عنصر دوم با عنصر پنجم ($3 + 2 = 5$) و عنصر سوم با عنصر ششم ($3 + 3 = 6$) مقایسه شده در جای مناسب خود قرار می گیرد.

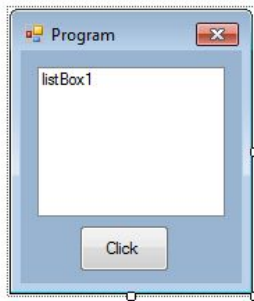
برای انتخاب فاصله در اولین مرحله، تعداد عناصر لیست بر ۲ تقسیم می شود ($n / 2$) و فاصله بعدی نیز با تقسیم فاصله فعلی بر ۲ حاصل می گردد و الگوریتم تا زمانی ادامه پیدا می کند که این فاصله به صفر برسد. برای نمونه اگر تعداد عناصر برابر با ۱۰ باشد، فاصله در مرحله اول برابر با ۵، در مرحله دوم برابر با ۲ و در مرحله سوم برابر با ۱ و در نهایت برابر با صفر خواهد بود. زمان مرتب سازی shell از رابطه n پیروی می کند که نسبت به n^2 بهبود خوبی پیدا کرده است لذا سرعت عمل روش مرتب سازی Shell از روش های انتخابی، درجی و حبابی بیشتر است. الگوریتم مرتب سازی Shell به صورت زیر است :

```
for (i = 0 ; i < numk ; i++)
{
    s = inck [i] ;
    for (j = s ; j < n ; j++)
    {
        y = x [j];
        for (k = j - s ; k >= 0 && y < x [k] ; k - = s)
            x [k + s] = x [k];
        x [k + s] = y;
    }
}
```

مثال ۱۵ - ۴ : برنامه ای بنویسید که در آن یک آرایه به طول ۵ تعریف کرده و سپس با استفاده از خاصیت SetValue آرایه مقادیر ۷ - ۴ - ۲ - ۹ - ۸ را با استفاده از مرتب سازی Shell مرتب کنید؟

(از خاصیت GetValue آرایه نیز برای بدست آوردن نتیجه مرتب سازی استفاده کنید.)

ابتدا یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۴ - ۴ طراحی کنید.



تصویر ۱۴ - ۴

در قسمت `public partial` قطعه کد زیر را وارد کنید.

```
public partial class Form1 : Form
{
    int[] arr = new int[5];
```

دقت کنید می توانیم از کلمه کلیدی `Array` به جای `int[]` استفاده کنیم. یعنی دستور زیر هیچ تفاوتی با دستور زیر ندارد.

```
public partial class Form1 : Form
{
    Array arr = new int[5];
```

حال متد `ShellSort` را به صورت زیر تعریف کنید. در مورد نحوه تعریف متد ها در فصل اول صحبت کردیم. در فصل آینده به طور کامل صحبت خواهیم کرد.

```
private void ShellSort()
{
    int stop, swap, limit, temp;
    int i = (int)(arr.Length / 2);
    while (i > 0)
    {
        stop = 0;
        limit = arr.Length - i;
        while (stop == 0)
        {
            swap = 0;
            for (int k = 0; k < limit; k++)
            {
                if ((int)arr.GetValue(k) > (int)arr.GetValue(k + i))
                {
                    temp = (int)arr.GetValue(k);
                    arr.SetValue(arr.GetValue(k + i), k); ;
                    arr.SetValue(temp, k + i);
                    swap = k;
                }
            }
        }
    }
}
```

```

    }
    limit = swap - i;
    if (swap == 0)
        stop = 1;
    }
    i = (int)(i / 2);
}
}

```

حال بر روی Form دابل کلیک کنید تا وارد Form_Load شده و دستورات زیر را بنویسید.

```

private void Form1_Load(object sender, EventArgs e)
{
    arr.SetValue(8, 0);
    arr.SetValue(9, 1);
    arr.SetValue(2, 2);
    arr.SetValue(4, 3);
    arr.SetValue(7, 4);
    for (int i = 0; i < arr.Length; i++)
        listBox1.Items.Insert(i, arr.GetValue(i));
    this.Text = "SHELL SORT APPLICATION";
}

```

بر روی دکمه Click دابل کلیک کرده و در رویداد Click این کنترل دستورات زیر را

بنویسید.

```

private void button1_Click(object sender, EventArgs e)
{
    ShellSort();
    listBox1.Items.Clear();
    for (int x = 0; x < arr.Length; x++)
        listBox1.Items.Insert(x, arr.GetValue(x));
}

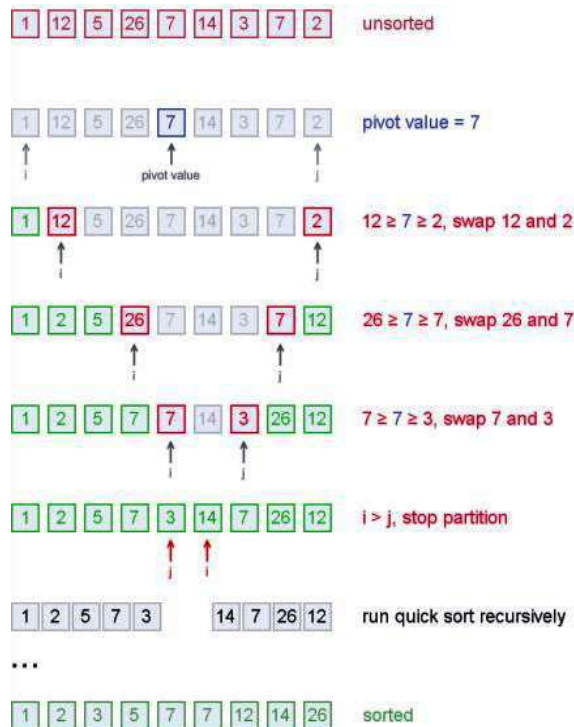
```

برنامه را اجرا کرده و نتیجه را مشاهده کنید.

مرتب سازی سریع : مرتب سازی سریع یا Qucik Sort یکی از روش های مرتب سازی آرایه است که به دلیل مصرف حافظه کم، سرعت اجرای مناسب و پیاده سازی ساده بسیار مورد قبول واقع شده است. مرتب سازی سریع از راهبرد تقسیم و حل استفاده می کند. این راهبرد یک روش بازگشتی است که در آن، مسئله ای که باید حل شود به مسئله های کوچکتر تقسیم می گردد و هر کدام مستقل حل می شوند. تمام یا بعضی از این مسئله های کوچکتر نیز ممکن است پیچیده باشند و نیاز باشد راهبرد تقسیم و حل بر روی آنها عمل شود. این روند می تواند آنقدر ادامه یابد تا مسئله های ساده ای ایجاد شوند که حل آنها آسان باشد. در مرتب سازی

سریع، روش تقسیم و حل، عنصری به نام محور انتخاب می کند و سپس دنباله ای از تعویض ها صورت می گیرد تا عناصری که کوچکتر از این محور هستند در سمت چپ آن و بقیه در سمت راست آن قرار گیرند. بدین ترتیب، محور در جای مناسبی قرار می گیرد و لیست را به دو بخش کوچکتر تقسیم می کند که هر کدام از این بخش ها به طور مستقل و به همین روش مرتب می شوند. تکرار این روند موجب می شود تا لیست های کوچکی ایجاد شوند که مرتب باشند و یا مرتب کردن آنها آسان باشد.

یکی از سریع ترین الگوریتم های مرتب سازی، الگوریتم Quick Sort می باشد که نه تنها در بحث آموزش بلکه در محیط ها و برنامه های کاربردی روزمره نیز مورد استفاده قرار می گیرد. این الگوریتم مرتب سازی توسط فردی به نام هور در سال ۱۹۶۲ ابداع گردید. قبل از شروع هرگونه توضیحی به دقت به تصویر ۱۵ - ۴ توجه نمایید و خودتان سعی کنید تا با توجه به این مثال به درکی هر چند مختصر نسبت به این الگوریتم، دست یابید.



در این روش ابتدا آرایه را به دو قسمت تقسیم نموده و بعد یکی از عناصر آرایه را به عنوان محور اصلی (pivot) در نظر می گیریم. فرقی نمی کند که عنصر اولی باشد یا دومی یا وسطی و یا هر عنصر دیگر. سپس تمام عناصر کوچکتر از عنصر محور را در سمت چپ و عناصر بزرگتر را در سمت راست عنصر محور قرار می دهیم. اکنون نتیجه کار به صورت دو آرایه با اضافه عنصر محوری تبدیل شده است. اگر دقت کنید عناصر موجود در هر آرایه یا از عنصر محوری بزرگتر و یا کوچکتر می باشند ولی هیچ یک از دو آرایه مرتب نمی باشند. پس باید دوباره به ازای هر یک از آرایه های حاصل و بصورت بازگشتی، الگوریتم فوق را اجرا نموده تا در نهایت آرایه اولیه مرتب گردد.

الگوریتم مرتب سازی سریع به صورت زیر است :

ورودی : لیستی در آرایه x ذخیره شده است $(x[1], \dots, x[n])$.

خروجی : x طوری مرتب می شود که $x[1] \leq x[2] \leq \dots \leq x[n]$

۱- اگر لیست دارای صفر یا یک عنصر است، برگشت کن (لیست مرتب است).

وگرنه اعمال زیر را انجام بده :

۲ - عنصری از لیست را به عنوان محور (pivot) انتخاب کن

۳- بقیه عناصر را به دو گروه مجزا تقسیم کن.

$\text{SmallerThanPivot} = \{ \text{تمام عناصر کوچکتر از محور} \}$

$\text{LargerThanpivot} = \{ \text{تمام عنصر کوچکتر از محور} \}$

۴- لیستی را که به صورت زیر مرتب شده است برگردان.

$\text{Quicksort}(\text{SmallThanPivot}), \text{pivot}, \text{Quicksort}(\text{largerTjanpivot})$

برای درک بهتر این الگوریتم به تصویر ۱۵ - ۴ مجددا نگاه کنید. آرایه ای که قرار است با استفاده از این الگوریتم مرتب گردد شامل مقادیر زیر می باشد.

۱۲, ۵, ۲۶, ۷, ۱۴, ۳, ۷, ۲

برای شروع یکی از اعضا «بطور مثال عدد ۷» را به عنوان عنصر محوری در نظر می گیریم.

حال دو متغیر i و j را در نظر گرفته که اولی به ابتدای آرایه و دومی به انتهای آرایه اشاره می کنند. به کمک این دو متغیر و تا زمانی که $i > j$ نشده است، یکی یکی عناصر را به عنصر

محوری مقایسه می کنیم و در جای خود قرار می دهیم. سپس یکی به i اضافه نموده و یک واحد نیز از j کم می نماییم. نتیجه کار به صورت زیر می گردد :

۱,۲,۵,۷,۳,۱۴,۷,۲۶,۱۲

در این حالت فرض می کنیم که دو آرایه برای مرتب کردن داریم و دوباره همین عملیات را برای هریک از آرایه ها انجام می دهیم تا در نهایت به جواب زیر دست یابیم.

۱,۲,۳,۵,۷,۷,۱۲,۱۴,۲۶

در ادامه نمونه برنامه ای که الگوریتم فوق را پیاده سازی کرده است، نمایش داده می شود.

مثال ۱۶ - ۴: برنامه ای بنویسید که با استفاده از مرتب سازی سریع مقادیر z, e, x, c, m ,

a, q را در یک آرایه رشته ای تعریف کرده و مرتب کند.

ابتدا یک پروژه از نوع Console ساخته و دستورات زیر را تایپ کنید.

```
while (elements[j].CompareTo(pivot) > 0)
{
    j--;
}

if (i <= j)
{
    // Swap
    IComparable tmp = elements[i];
    elements[i] = elements[j];
    elements[j] = tmp;
    i++;
    j--;
}
}
// Recursive calls
if (left < j)
{
    Quicksort(elements, left, j);
}
if (i < right)
{
    Quicksort(elements, i, right);
}
}
```

```
static void Main(string[] args)
{
    // Create an unsorted array of string elements
    string[] unsorted = { "z", "e", "x", "c", "m", "q", "a" };

    // Print the unsorted array
    for (int i = 0; i < unsorted.Length; i++)
    {
        Console.Write(unsorted[i] + " ");
    }
    Console.WriteLine();
    // Sort the array
    Quicksort(unsorted, 0, unsorted.Length - 1);
    // Print the sorted array
    for (int i = 0; i < unsorted.Length; i++)
    {
        Console.Write(unsorted[i] + " ");
    }
    Console.WriteLine();
}

public static void Quicksort(IComparable[] elements, int left, int right)
{
    int i = left, j = right;
    IComparable pivot = elements[(left + right) / 2];

    while (i <= j)
    {
        while (elements[i].CompareTo(pivot) < 0)
        {
            i++;
        }
    }
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

مرتب سازی درجی : روش مرتب سازی درجی (Insertion Sort) بدین گونه است که هنگام خواندن داده ها به داخل حافظه، آنها را مرتب می کنیم. با توجه به آن که اصولاً در عملیات ورودی و خروجی مقداری زمان جهت تبادل داده ها بین ترمینال های سخت افزاری کامپیوتر به هدر می رود، می توان عملیات مرتب سازی درجی را برای لیست های کوچک در این زمان مرده انجام داد. مرتب سازی درجی به اینگونه عمل می کند که عناصر لیست را تک تک از ورودی می خواند و همزمان در مکان مناسبی از لیست قرار می دهد. جهت ایجاد فضای مناسب در بین عناصر، عناصر لیست برای قرار دادن عنصر مورد نظر در مکان مناسب خود، عناصر لیست را از انتهای لیست تا مکان مورد نظر یکی یکی، یک خانه به سمت راست شیفت می دهیم. در آن صورت، فضای لازم در بین عناصر آرایه ایجاد می شود.

روش مرتب‌سازی درجی (Insertion Sort) یکی از روش‌های مقدماتی مرتب‌سازی مبتنی بر مقایسه عناصر است که در مقایسه با روش‌های دیگر بیشتر مورد توجه قرار دارد.

قفسه کتابی را در نظر بگیرید که قصد دارید کتابها را بر اساس عنوان و به ترتیب حروف الفبا مرتب کنید. از یک سمت قفسه شروع به مرتب کردن می‌کنید. ابتدا کتاب دوم را با کتاب اول مقایسه کرده و در صورت لزوم جابجا می‌کنید. سپس کتاب سوم را از محل خود برداشته، و در مقایسه با دو کتاب قبلی در محل مناسب قرار می‌دهید. به همین ترتیب کتابهای بعدی را نیز نسبت به کتابهای مرتب‌شده قبلی در محل مناسب درج می‌کنید، تا به آخر قفسه برسید.

عملکرد این الگوریتم به گونه‌ای است که در پایان هر مرحله قسمتی از داده‌ها به صورت کامل مرتب هستند. در مرحله بعدی نیز داده‌ای از میان داده‌های غیرمرتب به این قسمت مرتب وارد شده و در محل مناسب درج می‌شود. اگر بخواهیم با روش مرتب‌سازی درجی لیست اعداد زیر را به صورت صعودی (کوچک به بزرگ) مرتب کنیم، در پایان هر مرحله ترتیب عناصر به صورت زیر خواهد بود :

۰: ۵, ۱, ۲, ۷, ۳, ۶

۱: ۱, ۵, ۲, ۷, ۳, ۶

۲: ۱, ۲, ۵, ۷, ۳, ۶

۳: ۱, ۲, ۵, ۷, ۳, ۶

۴: ۱, ۲, ۳, ۵, ۶, ۷

۵: ۱, ۲, ۳, ۵, ۶, ۷

برخی از ویژگی‌های این الگوریتم به صورت زیر است :

۱- پیاده‌سازی آن آسان است و برای داده‌های کم، کارآمدتر است.

۲- پایدار است یعنی ترتیب نسبی عناصر یکسان را حفظ می‌کند.

یک الگوریتم برخط است. یعنی یک لیست را به محض دریافت کردن، مرتب می‌کند.

یک شبه کد ساده از الگوریتم به صورت زیر است. در اینجا آرایه ها از صفر شروع می شوند و حلقه for هم دارای هر دو کران بالا و پایین است.

```
insertionSort(array A)
begin
  for i = 1 to length[A]-1 do
    begin
      value := A[i]
      j := i-1
      while j >= 0 and A[j] > value do
        begin
          A[j + 1] = A[j]
          j := j-1
        end;
      A[j+1] := value
    end;
  end;
end;
```

مثال ۱۷ - ۴: برنامه ای بنویسید که در آن یک آرایه به طول ۵ تعریف کرده و سپس با استفاده از خاصیت SetValue آرایه مقادیر ۷ - ۴ - ۲ - ۹ - ۸ را با استفاده از مرتب سازی Shell مرتب کنید؟

(از خاصیت GetValue آرایه نیز برای بدست آوردن نتیجه مرتب سازی استفاده کنید).
ابتدا یک پروژه ویندوزی جدید ساخته و Form آن را مطابق پنجره تصویر ۱۶ - ۴ طراحی کنید.



تصویر ۱۶ - ۴

در بخش public partial دستورات زیر را بنویسید.

```
public partial class Form1 : Form
{
    Array arr = new int[5];
```

سپس تابع InsertionSort() را مطابق دستورات زیر بنویسید.

```
private void InsertionSort()
{
    int temp = 0;
    int j = 0, k = 0;
    for (j = 1; j <= (arr.Length) - 1; j++)
    {
        temp = (int)arr.GetValue(j);
        k = j - 1;
        while (k >= 0 && ((int)arr.GetValue(k)) > temp)
        {
            arr.SetValue(arr.GetValue(k), k + 1);
            k = k - 1;
        }
        arr.SetValue(temp, k + 1);
    }
}
```

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را

```
private void button1_Click(object sender, EventArgs e)
{
    InsertionSort();
    listBox1.Items.Clear();
    for (int x = 0; x < arr.Length; x++)
        listBox1.Items.Insert(x, arr.GetValue(x));
}
```

روی Form دابل کلیک کرده و وارد رویداد Form_Load شوید، بنابراین دستورات زیر را

```
private void Form1_Load(object sender, EventArgs e)
{
    arr.SetValue(8, 0);
    arr.SetValue(9, 1);
    arr.SetValue(2, 2);
    arr.SetValue(4, 3);
    arr.SetValue(7, 4);
    for (int i = 0; i < arr.Length; i++)
        listBox1.Items.Insert(i, arr.GetValue(i));
    this.Text = "INSERTION SORT APPLICATION";
}
```

برنامه را اجرا کرده و نتیجه را ببینید.

نکته!

به کار بردن مرتب سازی درجی برای لیست های بلند توصیه نمی شود، چرا که در آن صورت به دلیل عملیات وقت گیر بودن شیفต์ دادن عناصر، زمان مرتب سازی بیش از حد انتظار افزایش می یابد.

مرتب سازی انتخابی : مرتب سازی انتخابی (Selection Sort) یکی از انواع الگوریتم مرتب سازی می باشد که جزو دسته الگوریتم های مرتب سازی مبتنی بر مقایسه است. در مرتب سازی انتخابی به این صورت عمل می شود که در بین عناصر لیست، کوچکترین عنصر انتخاب می شود و جای آن با عنصر اول لیست عوض می شود. دفعه بعد، از محل دوم لیست به بعد، کوچکترین عنصر پیدا می شود و جای آن با عنصر دوم لیست عوض می شود. این روند تا مرتب شدن کامل لیست ادامه می یابد. مرتب سازی انتخابی برای ساختار هایی مانند لیست پیوندی که روش حذف و اضافه کارایی دارند سودمند است. در این حالت کوچکترین عنصر از لیست حذف شده و سپس به انتهای مقادیری که قبلا مرتب شده اند اضافه می شود. مثال زیر را در نظر بگیرید :

```
64 25 12 22 11
11 64 25 12 22
11 12 64 25 22
11 12 22 64 25
11 12 22 25 64
```

پیاده سازی الگوریتم مرتب سازی انتخاب به صورت زیر می باشد :

```
for i:=0 to n-2 do
  min:=i
  for j:=(i + 1) to n-1 do
    if A[j] < A[min] then
      min:=j
    end if
  end for
  swap (A[i] , A[min])
end for
```

مثال ۱۸ - ۴ : برنامه ای بنویسید که ۱۰ عدد را خوانده و با استفاده از روش مرتب سازی

انتخابی به صورت صعودی مرتب کند؟

حل : ابتدا یک پروژه جدید ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۱۷ - ۴ طراحی کنید.



تصویر ۱۷ - ۴

ثابت Count، آرایه a و متغیر i را در بخش public partial تعریف کنید.

```
public partial class Form1 : Form
{
    const int count = 10;
    int[] a = new int[count];
    int i;
```

بر روی ناحیه خالی Form دابل کلیک کنید سپس در رویداد Form1_Load دستورات زیر را

```
private void Form1_Load(object sender, EventArgs e)
{
    i = 0;
    label1.Text = "عدد" + Convert.ToString(i + 1, 10);
    button1.Enabled = false;
}
```

بر روی دکمه Click دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    for (int n = 0; n < count - 1; n++)
    {
        int min = a[n], l = n;
        for (int m = n + 1; m < count; m++)
            if (min > a[m])
            {
                min = a[m];
                l = m;
            }
        int temp = a[l];
        a[l] = a[n];
        a[n] = temp;
    }
    listBox1.Items.Clear();
    for (int n = 0; n < count; n++)
        listBox1.Items.Add(a[n].ToString());
}
```

رویداد KeyPress مربوط به کنترل textBox1 را به آن اضافه کرده و دستورات آن را به صورت زیر تغییر دهید.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == '\r')
    {
        a[i] = Convert.ToInt32(textBox1.Text);
        i++;
        if (i < count)
        {
            textBox1.Text = "";
            textBox1.Focus();
            label1.Text = "عدد" + Convert.ToString(i + 1, 10);
        }
        else
        {
            i = 0;
            label1.Text = "عدد" + Convert.ToString(i + 1, 10);
            MessageBox.Show("لطفا بر روی دکمه کلیک کنید");
            button1.Enabled = true;
        }
    }
}
```

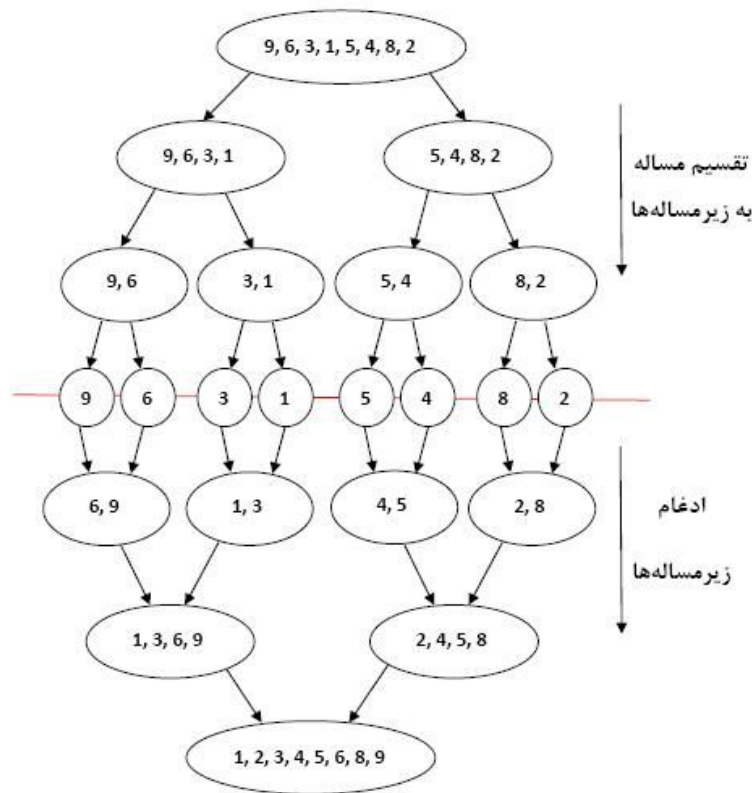
حالا برنامه را اجرا کرده و نتیجه را می توانید مشاهده کنید.

مرتب سازی ادغام : مرتب سازی ادغام یک الگوریتم مرتب سازی تطبیقی است که، در اکثر پیاده سازی ها این الگوریتم پایدار می باشد. بدین معنی که این الگوریتم ترتیب ورودی های مساوی را در خروجی مرتب شده حفظ می کند. این یک مثال از الگوریتم تقسیم و تسخیر می باشد. این الگوریتم در سال ۱۹۴۵ توسط جان فون نویمان اختراع شده است. روش مرتب سازی ادغامی (Merge Sort) یک روش مرتب سازی مبتنی بر مقایسه عناصر با استفاده از روش تقسیم و غلبه است. این روش از مراحل بازگشتی زیر تشکیل یافته است:

۱- آرایه را به دو زیر آرایه با اندازه تقریباً یکسان تقسیم کن.

۲- دو زیر آرایه را به روش مرتب سازی ادغامی مرتب کن.

۳- دو زیر آرایه مرتب شده را ادغام کن.



تصویر ۱۸ - ۴

مرتب سازی ادغام، یک الگوریتم بازگشتی است در این الگوریتم ابتدا آرایه به چند آرایه مرتب دوعنصری تقسیم می شود سپس دو به دو با هم ادغام می شوند و آرایه های مرتب چهار عنصری بوجود می آید، سپس آرایه های چهار عنصری با هم ادغام می شوند تا آرایه های مرتب هشت عنصری به وجود آید و این روند آن قدر ادامه می یابد تا آخرین دو آرایه ی مرتب با هم ادغام شوند و یک آرایه مرتب بدست آید. نمونه ای از آن، تابعی است که دو آرایه مرتب a و b به طول های n_1 و n_2 را گرفته، آن ها را در آرایه دیگری به نام c به طول n_3 به طور مرتب ادغام می کند. این تابع را `mergeArr()` نامیده به این صورت می نویسیم :

```

template <class T>
void mergArr(T a[], const int n1, T b[], const int n2, T c[], int n3)
{
    int i, j, k;
    i = 0;
    j = 0;
    if ((n1 + n2) != n3)
    {
        cout << "Size of n3 is incorrect.";
        getch();
        exit(0);
    }
    for (k = 0; i < n1 && j < n2; k++)
        if (a[i] < b[j])
            c[k] = a[i++];
        else
            c[k] = b[j++];
    while (i < n1)
        c[k++] = a[i++];
    while (j < n2)
        c[k++] = b[j++];
}

```

روش های جستجوی داده ها در آرایه ها

تاکنون روش های مختلفی برای ورود، ذخیره سازی و مرتب سازی داده ها آموختید. اما گاهی اوقات لازم است تا داده ای را در میان مجموعه ای از اطلاعات موجود جستجو کرده و مورد دستیابی قرار دهیم. یکی از دلایل استفاده از آرایه ها جستجوی سریع تر و راحت تر داده ها در میان انبوهی از اطلاعات است.

جستجوی داده در یک آرایه مانند مرتب سازی از روش های مختلفی امکان پذیر است که از مهم ترین آنها می توان روش جستجوی خطی و روش جستجوی دودویی را نام برد.

روش جستجوی خطی (Linear Search): این روش یکی از ساده ترین روش های جستجو است در این روش برای پیدا کردن یک مقدار، مقدار مربوطه را یک به یک با اعضای آرایه مورد مقایسه قرار داده و در صورت پیدا شدن اطلاعات مورد نظر جستجو خاتمه می یابد و اگر مقدار مورد نظر در هیچ یک از اعضای آرایه پیدا نشود در آن صورت نتیجه جستجو منفی خواهد بود و مقدار مربوطه در آرایه وجود نخواهد داشت. به جستجوی خطی، جستجوی ترتیبی (Sequential Search) نیز می گویند.

کد زیر روش جستجوی خطی را نشان می دهد. مقدار x درون آرایه A با n عنصر جستجو می شود و موقعیت آن را بر می گرداند اگر در آرایه وجود نداشته باشد صفر برگردانده می شود.

```
int BinarySearch(int A, int value, int low, int high)
{
    if (high < low)
        return -1 // not found
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid // found
}
```

روش جستجوی دودویی (Binary Search): روش جستجوی خطی در آرایه های بزرگ بسیار کند و وقت گیر است، برای آنکه زمان جستجو کاهش یابد از راه حل ساده ای می توان استفاده کرد. روش جستجوی دو دویی با استفاده از این راه حل ساده تعداد مقایسه ها را به مقدار زیادی کاهش می دهد.

در این روش ابتدا آرایه را مرتب کرده و سپس عضو میانی آرایه با مقدار مورد جستجو، مقایسه می شوند، اگر مقدار مورد جستجو کوچکتر از عضو میانی باشد جستجو در قسمت پایینی عضو میانی انجام می گیرد سپس همین اعمال برای نیمه پایینی و یا بالایی در آرایه انجام می شود و بدین ترتیب تا زمان پیدا شدن مقدار مورد نظر یا با نصف شدن تعداد اعضای باقیمانده عملیات ادامه می یابد. کد بازگشتی جستجوی دودویی بدین صورت می باشد:

```
int i=0;
for(i=0;i<=n;i++)
{
    if(a[i] == x)
    {
        cout<<"find!";
        return 1;
    }
    else if(a[i]!= x)
        continue;
}
cout<<"Not Found!";
return 0;
```

مثال ۱۹ - ۴ : برنامه ای بنویسید که ۱۰ عدد بین ۱ تا ۳۰ را به صورت مرتب در آرایه ای قرار داده یک عدد را از ورودی می خواند و در آرایه به روش دو دویی جستجو می کند.

حل : ابتدا یک پروژه ویندوز ایجاد کرده و Form آن را مطابق پنجره تصویر ۱۸ - ۴ طراحی کنید.



تصویر ۱۹ - ۴

بر روی دکمه Click دابل کلیک کرده تا وارد رویداد Click آن شوید، سپس دستورات زیر را

بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    int[] a = new int[10] { 1, 5, 8, 10, 14, 16, 23, 24, 26, 30 };
    int num = System.Convert.ToInt32(textBox1.Text);
    bool find = false;
    int low = 0;
    int high = 19;
    while (high >= low && !find)
    {
        int mid = (int)(high + low) / 2;
        if (a[mid] == num)
            find = true;
        else if (num > a[mid])
            low = mid + 1;
        else high = mid - 1;
    }
    if (find)
        MessageBox.Show("پیدا شد");
    else
        MessageBox.Show("پیدا نشد");
}
```

آرایه های چند بعدی

تاکنون با نحوه تعریف آرایه های یک بعدی آشنا شدید اما در برنامه نویسی پروژه های بزرگ، گاهی اوقات آرایه های یک بعدی نیز گره گشا نیستند و برنامه نویسی را با مشکلات متعددی روبرو می کنند. استفاده از آرایه های دو بعدی و یا بالاتر می تواند پاسخگوی نیازهای برنامه نویسی در این زمینه باشد.

آرایه دو بعدی که نام دیگر آن ماتریس است، کاربردهای فراوانی در حل مسائل دارد.

نمونه هایی از کاربرد های آن عبارتند از : جمع، ضرب و تفریق ماتریس ها، که آنها را به راحتی می توانید برنامه نویسی کنید. آرایه های دو بعدی مانند ماتریس های دو بعدی از دو اندیس برای شناسایی اعضای خود استفاده می کنند. اندیس اول شماره سطر و اندیس دوم شماره ستون متناظر با آرایه را مشخص می کند. برای روشن شدن بهتر موضوع به ماتریس زیر (3×4) توجه کنید، این ماتریس دارای ۴ سطر و ۳ ستون است.

شماره ستون		۰	۱	۲
شماره سطر	۰	a_{00}	a_{01}	a_{02}
	۱	a_{10}	a_{11}	a_{12}
	۲	a_{20}	a_{21}	a_{22}
	۳	a_{30}	a_{31}	a_{32}

همانطور که می بینید برای دسترسی به هر یک از عناصر آرایه a از دو اندیس استفاده می شود. مثلاً عنصر a_{21} ، عضوی است که در سطر شماره ۲ و ستون شماره ۱ قرار دارد. در آرایه نیز از همین شکل آدرس دهی برای دستیابی به اعضای یک آرایه دو بعدی استفاده می شود. آرایه چند بعدی دارای چند اندیس است، به عنوان مثال، آرایه دو بعدی، دو اندیس دارد و آرایه سه بعدی دارای سه اندیس است. در $C\#$ آرایه می تواند تا 32 بعد داشته باشد ولی آرایه هایی با بیش از ۳ بعد، بسیار نادر و اداره کردن آن بسیار دشوار است. تعریف آرایه چند بعدی به صورت زیر است :

[طول بعد ۱, ..., طول بعد ۲, طول بعد ۳] نوع آرایه = new نام آرایه [, ...,] نوع آرایه

به عنوان مثال، دستورات زیر را در نظر بگیرید :

```
int[, ] a = new int[2, 6];
int[, , ] b = new int[2, 3, 5];
```

دستور اول، یک آرایه دو بعدی به نام a تعریف می کند که دارای ۲ سطر و ۶ ستون است و نوع عناصر آن int است. دستور دوم، آرایه سه بعدی به نام m از نوع $float$ تعریف می کند که دارای ۲ سطر، ۳ ستون و ارتفاع ۵ است.

به عنوان مثال فرض کنید می خواهیم نمرات سه درس دانش آموزان یک کلاس ۵ نفره را به صورت یک ماتریس دو بعدی بنویسیم. اطلاعات را به صورت این جدول می توان نوشت :

	۰	۱	۲
۰	۱۷	۱۴	۱۶
۱	۱۲	۱۰	۸
۲	۲۰	۱۸	۱۵
۳	۱۴	۱۳	۱۹
۴	۱۷	۲۰	۱۱
	شیمی	فیزیک	ریاضی

جدول فوق را می توان به صورت ماتریس نوشت :

$$M_{5 \times 3} = \begin{pmatrix} 17 & 14 & 16 \\ 12 & 10 & 8 \\ 20 & 18 & 15 \\ 14 & 13 & 19 \\ 17 & 20 & 11 \end{pmatrix}$$

همانطور که مشاهده می کنید به وسیله ماتریس M با ۵ سطر (۰ تا ۴) و ۳ ستون (۰ تا ۲) توانستیم اطلاعات جدول را به صورت کامل اما خلاصه تر، نمایش دهیم. در $C\#$ ، عناصر آرایه به صورت سطری ذخیره می شوند. در این روش، ابتدا کلیه عناصر موجود در سطر اول، سپس تمام عناصر در سطر دوم و ... در حافظه ذخیره می شوند. اندیس هر بعد از صفر شروع می شود.

مقدار فضایی که آرایه n بعدی اشغال می کند، به صورت زیر محاسبه می شود :

$$\text{طول بعد } n * \dots * \text{طول بعد } ۲ * \text{طول بعد } ۱ * (\text{نوع آرایه}) * \text{sizeof} = \text{مقدار فضای مورد نیاز آرایه}$$

به عنوان مثال دستور زیر را در نظر بگیرید :

```
int[, ,] b = new int[5, 3, 4];
```

آرایه a ، ۲۴۰ بایت از حافظه را اشغال می کند.

$$a = ۲۴۰ = \text{sizeof}(\text{int}) * ۵ * ۳ * ۴ = \text{مقدار فضای مورد نیاز آرایه } a$$

تعیین مقادیر اولیه برای آرایه های دو بعدی

همانند آرایه های یک بعدی، وقتی آرایه دو بعدی را تعریف می کنید، بسته به نوع عناصر آرایه، مقادیری پیش فرض در عناصر آن قرار می گیرد. اما، می توانید هنگام تعریف آرایه دو بعدی، مقادیر مورد نظرتان را برای عناصر آرایه دو بعدی تعیین کنید. شکل کلی مقدار اولیه دادن به آرایه ها به صورت زیر است :

{مقادیر} = [تعداد عناصر بعد n, ..., تعداد عناصر بعد ۲, تعداد عناصر بعد ۱] = نام آرایه [نوع آرایه, ...]

↑
بعد - ۱

دستور زیر را ببینید :

```
int[, ] a = new int[2, 3] { 1, 2, 3, 4, 5, 6 };
```

این دستور یک آرایه $۳ * ۲$ تعریف کرده و مقادیر آن را مطابق زیر پر می کند :

a		
۱	۲	۳
۴	۵	۶

دستور زیر را در نظر بگیرید :

```
int[, ] b = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 5, 6, 7 }, { 9, 10, 11, 6 } };
```

این دستور یک آرایه $۴ * ۳$ را تعریف کرده مقادیر آن را مطابق شکل زیر پر می کند :

۱	۲	۳	۴
۱	۵	۶	۷
۹	۱۰	۱۱	۶

وقتی آرایه دو بعدی را مقدار اولیه می دهید، نمی توانید کران ها را تعیین کنید. کامپایلر، کران های بالای هر بعد را با توجه به داده های تعیین شده محاسبه می کند. عناصر هر سطر باید در داخل یک {} قرار گیرند و با کاما از هم جدا شوند. هر عنصر در سطر نیز با کاما از عنصر دیگر جدا می گردد.

دستیابی به عناصر آرایه دو بعدی

دستیابی به عناصر آرایه دو بعدی، همانند آرایه های یک بعدی، با استفاده از اندیس ها یا متد GetValue امکان پذیر است. برای دستیابی به آرایه چند بعدی می توان به صورت زیر عمل کرد :

[اندیس بعد n , ... , اندیس بعد ۲ , اندیس بعد ۱] نام آرایه

اندیس ابعاد آرایه از صفر شروع می شود. دستورات زیر را ببینید :

```
int a = b [3 , 4];  
textBox1.text = b [1 , 0].ToString();
```

دستور اول، مقدار عنصر سطر ۳ و ستون ۴ را در متغیر a قرار می دهد و دستور دوم، مقدار عنصر سطر ۱ و ستون صفر را در خاصیت Text کنترل textBox۱ قرار می دهد.

```
int [ , ] a = new int [2 , 3];  
a [1 , 2] = 100;
```

دستور اول آرایه دو بعدی x را تعریف می کند. دستور دوم مقدار عنصر [۲ , ۱] x را برابر ۱۰۰ قرار می دهد.

دستور زیر، مقدار عنصر [۲ , ۱] را بازیابی کرده و در متغیر y قرار می دهد.

```
int y = System.Convert.ToInt32 (x.GetValue (1 , 2));
```

توجه کنید که برای مقدار دادن به عناصر آرایه دو بعدی، نمی توانید از متد SetValue استفاده کنید، زیرا این متد فقط یک مقدار، و یک اندیس را می پذیرد که برای آرایه های یک بعدی مناسب است.

پردازش عناصر آرایه دو بعدی

عناصر آرایه دو بعدی را می توان به صورت سطری یا به صورت ستونی و یا به صورت روش تصادفی مورد پردازش قرار داد. در هر یک از این دو روش از حلقه های تکرار استفاده می شود. آرایه زیر را در نظر می گیریم که حاوی اطلاعات هواشناسی برای سه روز هفته (سطر های آرایه دو بعدی) و چهار ساعت مختلف از هر روز (ستون های آرایه دو بعدی) است :

```
int [, ] temp = new int [, ] { { 15, 20, 10, 30 } ,
                                { 18, 14, 13, 20 },
                                { 11, 13, 19, 10 } };
```

این آرایه ها را در ادامه به دو روش سطری و ستونی، طوری پردازش می کنیم که میانگین دما در هر روز هفته به طور جداگانه (پردازش سطری) و میانگین دما در هر ساعت از سه روز هفته (پردازش ستونی) محاسبه شود.

پردازش سطری آرایه دو بعدی : در این روش ابتدا عناصر سطر اول، سپس عناصر سطر دوم، و غیره پردازش می شوند. دستورات زیر میانگین عناصر هر سطر آرایه temp را محاسبه می کند که پردازش سطری است.

```
//Column process
for (col = 0 ; col <= temp.GetUpperBound (1) ; col++)
{
    sum = 0.0;
    for (row = 0 ; row <= temp.GetUpperBound (0) ; row++)
        sum += temp [row , col];
    ave = sum / temp.GetLength (0);
    Console.WriteLine ("Average col " + col + " : " + ave);
}
```

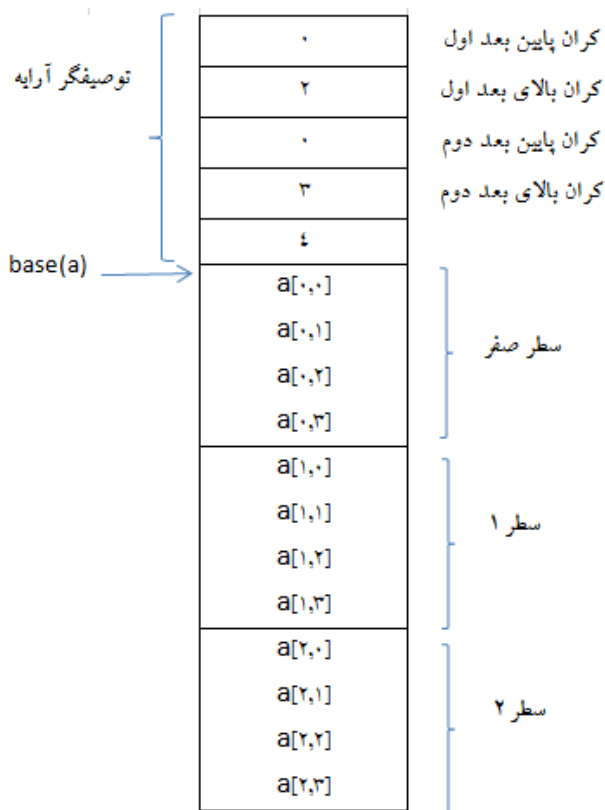
پردازش ستونی آرایه دو بعدی : در این روش پردازش، ابتدا ستون اول، سپس ستون دوم و غیره پردازش می شوند. با توجه به اعلان متغیر های col, row, sum, ave که قبلا دیدید، دستورات زیر آرایه temp را به صورت ستونی پردازش می کند :

```
double ave = 0.0;
double sum;
int row,col;
//Row process
for (row = 0 ; row <= temp.GetUpperBound (0) ; row++)
{
    sum = 0.0;
    for (col = 0 ; col <= temp.GetUpperBound (1) ; col++)
        sum += temp [row , col];
    ave = sum / temp.GetLength (1);
    Console.WriteLine ("Average row " + row + " : " + ave);
}
```

روش تصادفی : در این روش کاربر می تواند به هر عنصر آرایه دستیابی داشته باشد، یعنی در هر محلی از آرایه مقداری قرار دهد و از مقادیر موجود در هر عنصر آرایه استفاده کند.

پیاده سازی آرایه دو بعدی

آرایه های دو بعدی می توانند در حافظه کامپیوتر، به صورت سطری یا ستونی ذخیره شوند. در روش سطری، ابتدا عناصر سطر اول، سپس عناصر سطر دوم و غیره ذخیره می شوند. اما در روش ستونی، ابتدا عناصر ستون اول، سپس عناصر ستون دوم و غیره ذخیره می شوند. برای نمایش آرایه دو بعدی، بخشی از حافظه برای توصیفگر آرایه در نظر گرفته می شود که می تواند شامل کران های پایین و بالای ابعاد آرایه، و طول هر عنصر آرایه باشد. بخش دیگر حافظه مربوط به عناصر آرایه است. تصویر ۲۰ - ۴ آرایه دو بعدی صحیح $a[3,4]$ را پیاده سازی می کند.



تصویر ۲۰ - ۴

مثال ۲۰ - ۴: برنامه ای بنویسید که جدول ضرب اعداد را تولید کرده و در آرایه دو بعدی قرار می دهد؟

حل : ابتدا یک پروژه از نوع Console ایجاد کرده، سپس دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    int[,] table = new int[10, 10];
    int i, j;
    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            table[i, j] = (i + 1) * (j + 1);
    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            Console.Write(table[i, j]);
}
```

مثال ۲۱ - ۴: برنامه ای بنویسید که دو ماتریس 3×3 را دریافت کرده و حاصل ضرب آنها را در یک ماتریس نمایش دهد.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده، سپس دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    int[,] matrix1 = new int[3, 3];
    int[,] matrix2 = new int[3, 3];
    int[,] result = new int[3, 3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            Console.WriteLine("Enter 1st Matrix: ");
            matrix1[i, j] = Convert.ToInt32(Console.ReadLine());
        }
    }
    Console.ReadLine();
    for (int k = 0; k < 3; k++)
    {
        for (int l = 0; l < 3; l++)
        {
            Console.WriteLine("Enter 2nd Matrix: ");
            matrix2[k, l] = Convert.ToInt32(Console.ReadLine());
        }
    }
    Console.WriteLine();
    Console.WriteLine("Matrix 1: ");
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
```

```

Console.Write(matrix1[i, j] + " ");
}
Console.WriteLine();
}
Console.WriteLine();
Console.WriteLine("Matrix 2: ");
for (int k = 0; k < 3; k++)
{
    for (int l = 0; l < 3; l++)
    {
        Console.Write(matrix2[k, l] + " ");
    }
    Console.WriteLine();
}
Console.WriteLine("Matrix 1 * Matrix 2: ");
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        result[i, j] = result[i, j] + matrix1[i, j] * matrix2[i, j];
        Console.Write(result[i, j] + " ");
    }
    Console.WriteLine();
}
Console.ReadLine();
Console.ReadLine(); }
}

```

مثال ۲۲ - ۴: برنامه ای بنویسید که یک ماتریس 3×3 را دریافت کرده و معکوس آن را

چاپ کند؟

حل: یک پروژه از نوع Console ایجاد کرده و دستورات زیر را بنویسید.

```

static void Main(string[] args)
{
    int[ , ] a = new int[3, 3];
    int[ , ] transpose = new int[3,3];

    for(int i=0; i<3; i++)
    {
        for(int j=0; j<3; j++)
        {
            a[i, j] = 0;
            transpose[i, j] = 0;
        }
    }

    Console.WriteLine("Enter the number of rows of Matrix:");
    int r = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter the number of columns of Matrix:");
    int c = int.Parse(Console.ReadLine());
    Console.WriteLine("Matrix A");
}

```

```

for(int i=0; i<r; i++)
{
for(int j=0; j<c; j++)
{
Console.WriteLine("Enter element (row"+ (i + 1) + ", column" + (j + 1) + "):");
a[i, j] = int.Parse(Console.ReadLine());
}
}
for(int i=0; i<c; i++)
{
for(int j=0; j<r; j++)
transpose[i, j] = a[j, i];
}
Console.WriteLine("Matrix Transpose:");
for(int i=0; i<c; i++)
{
for(int j=0; j<r; j++)
Console.Write(transpose[i, j] + "\t");
Console.WriteLine();
}
}

```

مثال ۲۳ - ۴: برنامه ای بنویسید که n را خوانده آرایه ای $n * n$ ایجاد می کند که عناصر هر خانه برابر با شماره سطر * شماره ستون می باشد. سپس جمع سطرها و ستون ها را محاسبه می نماید و نمایش می دهد؟

حل: ابتدا یک پروژه ویندوزی ایجاد کرده و سپس Form مربوط به آن را مطابق پنجره تصویر ۲۱ - ۴ طراحی کنید.



تصویر ۲۱ - ۴

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید. سپس دستورات زیر را

بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt32(textBox1.Text);
    int[,] a = new int[n + 1, n + 1];
    for (int i = 0; i < n; i++)
    {
        a[i, n] = 0;
        for (int j = 0; j < n; j++)
        {
            a[i, j] = (i + 1) * (j + 1);
            a[i, n] = a[i, n] + a[i, j];
        }
    }
    for (int i = 0; i < n; i++)
    {
        a[n, i] = 0;
        for (int j = 0; j < n; j++)
            a[n, i] = a[n, i] + a[j, i];
    }
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            label2.Text = label2.Text + a[i, j].ToString() + " ";
        }
        label2.Text += "\n";
    }
}
```

بر روی دکمه Close کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را

بنویسید.

```
private void button2_Click(object sender, EventArgs e)
{
    Close();
}
```

برنامه را اجرا کرده و نتیجه را مشاهده کنید. دو دستور اول، n را خوانده و آرایه $n + 1$ در n + ۱ را تعریف می کنند. حلقه اول، آرایه را مقدار دهی کرده و مجموع عناصر سطر ها را در ستون آخر قرار می دهد. حلقه دوم، مجموع عناصر ستون ها را محاسبه می نماید و در سطر آخر قرار می دهد و حلقه سوم، عناصر آرایه را بر روی Label۲ نمایش می دهد.

رشته ها

در فصل دوم کمی در مورد رشته ها صحبت کردیم و با متد هایی مانند Replace و SubString آشنا شدیم. در زبان برنامه نویسی C# اعمالی از قبیل مقایسه دو رشته، تبدیل کلیه حروف به رشته به حروف بزرگ و ... را می توان از طریق متد ها انجام داد. برخی از این متد ها در زیر آمده اند :

متد CopyTo : تعدادی کاراکتر از مکان مشخص از یک رشته را در مکان خاص یک آرایه کاراکتری یونیکد کپی می کند. به عنوان مثال دستورات زیر را در نظر بگیرید :

```
string a;
char[] charArray;
a = "Hello Sajad";
a.CopyTo (0, charArray, 0, 5);
```

متد CopyTo، محتویات رشته a را در آرایه charArray کپی می کند. مکان شروع کپی در آرایه صفر و تعداد کاراکتر هایی که باید کپی شوند، ۵ کاراکتر است.

متد CompareTo : دو رشته را با هم مقایسه می کند. اگر رشته اول برابر رشته دوم باشد مقدار صفر، و اگر رشته اول بزرگتر از رشته دوم باشد مقدار ۱، و اگر مقدار ۱- را بر می گرداند. به عنوان مثال دستورات زیر را در نظر بگیرید :

```
string a = "Ali" , b = "Reza" , c = "Ali";
int i = a.CompareTo (b);
int j = a.CompareTo (c);
int k = b.CompareTo (c);
```

دستور اول، مقدار i را برابر ۱- قرار می دهد، زیرا رشته Ali کوچکتر از رشته Reza هست. دستور دوم مقدار j را برابر صفر قرار می دهد، زیرا رشته Ali برابر رشته Ali است و دستور سوم مقدار k را برابر ۱ قرار می دهد، زیرا رشته Reza بزرگتر از رشته Ali است.

متد Equals : دو رشته را مقایسه کرده تعیین می کند که آیا دو رشته با یکدیگر برابر هستند یا خیر، اگر برابر باشند مقدار ۱ و اگر مقدار صفر را بر می گرداند.

```
string a = "Ali" , b = "Reza";
bool result1 = a.Equals ("Ali");
bool result2 = string.Equals (a , b);
```

دستور اول، مقدار ۱ را در متغیر result۱ و دستور دوم مقدار صفر را در متغیر result۲ قرار می دهد.

متد StartWith : تعیین می کند آیا شروع یک رشته با رشته خاصی تطبیق دارد یا خیر، اگر مطابقت داشته باشد، مقدار ۱ و گرنه مقدار صفر را بر می گرداند.

متد StartsWith : تعیین می کند آیا شروع یک رشته با رشته خاصی تطبیق دارد یا خیر، اگر مطابقت داشته باشد مقدار ۱ و گرنه مقدار صفر را بر می گرداند.

```
static void Main(string[] args)
{
    string[] strings = { "started", "Starting", "ended", "ending" };
    string output = "";
    for (int i = 0; i<strings.Length;i++)
        if(strings [i].StartsWith("st"))
            output += "\"" + strings[i]+"\" + " start with \"st\"\\n";
    output += "\\n";
    Console.WriteLine(output);
}
```

با اجرای این دستور خروجی به صورت زیر خواهد بود.

"started" start with "st"

متد EndsWith : تعیین می کند آیا انتهای رشته با رشته معینی برابر است یا خیر. دقیقاً عکس متد StartsWith عمل می کند.

متد IndexOf : اولین مکان وقوع رشته ای را در رشته دیگر تعیین می کند. به عنوان مثال دستورات زیر را ببینید :

```
static void Main(string[] args)
{
    string letters = "abcdefghijklmabcdefghijklm";
    string output = "";
    output += "'c' is located at index " + letters.IndexOf('c');
    output += "\\n 'a' is located at index " + letters.IndexOf('a', 1);
    output += "\\n '$' is located at index" + letters.IndexOf('$', 3, 5);
    System.Console.WriteLine(output);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
'c' is located at index 2
'a' is located at index 13
'$' is located at index-1
```

اولین خروجی، اولین مکان وقوع کاراکتر 'c' را در رشته تعیین می کند. ولی دومین خروجی، مکان اولین وقوع کاراکتر 'a' را از کاراکتر یک به بعد تعیین می کند (اندیس اولین کاراکتر رشته، صفر می باشد). سومین خروجی، مکان اولین وقوع کاراکتر '\$' را از کاراکتر چهارم به طول ۵ کاراکتر تعیین می کند و چون کاراکتر '\$' در این بازه وجود ندارد مقدار ۱- برگردانده می شود.

متد LastIndexOf: مکان آخرین وقوع رشته را در رشته ی دیگر بر می گرداند. به دستورات

زیر توجه کنید :

```
static void Main(string[] args)
{
    // The string we are searching.
    string value = "Dot Net Perls";
    // Find the last occurrence of N.
    int index1 = value.LastIndexOf('N');
    if (index1 != -1)
    {
        Console.WriteLine(index1);
        Console.WriteLine(value.Substring(index1));
    }
    // Find the last occurrence of this string.
    int index2 = value.LastIndexOf("Perls");
    if (index2 != -1)
    {
        Console.WriteLine(index2);
        Console.WriteLine(value.Substring(index2));
    }
}
```

خروجی دستورات زیر به صورت زیر می باشد :

```
4
Net Perls
8
Perls
```

متد IndexOfAny: مکان اولین وقوع رشته ای را در تعدادی کاراکتر از نوع یونیکد تعیین

می کند. به عنوان مثال دستورات زیر را در نظر بگیرید :

```
static void Main(string[] args)
{
    // A.
    // Input.
    const string value1 = "Darth is my enemy.";
    const string value2 = "Visual Basic is hard.";
    // B.
    // Find first location of 'e' or 'B'.
    int index1 = value1.IndexOfAny(new char[] { 'e', 'B' });
    Console.WriteLine(value1.Substring(index1));
    // C.
    // Find first location of 'e' or 'B'.
    int index2 = value2.IndexOfAny(new char[] { 'e', 'B' });
    Console.WriteLine(value2.Substring(index2));
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

enemy.
Basic is hard.

متد LastIndexOfAny : مکان آخرین وقوع رشته ای را در تعدادی کاراکتر از نوع یونیکد تعیین می کند. به عنوان مثال دستورات زیر را در نظر بگیرید :

```
static void Main(string[] args)
{
    // Input string.
    const string value = "ccbbddeee";
    // Search for last of any of these characters.
    int index = value.LastIndexOfAny(new char[] { 'd', 'b' });
    Console.WriteLine(index);
}
```

خروجی دستورات بالا برابر با ۵ خواهد بود.

متد ToUpper : کلیه حروف کوچک رشته را به حروف بزرگ تبدیل می کند.
متد ToLower : کلیه حروف بزرگ رشته را به حروف کوچک تبدیل می کند.

```
static void Main(string[] args)
{
    // Uppercase this mixed case string.
    string value1 = "Lowercase string.";
    string upper1 = value1.ToUpper();
    Console.WriteLine(upper1);
    //*****
    // Input string
    string mixedCase = "This is a MIXED case string.";
    // Call ToLower instance method, which returns a new copy.
    string lower = mixedCase.ToLower();
    // Display results
    Console.WriteLine("{0}, {1}",mixedCase,lower);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

LOWERCASE STRING.
This is a MIXED case string., this is a mixed case string.

متد Trim : فضای خالی سمت چپ و راست رشته را حذف می کند. به عنوان مثال دستورات

زیر را در نظر بگیرید.

```
static void Main(string[] args)
{
    string st = " This is an example string. ";
    st = st.Trim();
    Console.WriteLine(st);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

This is an example string.

متد TrimEnd : فضای خالی انتهای رشته را حذف می کند.

متد TrimStart : فضای خالی ابتدای رشته را حذف می کند.

متد Remove : بخشی از رشته را حذف می کند. به عنوان مثال دستورات زیر را در نظر

بگیرید :

```
static void Main(string[] args)
{
    // 1. Remove all characters after an index.
    string test1 = "0123456";
    string result1 = test1.Remove(3);
    Console.WriteLine(result1);
    // 2. Remove range of characters in string.
    string test2 = "012 345 678";
    int index1 = test2.IndexOf(' ');
    int index2 = test2.IndexOf(' ', index1 + 1);
    string result2 = test2.Remove(index1, index2 - index1);
    Console.WriteLine(result2);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

012
012 678

متد Insert : رشته ای را در مکان خاصی از رشته دیگر اضافه می کند. به عنوان مثال

دستورات زیر را در نظر بگیرید.

```
static void Main(string[] args)
{
    // A. You can insert one string between two others.
    string names = "Romeo Juliet";
    string shakespeare = names.Insert(6, "and ");
    Console.WriteLine(shakespeare);
    // B. You can insert a string right after another string.
    string names2 = "The Taming of Shrew";
    int index2 = names2.IndexOf("of ");
    string shakespeare2 = names2.Insert(index2 + "of ".Length, "the ");
    Console.WriteLine(shakespeare2);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

Romeo and Juliet
The Taming of the Shrew

متد `PadLeft` : با افزودن فضای خالی یا کاراکتر یونیکد خاصی به سمت چپ، رشته کاراکتری را از سمت راست تنظیم می کند. به عنوان مثال دستورات زیر را در نظر بگیرید :

```
static void Main(string[] args)
{
    string[] words = { "1", "200", "Behnam", "Rasol" };
    foreach (string word in words) // Loop over words
    {
        Console.WriteLine(word.PadLeft(10, '_')); // Write ten characters
        Console.WriteLine(word.PadLeft(10)); // Write ten characters with space
    }
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
_____ 1
_____ 1
_____ 200
_____ 200
_____ behnam
_____ behnam
_____ Rasol
_____ Rasol
```

متد `PadRight` : با افزودن فضای خالی یا کاراکتر یونیکد خاصی به سمت راست، رشته کاراکتری را از سمت چپ تنظیم می کند و عملکرد این تابعی از لحاظ ساختار برنامه نویسی شبیه متد `PadLeft` می باشد.

متد `Split` : رشته ای را با استفاده از یک جدا کننده به چند زیر رشته تبدیل می کند. به عنوان

مثال دستورات زیر را در نظر بگیرید :

```
static void Main(string[] args)
{
    string s = "He is a programmer";
    // Split string on spaces.
    // ... This will separate all the words.
    string[] words = s.Split(' ');
    foreach (string word in words)
    {
        Console.WriteLine(word);
    }
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
He
is
a
programer
```

متد Format : برای تعیین فرمت نمایش اطلاعات به کار می رود. این متد با کلاس string استفاده می شود. یعنی برای نمونه ایجاد شده از کلاس string نمی توان از این متد استفاده کرد. برای استفاده از متد Format باید ساختار آن را بشناسیم. هر Format دارای سه بخش زیر است :

`{index [, alignment] : formatstring}}`

بخش index : یک بخش عددی اجباری است که از صفر شروع می شود. این بخش Format عنصری را در لیستی از مقادیر تعیین می کند. اگر مقدار این بخش صفر باشد، Format اولین مقدار لیست، اگر مقدار این بخش یک باشد، Format دومین مقدار لیست را تعیین می کند و همین طور ادامه دارد. در پارامتر می توانید از Format های چندگانه استفاده کنید. به عنوان مثال Format های `{0:X}`, `{0:E}`, `{0:N}` به ترتیب Format های مقادیر عددی مبنای ۱۶، اعداد علمی و اعداد دهدهی را تعیین می کنند.

هر Format می تواند به هر پارامتری نسبت داده شود. به عنوان مثال، اگر سه پارامتر وجود داشته باشند، Format دومین، اولین و سومین مقدار را به صورت زیر تعیین می کند.

`"{1} {0} {2}"`

بخش alignment : یک عدد اختیاری است که طول فیلد و چگونگی تنظیم آن را تعیین می کند. اگر مقدار آن کوچکتر از طول رشته ای باشد که Format برای آن در نظر گرفته شده است نادیده گرفته می شود. اگر مقدار آن مثبت باشد تنظیم داده از سمت راست انجام می شود ولی اگر منفی باشد تنظیم داده از سمت چپ انجام می گیرد. این پارامتر توسط کاما از پارامتر index جدا می شود. بخش formatstring : پارامتری اختیاری است که شامل Format استاندارد یا سفارشی می باشد. اگر این پارامتر ذکر نشود Format عمومی ("G") استفاده می شود. اگر پارامتر formatstring ذکر شود باید با دو نقطه کولن (:) از بخش قبلی جدا شود. برخی از مقادیر formatstring در جدول ۲-۴ آماده است.

به عنوان مثال و برای درک بهتر، دستورات زیر را در نظر بگیرید :

```
static void Main(string[] args)
{
    // Declare three variables that we will use in the format method.
    // ... The values they have are not important.
    string value1 = "Dot Net Perls";
    int value2 = 10000;
    DateTime value3 = new DateTime(2007, 11, 1);
    // Use string.Format method with four arguments.
    // ... The first argument is the formatting string.
    // ... It specifies how the next three arguments are formatted.
    string result = string.Format("{0}: {1:0.0} - {2:yyyy}", value1, value2, value3);
    // Write the result.
    Console.WriteLine(result);
}
```

خروجی دستورات به صورت زیر خواهد بود.

Dot Net Perls: 10000.0 - 2007

مقدار	هدف	مقدار	هدف
C	مقادیر ارزی	T	زمان بلند
D	مقادیر عددی دهدهی	f	زمان کوتاه / تاریخ کامل
E	مقادیر نمایی	F	زمان بلند / تاریخ کامل
F	مقادیر اعشاری	g	زمان کوتاه / تاریخ عمومی
G	مقادیر عمومی	G	زمان بلند / تاریخ عمومی
N	مقادیر عددی	M	ماه
P	درصد	R	RFC ۳۳
R	Round-trip	s	sortable
X	مقادیر مبنای ۱۶	u	universal sortable
d	تاریخ کوتاه	U	universal sortable
D	تاریخ بلند	y	سال
t	زمان کوتاه		

برخی از کاراکترهای کنترلی وجود دارد که در کار با رشته ها به کار می رود و با \ شروع می شود که این کاراکتر ها را در جدول ۳ - ۴ می توانید مشاهده کنید.

\a	بوق سیستم را به صدا در می آورد
\b	کاراکتر BackSpace را تعیین می کند
\t	به اندازه یک Tab (هشت کاراکتر) جای خالی بوجود می آورد
\r	کاراکتر Enter را تعیین می کند
\f	کاراکتر Form Feed را تعیین می کند
\v	مکان نما به هشت سطر بعد انتقال می یابد
\n	مکان نما را به خط جدید می برد
\e	کاراکتر Escape را تعیین می کند
\xxx	یک کاراکتر اسکی در مبنای هشت بوجود می آورد
\xxx	یک کاراکتر مبنای ۱۶ را بوجود می آورد
\cx	یک کاراکتر کنترلی اسکی است
\uxxx	یک کاراکتر کنترلی یونیکد در مبنای ۱۶ را نشان می دهد
\\	کاراکتر \ را تعیین می کند
\"	کاراکتر " را تعیین می کند
\'	کاراکتر ' را تعیین می کند.

جدول ۳ - ۴

کلاس StringBuilder

کلاس String فضایی ایستا برای رشته ها در نظر می گیرد، به طوری که ظرفیت رشته ها قابل تغییر نیستند. عملگرهایی که بر روی رشته ایجاد شده و با کلاس String کار می کنند، عبارتند از: = و +=.

عملگر +=، که یک رشته جدید ایجاد می کرد به طوری که یک رشته را به انتهای رشته دیگر اضافه می نمود. ولی کلاس StringBuilder برای ایجاد اطلاعات رشته به صورت پویا به کار می رود. این کلاس در فضای نام System.Text قرار دارند. اگر سازنده کلاس را بدون آرگومان فراخوانی کنید، هیچ کاراکتری در رشته قرار نمی دهد و ظرفیت آن را ۱۶ کاراکتر در نظر می گیرد.

دستورات زیر را ببینید :

```
StringBuilder buffer1 = new StringBuilder();  
string output = "buffer1 is" + buffer1.ToString();  
output += "\nCapacity is" + buffer1.Capacity.ToString();
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
buffer1 is  
Capacity is 16
```

اگر برای سازنده کلاس StringBuilder یک آرگومان از نوع int تعیین شود، ظرفیت آن توسط این آرگومان تعیین می شود و هیچ کاراکتری در آن قرار نمی گیرد. دستورات زیر را

```
StringBuilder buffer1 = new StringBuilder(10);  
string output = "buffer1 is" + buffer1.ToString();  
output += "\nCapacity is" + buffer1.Capacity.ToString();  
Console.WriteLine(output);
```

ببینید :

این دستورات، buffer1 را به ظرفیت ۱۰ کاراکتر در نظر می گیرد و هیچ کاراکتری در آن قرار نمی دهد. با اجرای این دستورات خروجی زیر ظاهر می شود.

```
buffer1 is  
Capacity is 10
```

اگر یک آرگومان رشته ای را برای سازنده StringBuilder تعیین کنید، و طول رشته کوچکتر از ۱۶ کاراکتر باشد، ظرفیت آن ۱۶ کاراکتر در نظر گرفته می شود، ولی اگر طول رشته بیش از ۱۶ کاراکتر باشد، ظرفیت آن، اولین توان ۲ بزرگتر از تعداد کاراکتر های آرگومان در نظر گرفته می شود. به عنوان مثال دستورات زیر را ببینید :

```
StringBuilder buffer1 = new StringBuilder("abcdefgh");  
string output = "buffer1 is" + buffer1.ToString();  
output += "\nCapacity is" + buffer1.Capacity.ToString();  
StringBuilder buffer2 = new  
StringBuilder("abcdefghabcdefghabcdefghabcdefghabcdefghabcdefgh");  
output += "\nbuffer2 is" + buffer2.ToString();  
output += "\nCapacity is" + buffer2.Capacity.ToString();  
Console.WriteLine(output);
```

با اجرای دستورات بالا خروجی زیر ظاهر خواهد شد :

```
buffer1 is abcdefgh  
Capacity is 16  
buffer2 is abcdefghabcdefghabcdefghabcdefghabcdefghabcdefgh  
Capacity is 64
```

برخی از خواص و متد های کلاس StringBuilder در زیر آمده است :

خاصیت Length : طول شیء کلاس StringBuilder را مشخص می کند.

خاصیت Capacity : ظرفیت شیء کلاس StringBuilder را تعیین می کند.

خاصیت MaxCapacity : حداکثر ظرفیت شیء StringBuilder را مشخص می کند. به

عنوان مثال دستورات زیر را ببینید :

```
StringBuilder buffer1 = new StringBuilder ("abcd");
string output = "buffer1 is" + buffer1.ToString();
output += "\nCapacity is" + buffer1.Capacity.ToString();
output += "\nLength is" + buffer1.Length.ToString();
output += "\nMaxCapacity is" + buffer1.MaxCapacity.ToString();
Console.WriteLine (output);
```

دستورات بالا مقدار، ظرفیت، طول و حداکثر ظرفیت buffer1 را تعیین می کند. خروجی

دستورات بالا به صورت زیر خواهد بود :

```
buffer1 is abcd
Capacity is 16
Length is 4
MaxCapacity is 2147483647
```

متد Append : این متد دارای ۱۹ نوع سازنده همنام است که اجازه می دهد انواع مختلف داده

ها به انتهای شیء StringBuilder اضافه شوند.

```
const string s = "Value";
StringBuilder builder = new StringBuilder();
for (int i = 0; i < 1000; i++)
{
    builder.Append("One string ").Append(s).Append("Another string");
}
```

متد AppendFormat : یک رشته را به Format تعیین شده تبدیل و به انتهای شیء ای از نوع

StringBuilder اضافه می کند. به عنوان مثال دستورات زیر را ببینید :

```
static void Main(string[] args)
{
    StringBuilder builder = new StringBuilder();
    string[] data = { "I", "a", "Programer" };
    int counter = 0;
    foreach (string value in data)
    {
        builder.AppendFormat("You have visited {0} ({1}).\n",value,++counter);
    }
    Console.WriteLine(builder);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
You have visited I <1>.
You have visited a <2>.
You have visited Programmer <3>.
```

متد AppendLine : این متد یک رشته را به انتهای شیء ای از نوع StringBuilder اضافه کرده و به خط جدید می رود.

```
static void Main(string[] args)
{
    StringBuilder builder = new StringBuilder();
    builder.AppendLine("One");
    builder.AppendLine();
    builder.AppendLine("Two").AppendLine("Three");
    // Display.
    Console.Write(builder);
    // AppendLine uses \r\n sequences.
    Console.WriteLine(builder.ToString().Contains("\r\n"));
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
One

Two
Three
True
```

متد Remove : برای حذف از هر مکان شیء StringBuilder به کار می رود. این متد دو آرگومان دارد، آرگومان اول، مکان شروع حذف را تعیین می کند و آرگومان دوم، تعداد کاراکترهایی که باید حذف شوند را تعیین می کند.

```
static void Main(string[] args)
{
    StringBuilder builder = new StringBuilder("Dot Net for you");
    builder.Remove(4, 3);
    Console.WriteLine(builder);
}
```

خروجی دستورات بالا به صورت زیر خواهد بود.

```
Do for you
```

متد Insert : اجازه می دهد انواع مختلف داده به هر مکان شیء ای از نوع StringBuilder اضافه شوند. این متد، مکان اضافه کردن داده را به عنوان آرگومان دریافت می کند.

متد Replace : برای جایگزینی یک رشته با رشته دیگر یا یک کاراکتر با کاراکتر دیگر در نوع StringBuilder به کار می رود.

```
static void Main(string[] args)
{
    StringBuilder builder = new StringBuilder("This is an example string that is an example.");
    builder.Replace("an", "the"); // Replaces 'an' with 'the'.
    Console.WriteLine(builder.ToString());
    Console.ReadLine();
}
```

خروجی دستورات بالا به صورت زیر می باشد.

This is the example string that is the example.

مثال ۲۴ - ۴: برنامه ای که اعمال زیر را بر روی داده ای از نوع کلاس StringBuilder

انجام می دهد.

. دو رشته دریافتی را به انتهای داده اضافه می کند.

. دو عدد صحیح را به انتهای داده اضافه می کند.

. دو داده منطقی را به انتهای داده اضافه می کند.

. رشته ای را به داده اضافه می کند. مکان اضافه شدن رشته، عددی است که در textBox۱

وارد شده است و رشته ای که باید اضافه شود در textBox۲ قرار می گیرد.

. داده ای را از داده StringBuilder حذف می کند. مکان شروع حذف و تعداد کاراکترهایی

که باید حذف شوند، توسط کنترل های textBox۱ و textBox۲ تعیین می شوند.

. داده را جایگزین داده دیگر در نوع StringBuilder می کند. به جای داده وارد شده در

textBox۱ باید داده ورودی textBox۲ جایگزین شود.

حل : ابتدا یک پروژه جدید ویندوزی جدید و Form آن را مطابق پنجره تصویر ۲۱ - ۴

طراحی می کنیم.



تصویر ۲۱ - ۴

ابتدا در بخش public partial دستورات زیر را تایپ کنید.

```
public partial class Form1 : Form
```

```
{
    StringBuilder sBuilder = new StringBuilder();
    string output;
```

حال بر روی دکمه String دابل کلیک کرده و در رویداد Click آن کدهای زیر را تایپ کنید:

```
private void button1_Click(object sender, EventArgs e)
{
    sBuilder.Append(textBox1.Text);
    sBuilder.Append(" ");
    sBuilder.Append(textBox2.Text);
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

اکنون بر روی دکمه Integer دابل کلیک کرده و در رویداد Click آن دستورات زیر را تایپ

```
private void button2_Click(object sender, EventArgs e)
{
    sBuilder.Append(Convert.ToInt32(textBox1.Text));
    sBuilder.Append(" ");
    sBuilder.Append(Convert.ToInt32(textBox2.Text));
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

کنید.

بر روی دکمه Boolean دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.

```
private void button3_Click(object sender, EventArgs e)
{
    sBuilder.Append(Convert.ToBoolean(textBox1.Text));
    sBuilder.Append(" ");
    sBuilder.Append(Convert.ToBoolean(textBox2.Text));
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

بر روی دکمه Insert دابل کلیک کرده و در رویداد Click آن کدهای زیر را تایپ کنید:

```
private void button4_Click(object sender, EventArgs e)
{
    sBuilder.Insert(Convert.ToInt32(textBox1.Text), textBox2.Text);
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

حال بر روی دکمه Remove دابل کلیک کرده و در رویداد Click آن کد های زیر را تایپ کنید :

```
private void button5_Click(object sender, EventArgs e)
{
    sBuilder.Remove(Convert.ToInt32(textBox1.Text), Convert.ToInt32(textBox2.Text));
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

حال بر روی دکمه Replace دابل کلیک کرده و در رویداد Click آن کد های زیر را تایپ کنید :

```
private void button6_Click(object sender, EventArgs e)
{
    sBuilder.Replace(textBox1.Text, textBox2.Text);
    sBuilder.Append(" ");
    output = sBuilder.ToString();
    label4.Text = output;
}
```

برنامه را ابتدا ذخیره و سپس اجرا کنید. ابتدا داده های Ali و Reza را وارد کرده و دکمه String را کلیک کنید. سپس مقادیر ۱ و ۲ را وارد کرده دکمه Integer را کلیک نمایید. در ادامه مقادیر true و false را وارد کرده و دکمه Boolean را کلیک کنید. برای درج اطلاعات عدد ۲ و Boolean را وارد کرده و دکمه Insert را کلیک کنید. برای حذف اطلاعات عدد ۳ و ۴ را وارد کرده و دکمه Remove را کلیک کنید. برای جایگزینی اطلاعات، true و false را وارد کرده دکمه Replace را کلیک کنید.



فصل پنجم

برنامه نویسی شیء گرا در زبان C#.NET

مقدمه ای بر شیء گرایی

اولین نرم افزار برای نخستین رایانه ها، زنجیره ای از صفر و یک ها بود که فقط عده اندکی از این توالی سر در می آوردند. به تدریج کاربرد رایانه گسترش یافت و نیاز بود تا نرم افزار های بیشتری ایجاد شود. برای این منظور برنامه نویسان مجبور بودند با انبوهی از صفر ها و یک ها سر و کله بزنند و این باعث می شد مدت زمان زیادی برای تولید یک نرم افزار صرف شود. از این گذشته اگر ایرادی در کار برنامه یافت می شد پیدا کردن محل ایراد و رفع آن بسیار مشکل و طاقت فرسا بود. ابداع زبان اسمبلی جهش بزرگی به سوی تولید نرم افزار های کارآمد بود.

اسمبلی قابل فهم تر بود و دنبال کردن برنامه را سهولت می بخشید. سخت افزار به سرعت رشد می کرد و این رشد به معنی نرم افزار های کامل تر و گسترده تر بود. کم کم زبان اسمبلی هم جوابگوی شیوه های نوین تولید نرم افزار نبود. هشت خط کد اسمبلی برای یک جمع ساده به معنای ده هزار خط کد برای یک برنامه حسابداری روزانه است. این بار، انبوه کد های اسمبلی مشکل ساز شدند. (زبان های سطح بالا) دروازه های تمدن جدید در دنیای نرم افزار را به روی برنامه نویسان گشودند.

زبان های سطح بالا دو نشان درخشان از ادبیات و ریاضیات را همراه خود آوردند : اول، دستوراتی شبیه زبان محاوره ای که باعث شدند برنامه نویسان از دست کد های تکراری و طولی اسمبلی خلاص شوند و دوم، مفهوم تابع که سرعت تولید و عیب یابی نرم افزار را چند برابر کرد. اساس کار، این گونه بود که وظیفه اصلی برنامه به وظایف کوچکتری تقسیم می شد و برای انجام دادن هر وظیفه، تابعی نوشته می شد. پس این ممکن بود که توابع مورد نیاز یک برنامه به طور همزمان نوشته و آزمایش شوند و سپس همگی در کنار هم چیده شوند. دیگر لازم نبود قسمتی از نرم افزار، منتظر تکمیل شدن قسمت دیگری بماند. همچنین عیب یابی نیز آسان صورت می گرفت و به سرعت محل خطا یافت شده و اصلاح می شد. علاوه بر این، برای بهبود دادن نرم افزار موجود یا افزودن امکانات اضافی به آن، دیگر لازم نبود که برنامه از نو نوشته شود و فقط توابع مورد نیاز را تولید کرده یا بهبود می دادند و آن را به برنامه موجود پیوند می زدند. از این به بعد بود که گروه های تولید نرم افزاری برای تولید نرم

افزار های بزرگ ایجاد شدند و بحث مدیریت پروژه های نرم افزاری و شیوه های تولید نرم افزار و چرخه حیات و... مطرح شد.

نرم افزار های بزرگ، تجربیات جدیدی به همراه آوردند و برخی از این تجربیات نشان می داد که توابع، چندان هم بی عیب نیستند. برای ایجاد یک نرم افزار، توابع زیادی نوشته می شد که اغلب این توابع به یکدیگر وابستگی داشتند. اگر قرار می شد ورودی یا خروجی یک تابع تغییر کند، سایر توابعی که با تابع مذکور در ارتباط بودند نیز باید شناسایی می شدند و به تناسب تغییر می نمودند. این موضوع، اصلاح نرم افزار ها را مشکل می کرد. علاوه بر این، اگر تغییر یک تابع مرتبط فراموش می شد، صحت کل برنامه به خطر می افتاد. این اشکال ها برای مدیران و برنامه نویسان بسیار جدی بود. بنابراین باز هم متخصصین به فکر راه چاره افتادند. پس از ریاضیات و ادبیات، این بار نوبت فلسفه بود.

((شیء گرایی)) رهیافت جدیدی بود که برای مشکلات ذکر شده راه حل داشت. این مضمون از دنیای فلسفه به جهان برنامه نویسی آمد و کمک کرد تا معضلات تولید و پشتیبانی نرم افزار کم تر شود. برنامه نویسی شیء گرا، برترین نظریات برنامه نویسی تابعی را انتخاب کرده و با مفاهیم جدیدی در هم می آمیزد که نتیجه آن، ابداع روشی متفاوت و موثر در تولید نرم افزار است. روشی که پیچیدگی های برنامه نویسی را کم می کند و نگهداری و پشتیبانی از آن را آسان می نماید.

در اغلب موارد یک برنامه را می توان به دو روش شکل داد : روش کد محور و روش داده محور.

در روش کد محور، نرم افزار با این تفکر شکل می گیرد که ترتیب منطقی اتفاقات در برنامه مشخص می شود و سپس برنامه به شکل پیمانه ای به ترتیب به این اتفاقات پاسخ می دهد. به بیان ساده، در روش کد محور ابتدا مشخص می کنید که برنامه چه کار هایی باید انجام دهد و سپس برای هر کار تابع مناسبی تهیه می کنید و برنامه را با ترکیب این توابع به پایان می برید. این همان روشی است که در زبان های سطح بالای پیمانه ای مثل C یا Pascal به کار می رفت. اما در روش داده محور، پی ریزی نرم افزار بر اساس شناخت داده های مورد نیاز و مسیر پردازش آن است. یعنی قبل از هر چیز مشخص می کنید که چه داده هایی قرار است

وارد نرم افزار شوند و این داده ها چه مسیری را قرار است طی کنند و چگونه با داده های دیگر تراکنش داشته باشند. وقتی چنین نقشه ای از نرم افزار تهیه شد داده ها را در قالب اشیاء در آورده و پردازش لازم برای انتقال داده ها را نیز به وسیله کلاس ها انجام می دهید و برنامه پایان می یابد.

برنامه نویسی شیء گرا از روش داده محور برای تولید نرم افزار استفاده می کند. ممکن است روش داده محور به نظر پیچیده و عجیب بیاید اما زمانی که این نظریه مطرح شد برنامه ها به اندازه ای بزرگ و پیچیده بودند که روش داده محور توانست گشایشی در این راه حاصل کند. امروزه بدون استفاده از شیء گرایی تولید نرم افزار های رایج کنونی تقریباً ناممکن است.

پیدایش شیء گرایی

برنامه نویسی شیء گرا یا OOP (Object Oriented Programing) در اوایل دهه ۱۹۷۰ توسط Alan Kay طراحی شده است. یعنی او اولین قدم های این سبک برنامه نویسی را برداشته شده است. اولین زبان شیء گرا توسط Alan Kay طراحی شد. عنوان این زبان Small Talk است.

Alan Kay گفته بود آن چیزی که باعث شد این فکر به ذهنم برسد نحوه عملکرد سلول های زیست محیطی بود. یعنی این سبک برنامه نویسی از روی سلول های جانداران الگو برداری شده است. آن چیزی که باعث شد که Alan Kay از روی سلول های جانداران الگو برداری کند نحوه زندگی سلولها بود :

هر سلول نمونه ای از اصل است و هر خصوصیتی که دارد از اصل خود به ارث برده است (ژنتیک سلول).

هر سلول رفتارهایی دارد که از اصل خود به ارث برده است.

سلول ها همگی مستقل از هم زندگی می کنند و براساس ارسال پیام های شیمیایی با یکدیگر ارتباط برقرار می کنند. ارسال پیام به این صورت است که پیام از پوسته یکی خارج و به پوسته دیگری وارد می شود.

سلولها می توانند از یکدیگر متمایز شوند.

با توجه به گفته های بالا می توان متوجه شد که همان مشخصه کلاس ها را بیان می کند یعنی هر شی از یک کلاسی تشکیل شده که ویژگی های آن کلاس را با خودش به ارث برده است. همانطور که می دانیم اشیاء با یکدیگر ارتباط برقرار می کنند. نحوه ارتباط یا فرستادن پیام در اشیاء هنگام فراخوانی رفتار ها در یک رویداد است. هر شیء خودش یک شناسنامه یا Identifier دارد که ویژگی های آن شیء را بیان می کند. Small Talk مانند سلول های جاندار عمل می کند. یعنی Alan Kay در تمامی قسمت های این زبان تعیین کرده بود که اشیاء با هم ارتباط برقرار می کنند و دارای شناسنامه ای هستند و همچنین مستقل از همدیگر کار می کنند.

اصول اولیه ای که Alan Kay برای برنامه نویسی شیء گرا تعیین کرده بود به قرار زیر هستند : هر چیزی یک شیء است.

هر برنامه ای شامل اشیاء است که اشیاء با ارسال پیام به یکدیگر تعیین می کنند که چه کاری باید الان انجام شود.

هر شیء یک حافظه برای خودش دارد که بتوان به وسیله آن اشیای دیگر را ساخت.

هر شیء خودش نمونه ای از یک کلاس است.

زبان های شیء گرا

از ابتدای بوجود آمدن برنامه نویسی شیء گرا دو نوع از زبان های برنامه نویسی برای این سبک برنامه نویسی وجود داشته اند. نوع اول زبان های خاص مانند Smalltalk. که در این زبان ها برنامه نویس برای هر کاری بایستی از اصول برنامه نویسی شیء گرا پیروی کند حتی برای ساده ترین کار ها مانند جمع دو عدد. نوع دوم از زبان های برنامه نویسی شیء گرا که بیشتر رواج دارند، زبان های مختلط هستند. در این نوع زبان های برنامه نویسی، برنامه نویس مختار است از امکانات برنامه نویسی شیء گرا استفاده کند یا نه. برای مثال می توان به زبان هایی مانند ++C، Java اشاره کرد.

برنامه نویسی شیء گرا شیوه نوینی است که در آن می توان قطعاتی را ایجاد کرد و در برنامه های مختلف مورد استفاده قرار داد. قابلیت خوانایی برنامه هایی که در این روش نوشته می

شوند بالا بوده، تست، عیب یابی و اصلاح آن ها آسان است. شیء گرایی بر اشیا تاکید دارد، در برنامه نویسی شیء گرا اشیاء به صورت انتزاع مطرح می شوند. انتزاع به آن چیزی می گویند که شما در مورد آن فکر می کنید و در یک دید کلی مطرح می کنید. مثلاً وقتی به یک دانه شن فکر می کنید فکرتان به سمت ساحل می رود، یا وقتی به یک درخت فکر می کنید ذهن تان به سمت جنگل متمرکز می شود به این انتزاع می گویند که در این سبک برنامه نویسی مطرح می شود. اشیاء با توجه به کلاس های خودشان ساخته می شوند که خود کلاس ها ممکن است که از کلاس های دیگری مشتق شده باشند.

فواید برنامه نویسی شیء گرا

برنامه نویسی شیء گرا اهدافی را در توسعه نرم افزار دنبال می کند. OOP کوشش می کند تا نرم افزاری را تولید نماید که در برگرفته مشخصه های زیر باشد :

۱- سادگی یا طبیعی بودن (Natural) : OOP برنامه های طبیعی تولید می کند. برنامه های ساده بیشتر قابل فهم هستند. به جای آن که برنامه نویسی را عبارتی همچون آرایه ای از نواحی حافظه بدانیم، می توانید برنامه نویسی را مجموعه اصطلاحات مسئله مشخص خودتان بدانید. نیازی نیست که در جزئیات کامپیوتر غوطه ور شوید تا برنامه را طراحی کنید. برنامه نویسی شیء گرا اجازه می دهد که مسئله را در سطح تابعی مدل کنید نه در سطح پیاده سازی. بنابراین نیازی نیست که بدانید قسمت هایی از نرم افزار چگونه کار می کنند تا از آن استفاده نمایید، لذا می توانید تنها بر روی آنچه که انجام می دهید متمرکز شوید.

۲- پایداری (Reliability) : برای ایجاد نرم افزار هایی مفید نیاز دارید که نرم افزاری بسازید که همچون دیگر محصولات مانند یخچال و تلویزیون از پایداری خوبی برخوردار باشند. ذات ماژولار بودن اشیاء اجازه می دهد که قسمت هایی از برنامه را بدون آن که دیگر اجزاء تحت تاثیر قرار گیرند، تغییر دهید. در واقع اشیاء حالت (وضعیت فعلی) و وظایف شان را از دیگر اشیاء ایزوله می کنند.

یکی از راه های افزایش پایداری، از طریق آزمایش کامل به دست می آید. شیء گرایی نحوه آزمایش را از طریق ایزوله کردن اشیاء از یکدیگر، بهبود می بخشد. این ایزولاسیون اجازه می

دهد هر جز را به طور مستقل مورد آزمایش قرار دهید. زمانی که یک جزء را آزمایش کرده و از آن اطمینان حاصل کردید می توانید از آن با اعتماد کامل استفاده نمایید.

۳- قابلیت استفاده مجدد (Reusability): در برنامه نویسی شیء گرا می توان از کدی که قبلا و حتی توسط فرد دیگری ایجاد شده استفاده کرد. حتی می توان کتابخانه ای از اشیای مفید و لازم در برنامه نویسی را فراهم کرد و در برنامه نویسی از آنها استفاده کرد. یعنی برنامه نویس بدون آنکه نیاز داشته باشد برای هر بار استفاده از یک سری کد در برنامه اش آن را تایپ کند می تواند آن کد را در درون یک شیء تعبیه نماید و از آن شیء در برنامه های خود استفاده کند.

آیا بنا هر بار که می خواهد خانه ای را بنا کند، آجر جدیدی اختراع می کند؟ آیا مهندس الکترونیک هر بار که می خواهد مداری طراحی کند، مقاومت جدیدی را اختراع می کند؟ پس چرا برنامه نویسان می خواهند (دوباره چرخ را اختراع کنند؟) زمانی که مسئله ای یک بار حل شد، باید راه حل را دوباره استفاده کرد. می توانید به راحتی از کلاس های شیء گرا دوباره استفاده کنید. مثل ماژول ها می توانید از اشیاء در برنامه های مختلف استفاده مجدد کنید. بر خلاف ماژول ها، OOP وراثت را معرفی می کند که از این طریق می توان اشیاء موجود را توسعه داده و از طریق چند شکلی می توانید کد های عمومی و جامعی بنویسید. شیء گرایی کد جامع را تضمین نمی کند.

ایجاد کلاس های با طراحی خوب اندکی مشکل است و نیاز به توجه و تمرکز بر روی تجزیه دارد. برنامه نویسان عموما این روش را روشی نمی یابند. از طریق OOP می توانید ایده های کلی را مدل کنید و از آن ایده ها جهت حل مسائل مشخص بهره بجوید. همچنین اشیاء ساخته می شوند تا مسئله مشخص را حل کنند. عموما برای ساخت این اشیاء مشخص از کد های جامع و عمومی استفاده می شود.

۴- قابلیت نگهداری (Maintainability): چرخه حیات یک برنامه پس از ارائه آن به بازار به پایان نمی رسد. در عوض، باید نگهداری و پشتیبانی از کد های خود را ادامه دهید. در حقیقت ۶۰ تا ۸۰ درصد زمانی که بر روی یک برنامه صرف می شود، صرف نگهداری آن می گردد. توسعه نرم افزار و نوشتن کد های آن ۲۰ درصد این معادله را شامل می شود. کد های

شیء گرا با طراحی خوب قابلیت نگهداری خوبی دارند. برای بر طرف کردن خطایی در برنامه، کافی است تنها به یک قسمت از آن مراجعه کنید. از آنجا که پیاده سازی کد بسیار شفاف صورت می گیرد، اشیاء دیگر به طور خودکار از این مزیت سود می جویند، زبان طبیعی برنامه (کد) باید به نحوی باشد که دیگر برنامه نویسان به راحتی آن را بفهمند.

۵- قابلیت توسعه (Extendibility): در برنامه نویسی شیء گرا می توانید بعضی از امکانات که خود زبان از آنها به طور مستقیم حمایت نمی کند را پیاده سازی کرد. به عنوان مثال می توان از اعداد مختلط به عنوان یک مثال نام برد. می توان با استفاده از روش های برنامه نویسی شیء گرا انواع عملیات را پیاده سازی کرد. همچنان که از کد های خود نگهداری و پشتیبانی می کنید، کاربران برای اضافه کردن قابلیت های جدید به سیستم با شما تماس می گیرند. همچنان که به ساخت مجموعه های اشیاء و کتابخانه های مرتبط با آن می پردازید، باید قابلیت ها و کارکرد های خود اشیاء را نیز توسعه دهید. OOP در واقع به این درخواست جامعه عمل می پوشاند. نرم افزار چیزی ایستا نیست نرم افزار باید رشد کرده و در طول زمان تغییر یابد تا همچنان قابل استفاده باقی بماند. OOP خواصی را به برنامه نویس ارائه می دهد که از طریق آن می توان کد ها را توسعه داد. این قابلیت ها شامل وراثت، چند شکلی بودن، سربار گذاری، جایگزینی و تعدادی دیگر از الگو های طراحی می باشند.

۶- به موقع بودن (Timely): چرخه زندگی پروژه های نرم افزاری مدرن عموماً براساس هفته ها سنجیده می شوند. OOP هدفش کوتاه کردن این چرخه توسعه است. در واقع OOP با استفاده از فراهم کردن توانایی هایی نظیر پایداری، قابلیت استفاده مجدد و قابلیت توسعه چرخه زمانی، توسعه نرم افزار را کوتاه می کند.

نرم افزار های طبیعی (ساده) طراحی سیستم های پیچیده را ساده می کنند. تا زمانی که از طراحی دقیق چشم پوشی نکنید، نرم افزار های طبیعی (ساده) می توانند چرخه طراحی را کوتاه نمایند، چرا که با OOP تنها بر روی مسئله ای که برای حل آن اقدام کرده اید متمرکز شده اید. زمانی که برنامه ای را به تعدادی از اشیاء تقسیم می کنید، توسعه هر یک از آن قطعات می تواند به صورت موازی پیش رود. چند برنامه نویس می توانند بر روی کلاس های

مختلف به صورت مستقل کار کنند. توسعه نرم افزار ها به صورت موازی، زمان کمتری برای اتمام نرم افزار می طلبد.

۷- محلی سازی تغییرات : در برنامه نویسی شیء گرا تغییراتی که در کد برنامه اعمال می شوند عموماً در داخل یک کلاس اتفاق می افتند و به طور عمومی در کل برنامه اعمال نمی شوند. این امر موجب می شود تا کنترل تاثیرات وارد بر نرم افزار راحت تر و مطمئن تر باشد. این ویژگی را می توان متأثر از قابلیت پنهان سازی داده ها دانست.

۸- طراحی آسان تر : سیستم ارث بری برنامه نویسی شیء گرا به برنامه نویس این امکان را می دهد تا بتواند راحت تر نرم افزار خود را طراحی کند و داشتن یک طرح کلی و اجمالی از نرم افزار به راحتی امکان پذیر باشد. لذا برنامه نویس حتی قبل از نوشتن کد برنامه می تواند طراحی کلی نرم افزار را انجام دهد.

۹- طراحی سریع تر : با استفاده از روش های برنامه نویسی شیء گرا می توان نرم افزار ها را سریع تر از سایر روش های برنامه نویسی طراحی کرد. این امر در حقیقت نتیجه استفاده مجدد از کد و طراحی سریع تر نرم افزار می باشد. بالطبع پروژ های نرم افزاری با حجم بالا، با این امر بسیار سریع تر و آسان تر انجام می شوند.

مفاهیم بنیادین شیء گرایی

در دنیای واقعی، یک شیء چیزی است که مشخصاتی دارد مثل اسم، رنگ، وزن، حجم و همچنین هر شیء رفتار های شناخته شده ای نیز دارد مثلاً در برابر نیروی جاذبه یا تابش نور یا وارد کردن فشار، واکنش نشان می دهد. اشیاء را می توان با توجه به مشخصات و رفتار های آنان دسته بندی کرد. برای نمونه، می توانیم همه اشیایی که دارای رفتار (تنفس) هستند را در دسته ای به نام (جانداران) قرار دهیم و همه اشیایی که چنین رفتاری را ندارند در دسته دیگری به نام (جامدات) بگذاریم. بدیهی است که اعضای هر دسته را می توانیم با توجه به جزئیات بیشتر و دقیق تر به زیر دسته هایی تقسیم کنیم. مثلاً دسته جانداران را می توانیم به زیر دسته های گیاهان، جانوران و انسان ها بخش بندی کنیم. البته هر عضو از این دسته ها، علاوه بر اینکه مشخصاتی مشابه سایر اعضا دارد، مشخصات منحصر به فردی نیز دارد که این تفاوت باعث می شود بتوانیم اشیای همگون را از یکدیگر تفکیک کنیم. مثلاً هر انسان دارای

نام، سن، وزن، رنگ مو، رنگ چشم و مشخصات فردی دیگر است که باعث می شود انسان ها را از یکدیگر تفکیک کنیم و هر فرد را بشناسیم.

در بحث شیء گرایی به دسته ها (کلاس) می گویند و به نمونه های هر کلاس (شیء) گفته می شود. مشخصات هر شیء را (صفت) می نامند و به رفتار های هر شیء (متد) می گویند. درخت سرو یک شیء از کلاس درختان است که برخی از صفات آن عبارتند از : نام، طول عمر، ارتفاع، قطر و ... و برخی از متد های آن نیز عبارتند از : غذا ساختن، سبز شدن، خشک شدن، رشد کردن و

شیء گرایی (OOP) در زبان C# بر چند پایه استوار است که سه مورد آن اصل است و به قرار زیر هستند :

Encapsulation
Inheritance
Polymorphism
Abstraction
Interface

حالا به بررسی و تعریف سه اصل اساسی در برنامه نویسی شیء گرا می پردازیم :

بسته بندی (Encapsulation): یعنی این که داده های مرتبط، با هم ترکیب شوند و جزئیات پیاده سازی مخفی شود. وقتی داده های مرتبط در کنار هم باشند، استقلال کد و پیمانه ای کردن برنامه راحت تر صورت می گیرد و تغییر در یک بخش از برنامه، سایر بخش ها را دچار اختلال نمی کند. مخفی کردن جزئیات پیاده سازی که به آن تجرید نیز می گویند سبب می شود که امنیت کد، حفظ شود و بخش های بی اهمیت یک فرایند از دید استفاده کننده آن مخفی باشد. به بیان ساده تر، هر بخش از برنامه تنها می تواند اطلاعات مورد نیاز کاربر را ببیند و نمی تواند به اطلاعات نامربوط دسترسی داشته باشد و آنها را دستکاری کند. در برخی از منابع از واژه کپسوله کردن یا محصور سازی به جای بسته بندی استفاده شده است.

وراثت (Inheritance): در دنیای واقعی، وراثت به این معناست که یک شیء وقتی متولد می شود، خصوصیات و ویژگی هایی را از والد خود به همراه دارد هر چند این شیء جدید با والدش در برخی از جزئیات تفاوت دارد. در برنامه نویسی نیز وراثت به همین معنا به کار می رود. یعنی از روی یک شیء موجود، شیء جدیدی ساخته می شود که صفات و متد های

شیء والدش را دارا بوده و البته صفات و متد های خاص خود را نیز داشته باشد. امتیاز وراثت در این است که از کد های مشترک استفاده می شود و علاوه بر این که می توان از کد های قبلی استفاده مجدد کرد، در زمان نیز صرفه جویی شده و استحکام منطقی برنامه هم افزایش می یابد.

چند ریختی (Polymorphism) : که به آن چند شکلی هم می گویند به معنای یک چیز بودن و چند شکل داشتن است. چند ریختی بیشتر در وراثت معنا پیدا می کند. برای مثال اگر چه هر فرزندی مثل والدش اثر انگشت دارد، ولی اثر انگشت هر شخص با والدش یا هر شخص دیگری متفاوت است. پس اثر انگشت در انسان ها چند شکلی است.

در ادامه به شکل عملی خواهید دید که چگونه می توان مفاهیم فوق را در قالب برنامه های کامپیوتری پیاده سازی کرد.

کلاس و شیء (Class & Object)

زبان های شیء گرا با الهام و تبعیت از دنیای واقعی بوجود آمده اند بدین گونه که در دنیای واقعی به هر جا که بنگریم، موارد زیادی از نمونه های متمایز و متفاوت و در عین حال با ساختار و قالب یکسان قابل مشاهده است. به عنوان مثال، تمامی انسان ها با تفاوت های بسیاری آفریده شده اند، به گونه ای که هیچ دو انسانی ویژگی های کاملاً یکسانی ندارد. با این حال همگی دارای ساختار و قالب یکسانی هستند. به این ساختار و قالب مشترک، کلاس (Class) گفته می شود و نمونه های موجود در یک ساختار را شیء (Object) آن کلاس می نامیم.

برای مثال کلاس انسان را می توان ترکیبی از خاصیت ها یا فیلد های رنگ چشم، رنگ پوست، جنسیت، سن، قد، وزن و ... و همچنین ترکیبی از متد های راه رفتن، حرف زدن، دویدن، ... و در عین حال ترکیبی از رویداد های عصبانی شدن، شاد شدن و ... دانست.

پس از طراحی کلاس، با ایجاد نمونه هایی از ساختار انتخابی و پر کردن قالب آن با گونه های متنوع، می توانیم اشیای کاملاً متمایزی در اختیار داشته باشیم. همانطور که در دنیای ما، انسان های متمایز با ساختار یکسان ولی با رنگ چشم و رنگ پوست های متفاوت قابل مشاهده اند.

اشیاء در برنامه نویسی نیز به همین صورت هستند وقتی که یک شیء در اختیار داشته باشید می توانید به سادگی از آن استفاده کنید و بخواهید که وظایف لازم را انجام دهد. برای این موارد لازم نیست بدانید که شیء به صورت درونی چگونه کار می کند و یا لازم نیست حتی کوچکترین اطلاعاتی در مورد نحوه عملکرد آن داشته باشید.

اشیای نرم افزار عموماً دارای مشخصات زیر هستند :

. هویت : یک شیء همواره نوع خود را می داند.

. حالت : هر شیء در هر زمانی وضعیت و حالت خود را می داند.

. رفتار : به عکس العمل یک شیء در مقابل درخواست های کاربر، رفتار آن شیء می گویند.

برای مثال یک کنترل Button یا یک Form در محیط برنامه نویسی یک شیء محسوب می شود زیرا دارای هویت، حالت و رفتار می باشد.

نحوه تعریف کلاس در زبان C#

ساختار کلی تعریف کلاس در زبان C# به صورت زیر است :

```
نام کلاس  class  سطح دسترسی  
{  
    تعریف فیلدها  
    تعریف خاصیت ها  
    تعریف متد ها  
    تعریف رویداد ها  
}
```

سطح دسترسی : به طور کلی در زبان های برنامه نویسی OOP و به طور خاص در .NET، ممکن است در تعریف و اعلان متغیر ها، کلاس، متد ها، ساختار ها و هر موجودیت دیگر، در ابتدا یکی از چند کلید واژه (کلمه کلیدی) زیر استفاده شود. به این کلید واژه ها که دامنه و اعتبارسنجی به آن موجودیت را مشخص می سازند اصطلاحاً Access Modifier می گویند، که به ۵ دسته کلی زیر تقسیم می شود :

۱- public (عمومی): وقتی سطح دستیابی public در نظر گرفته شود به مفهوم عدم وجود محدودیت در دستیابی است. وقتی سطح دستیابی public تعیین شود به معنای این است که این کلاس خارج از فضای نامی که در آن تعریف می شود قابل استفاده است.

۲- private (خصوصی): سطح دسترسی private به این معناست که داده ها و توابع تعریفی در این سطح، فقط در این کلاس شناخته می شوند و در خارج از کلاس هیچگونه دسترسی به آنها وجود ندارد و تمامی عملیات روی آنها فقط باید در همان کلاس تعریف شوند که همان مفهوم Encapsulation یا محرمانگی و بسته بندی را پیاده سازی می کنند.

۳- protected (محافظت شده): سطح دسترسی protected به این معناست که موجودیت های داخل این کلاس، تنها داخل خود کلاس و داخل کلاس فرزندان خود قابل دسترس است.

۴- internal (درونی): سطح دسترسی internal به این معناست که موجودیت های داخل این کلاس، از هر قسمت پروژه (Assembly) قابل دسترس می باشد. در واقع این نوع سطح دستیابی مشخص می کند که کلاس فقط در همان فضای نامی که تعریف می شود قابل استفاده است.

۵- protected internal: اجتماع دو حالت protected و internal است. یعنی هم در داخل کلاس خود و کلاس های فرزندان (کلاس های ارث برده) قابل دسترس است و هم در هر جایی از داخل پروژه.

class: واژه ای کلیدی است که به همین صورت برای تعریف کلاس به کار می رود. نام کلاس: نامی است که توسط برنامه نویس برای کلاس انتخاب می گردد. نامگذاری برای کلاس ها، از نام گذاری برای متغیر ها پیروی می کند.

اعضای کلاس در هنگام تعریف کلاس در داخل {} قرار می گیرند. که طبق تعریف صفحه قبل عبارتند از:

فیلد ها (Fields): فیلد ها عناصر نگهدارنده اطلاعات کلاس هستند که معمولاً فقط و فقط یک اطلاعات خاص را نگه می دارند. مثل سن در مثال انسان و tag در کلاس کنترل button.

خاصیت ها (Properties): خاصیت ها، عناصری هستند که علاوه بر نگهداری اطلاعات، تغییر آنها، تغییراتی را در شیء سبب می شود. به عنوان مثال خاصیت رنگ چشم در مثال انسان که تغییر آن، نمایی متفاوت به بیننده شیء خواهد داد یا خاصیت Color در کلاس button که تغییر آن، تغییر ظاهری طول دکمه را سبب می شود. فیلد ها و خاصیت ها علاوه بر این که در زمان ایجاد شدن قابل مقدار دهی اولیه هستند، می توانند بعدا توسط خود شیء نیز مقادیر جدید بپذیرند. گاهی به منظور در نظر گرفتن تغییرات احتمالی آینده، خیلی از فیلد ها را به صورت خاصیت، پیاده سازی می کنند.

متد ها (Methods): متد ها، توابعی هستند که توسط طراح کلاس (در داخل کلاس)، پیاده سازی شده اند (در زبان C# تمامی متغیر ها و توابع تنها در داخل کلاس ها قابل تعریف هستند که به این ویژگی، شیء گرایی قوی می گویند). فراخوانی متد ها موجب انجام عملیاتی برای شیء مورد نظر می گردد. در مثال انسان، فراخوانی متد دویدن باعث حرکت انسان می شود یا فراخوانی متد Focus باعث حرکت فوکوس روی دکمه (شیء) مورد نظر می گردد. رویداد ها (Events): رویداد ها، اتفاقاتی هستند که زمان رخ دادن آنها توسط طراح کلاس مشخص می شود و اشیاء بعد از ایجاد شدن، می توانند پاسخ به رویداد ها را مشخص نمایند. در این صورت بعد از انجام این کار هر بار که اتفاق مورد نظر به وقوع بپیوندد پاسخ مشخص شده، اجرا می شود.

به عنوان مثال می توان شکستن ظروف در هنگام عصبانیت در کلاس انسان یا چاپ یک پیغام در زمان کلیک یک دکمه در کلاس button را نام برد.

برای تفهیم بیشتر مطالب، به بررسی یک سری رفتار های اشیاء در کلاس انسان (با شبیه سازی کلاس انسان به روشی که گفته شد) و کلاس button (دکمه) و توجیه آنها با توجه به موضوعات بالا می پردازیم:

* همانگونه که گفته شد انسانها با این که کاملا از همدیگر متمایز هستند، همگی قالب و ساختار یکسانی دارند. برای مثال همگی فیلد هایی مثل سن، خاصیت هایی مثل رنگ چشم، متد هایی مثل دویدن و رویداد هایی مثل عصبانی شدن دارند که این بدین دلیل است که همگی از کلاس انسان به ارث برده اند. به همین صورت تمامی دکمه هایی که روی Form ها

مشاهده می کنید علیرغم تفاوت هایی که دارند دارای ساختار یکسانی هستند. برای مثال همگی فیلد tag، خاصیت width، و رویداد click دارند که این ساختار یکسان از آنجا نشأت می گیرد که همگی از کلاس button ایجاد شده اند.

* موجودیت انسان ها با یکدیگر متفاوت است. برای مثال انسان ها در میزان سن، نوع رنگ چشم (سیاه، آبی و...)، و واکنش هایی که در هنگام عصبانی شدن بروز می دهند (کنترل خشم، زد و خورد و ...) متفاوتند. این امر بدین دلیل است که اشیاء می توانند قالب خود را به طور متفاوت پر نموده و مقدار فیلد ها، خاصیت ها و واکنش هایی را که به رویداد ها می دهند مخصوص خود تنظیم کنند. همین طور دکمه هایی که روی Form ها قابل مشاهده اند می توانند در مقدار فیلد tag، مقدار خاصیت width، اتفاقی که با کلیک روی آنها اتفاق می افتد متفاوت باشند. همه این ها بیانگر این است که با پر کردن و مقدار دهی متفاوت به قالبی که کلاس مشخص کرده است اشیایی متفاوت و متمایز در اختیار خواهیم داشت.

بنابراین نمونه ای از تعریف کلاس sum به صورت زیر است :

```
public class sum
{
classMember
}
```

در این تعریف، کلاسی به نام sum با سطح دستیابی public مشخص شده ولی اعضای آن هنوز مشخص نیستند.

نکته!

اگر در هنگام تعریف کلاس، سطح دستیابی را مشخص نکنید، پیش فرض آن private است.

نحوه ساخت یک شیء

فرآیند ساخت یک شیء را نمونه سازی می گویند، چرا که شیء نمونه ای از کلاس است. همانطور که قبلاً گفتیم کلاس ها، تنها ساختار و قالبی هستند که تعریف شده اند و با ایجاد شیء ای از کلاس، موجودیتی با همان ساختار ایجاد می شود و ما می توانیم از متد، فیلد ها، خاصیت ها، و ... تعریف شده در کلاس مربوطه استفاده نماییم. به این صورت ما یک شیء در

اختیار داریم که علاوه بر این که ساختار و قالب کلی کلاس مربوطه را دارد، می تواند داده های متمایز را از طریق فیلد های خود نگهداری کند و عملیات مربوطه روی شیء نیز می توانند با استفاده از متد ها انجام شوند. همیشه برای استفاده از متد ها و فیلد های کلاس می بایست شیء ای از آن کلاس در اختیار داشته باشیم البته باید توجه داشت که فیلد ها و متد های static کلاس از این قاعده مستثنی هستند که در بخش های بعدی بررسی خواهیم کرد. برای ایجاد شیء ای از یک کلاس به روش زیر عمل می شود :

ابتدا متغیر یا فیلدی از کلاس مربوطه تعریف شده، سپس شیء ای از کلاس مربوطه ایجاد شده و به آن متغیر یا فیلد تخصیص داده می شود. آنگاه متغیر یا فیلد، حاوی شیء ای از کلاس مربوطه می باشد. البته این امکان وجود دارد که این دو مرحله هم زمان هم انجام شود.

```
اسم متغیر نام کلاس  
(); اسم کلاس new = اسم متغیر
```

در صورتی که بخواهیم نحوه تعریف شیء را در یک مرحله انجام دهیم به صورت زیر خواهد بود :

```
(); اسم کلاس new = اسم متغیر نام کلاس
```

در زبان C# اشیاء باید توسط کلمه کلیدی new از حافظه فضا دریافت کنند. اگر شما از کلمه کلیدی new برای این کار استفاده نکنید و بخواهید از آن شیء استفاده کنید، یک خطای کامپایلری دریافت خواهید کرد. پس برای اینکه شیء ای را ایجاد کنید باید با استفاده از عملگر new فضایی را از حافظه به وی اختصاص دهید.

برای مثال، برای ایجاد نمونه ای از کلاس sum که ساخته شد، به صورت زیر عمل کنید :

```
sum objSum = new sum();
```

با دستور بالا، شیء ای به نام objSum از کلاس sum ایجاد می شود.

دسترسی به اعضای شیء

پس از این که نمونه ای از کلاس یا شیء ای از کلاس را ایجاد کردید، باید بتوانید به اعضای شیء دسترسی داشته باشید. دقت کنید که در خارج از کد شیء، فقط می توانید به اعضای با سطح دسترسی public دسترسی داشته باشید. به عنوان مثال، برای دسترسی به عضو

classMember از شیء objSum که دارای سطح دستیابی public است به صورت زیر عمل می شود :

```
objSum.classMember;
```

اعضای کلاس

تعریف کلاس بدون تعریف اعضای آن کامل نمی شود. اعضای کلاس در زبان C# می تواند موارد زیر باشد :

ثوابت (Constants) - فیلدها (Fields) - متدها (Methods) - خواص (Properties) - شاخص بندها (Indexers) - رویدادها (Events) - عملگرها (Operators) - سازنده ها یا مولد ها (Constructors) - مخرب ها (Destructors).

بعضی از اعضای کلاس را در این فصل و بعضی دیگر را در فصل های دیگر بررسی می کنیم.

ثوابت

با نحوه تعریف ثابت ها در فصول گذشته آشنا شدید. حالا می خواهیم در داخل کلاس sum چند ثابت را تعریف کنیم :

```
public class sum
{
    private const int x = 5;
    const int a = 5 , b = 6;
}
```

دستور اول ثابتی به نام x را با مقدار 5 و با سطح دستیابی private تعریف می کند. دستور دوم ثابت a را با مقدار 5، و ثابت b را با مقدار 6 تعریف می کند. چون سطح دستیابی برای این دو ثابت مشخص نشده به طور خودکار، private منظور خواهد شد.

نحوه تعریف فیلدها

فیلدها معمولاً بعضی از مقادیر اطلاعاتی را برای شیء نگهداری می کنند. فیلدها می توانند شامل انواع داده ها، Object ای از کلاس های دیگر، شمارش ها، ساختار ها باشند. برای تعریف فیلدها به شکل زیر عمل می کنیم :

نام اختیاری فیلد نوع داده سطح دسترسی

در مورد سطوح دسترسی پیش از این صحبت کردیم. نوع داده یکی از انواع متداول در زبان C# است، مثل float, int. نام اختیاری فیلد که از قوانین نامگذاری برای متغیرها پیروی می کند. شما می توانید همزمان با تعریف یک فیلد در کلاس، به آن مقدار اولیه نیز اختصاص دهید. برای مثال دستور زیر فیلد x را از نوع int، با مقدار اولیه ۹ و سطح دستیابی public و دستور دوم فیلد y را از نوع float و با سطح دستیابی private معرفی می کند.

```
public class sum
{
    public int x = 9;
    private float y;
}
```

نکته!

هر یک از انواع داده ها برای خود دارای مقدار پیش فرضی هستند. در صورتی که هنگام تعریف فیلدهای یک کلاس به آنها مقداری را تخصیص ندهید، کامپایلر به صورت خودکار مقدار پیش فرض آن نوع را طبق جدول ۱ - ۵ به آن اختصاص می دهد.

انواع داده	مقدار پیش فرض
int	0
long	0
float	0
double	0
bool	FALSE
char	'\0' (null character)
string	"" (empty string)
objects	Null

جدول ۱ - ۵

خاصیت ها

خاصیت ها مشابه فیلدهای کلاس هستند با این تفاوت که معمولاً علاوه بر نگهداری یک مقدار، کارهای بیشتری انجام می دهند. در واقع خاصیت ها اعضای از یک شیء هستند که ویژگی های آن را شرح می دهد. خاصیت ها در کاربرد متداول خود معمولاً با فیلدهایی کار می کنند که دارای سطح دسترسی خصوصی (private) بوده و از دسترسی استفاده کننده

کلاس مخفی هستند. از دید استفاده کننده کلاس در کار با خاصیت ها دو کار مهم را انجام می دهد :

۱- داده های خود را مقدار می دهند.

۲- مقدار داده ها را بازیابی می کند.

حال آنکه معمولاً عملیاتی بیش از این، توسط خاصیت های مربوطه انجام می شود و حتی ممکن است مقداری که در فیلد مربوطه ذخیره می شود یا مقداری که استفاده کننده کلاس داده است متفاوت باشد.

با این توضیحات در تعریف خاصیت باید این امکان باشد که عملیات مربوطه در هنگام تخصیص مقدار به خاصیت یا خواندن مقدار از خاصیت مشخص شود. این دو ترتیب با قسمت های `set` و `get` در خاصیت مشخص می شود.

برای مقدار دادن به خاصیت از کلمه `set` استفاده می شود. قسمت `set` مانند متدی است که پارامتری (ضمنی) از نوع داده خاصیت با نام `value` گرفته است و خروجی ندارد. برای بازیابی خاصیت از واژه `get` استفاده می شود. قسمت `get` مانند متدی است که پارامتری ندارد و خروجی از نوع داده خاصیت بر می گرداند.

برای اینکه خاصیت بتواند مقداری را به کد خارج از کلاس برگرداند، از دستور `return` استفاده می کند. در واقع برای برگرداندن خروجی از دستور `return` استفاده می شود. دستور `return` به صورت زیر استفاده می شود :

`return` مقدار خروجی;

دستور `return`، خروجی مشخص شده در جلوی این دستور را بر می گرداند و از متد خارج می شود. مقدار خروجی مقداری است که باید به کد خارج از کلاس برگردانده شود. در صورتی که خروجی از نوع `void` باشد الزامی به استفاده از مقدار در جلوی دستور `return` نیست و می تواند تنها باعث خروج از متد شود. یعنی به صورت مقابل `return;`.

اگر دستوری بعد از دستور `return` در متد قرار بگیرد اجرا نمی شود چرا که همان طور که گفتیم اجرای این دستور باعث خروج از متد می شود، مگر این که دستور `return` شرطی باشد به این مفهوم که تحت شرایطی اجرا نشود و در آن شرایط دستور بعدی اجرا می شود.

برای مثال در کد زیر در صورتی که باقیمانده متغیر i تقسیم بر ۲ برابر ۰ باشد، اجرای برنامه از متد خارج شده و دستورات بعدی در متد اجرا نمی شود، در غیر اینصورت دستور بعدی یعنی افزایش مقدار متغیر i اجرا می گردد.

```
int i = 0;
.
.
if (i % 2 == 0)
{
return;
}
i = i + 8;
```

فرمت کلی تعریف خاصیت به صورت زیر است :

نام خاصیت نوع داده خاصیت سطح دسترسی

```
{
    get
    {
        ...
        ...
        عملیات مربوط به خواندن مقدار از خاصیت
        Return    مقدار خروجی از نوع داده خاصیت
    }
    set
    {
        ...
        ...
        عملیات مربوط به تخصیص مقدار به خاصیت
    }
}
```

در صورت ذکر نکردن `set` یا `get`، خاصیت به ترتیب فقط خواندنی و فقط نوشتنی می شود. موقعی که نام خاصیت در سمت راست عبارتی قرار گیرد، در واقع شما مقدار خاصیت را می خوانید و دستورات قسمت `get` اجرا شده و مقدار حاصله برگردانده می شود. و موقعی که خاصیت در سمت چپ عبارت تخصیص به کار می رود، دستورات قسمت `set` اجرا می شوند در این حالت مقدار عبارت سمت راست به صورت تخصیص پارامتر ضمنی به نام `value` به دستورات قسمت `set` ارسال شده و در دستورات `set` شما از `value` با همین معنی و مقدار استفاده خواهد کرد.

در واقع وقتی می خواهید از خارج کلاس به یک خاصیت دستیابی داشته باشید بخش `get` خاصیت عمل می کند و با دستور `return` مقدار خاصیت به شما برگردانده می شود. وقتی می خواهید مقدار خاصیتی را تغییر دهید، بخش `set` خاصیت عمل می کند. بنابراین، اگر خاصیتی فاقد بخش `set` باشد فقط خواندنی، و اگر فقط شامل بخش `get` باشد فقط نوشتنی خواهد بود. برای مثال فرض کنید در کلاس `sum` خاصیتی به نام `CalcField` تعریف می کنیم که این خاصیت، مقداری را که کاربر به آن تخصیص داده است را دو برابر کرده و در فیلدی با سطح دسترسی خصوصی (`private`) و با نام `PCalcField` قرار داده است. و در زمان خواندن مقدار فیلد مربوطه را تقسیم بر ۲ کرده و حاصل را بر می گرداند. در این عملیات، استفاده کننده از عملیات اضافی اطلاعی ندارد و از دید او خاصیت مانند یک فیلد معمولی عمل می کند.

```
public class sum
{
    private int PCalcField;
    public int CalcField
    {
        get
        {
            return PCalcField / 2;
        }
        set
        {
            PCalcField = value * 2;
        }
    }
}
```

مثال دیگر از کاربرد خاصیت ها زمانی است که مدیران یک پروژه نرم افزاری، کار طراحی و پیاده سازی کلاس ها را انجام می دهند و `dll` آن را در اختیار برنامه نویسان دیگر قرار می دهند تا آنها از کلاس های آماده شده، استفاده نمایند. کاربران به خاصیت ها، همانند فیلد های دیگر، دسترسی دارند (در مقدار دهی و خواندن مقدار) ولی از نحوه انجام عملیات و کارکرد آنها اطلاعی ندارند و از دید آنها مخفی است.

نحوه تعریف متد ها

متد، مجموعه ای از دستورالعمل ها است که عملیاتی را بر روی داده های کلاس انجام می دهد. متد ها، روش هایی هستند که به وسیله آن می توان به یک شیء گفت که چگونه وظیفه

خاصی را انجام دهد. در زبان های قبل از NET امکان تعریف عملیات در توابع و روال ها وجود داشت و زمانی که توابع عضو کلاس بودند به آنها متد گفته می شد. در NET تنها متد داریم چون امکان تعریف تابع در بیرون از کلاس وجود ندارد که نشان دهنده قوی بودن شیء گرای در زبان C# می باشد.

شکل کلی تعریف یک متد به صورت زیر است :

[[لیست پارامتر]] نام متد نوع داده خروجی سطح دسترسی

{

دستورات بدنه متد

;خروجی return (در حالت خروجی void اختیاری است)

}

سطوح دسترسی یکی از سطوح دستیابی است که در مورد آن قبلا صحبت کردیم و معمولا متد ها دارای سطح دستیابی public هستند. نوع داده خروجی مشخص می کند که متد از چه نوعی است، به عبارت دیگر مشخص می کند متد چه نوع داده ای را باید برگرداند. نام متد هم از قوانین نامگذاری متغیر ها پیروی می کند. پارامتر ها برای تعریف عملیات بدنه متد به صورت پارامتریک و سراساس پارامتر های ورودی به کار می روند و در حالت هایی برای برگرداندن خروجی نیز مفید می باشند. اگر متد دارای چند پارامتر باشد، پارامتر ها با کاما از هم جدا می شوند. متد می تواند فاقد پارامتر هم باشد. لیست پارامتر ها به صورت زیر نوشته می شوند :

[.. و اسم پارامتر۲ نوع داده پارامتر۲ و اسم پارامتر۱ نوع داده پارامتر۱]

نوع داده یکی از انواع داده معتبر می باشد، انواع داده هایی که شما تعریف می کنید مانند کلاس ها، آرایه ها و ... نیز می توانید مورد استفاده قرار گیرند. برای مثال لیست پارامتری (int p1 , string s) دو پارامتر به نام های p1 و s و از نوع داده صحیح و رشته ای تعریف می کند. یک متد می تواند پارامتری هم نداشته باشد. در صورتی که متد شما خروجی نداشته باشد، نوع خروجی آن void تعریف می شود. دستورات بدنه متد مشخص کننده عملکرد متد می باشد و حاوی مجموعه دستوراتی است که متد جهت رسیدن

به هدف مورد نیاز خویش از آنها بهره می برد. برای برگرداندن خروجی از دستور return استفاده می شود. به عنوان مثال به دستورات زیر توجه کنید :

```
public class sum
{

public int Power2 (int m)
{
m = m * m;
return m;
}

public int Power (int m , int n)
{
int Result = 1;
for (int i = 0 ; i < n ; i++)
{
Result = Result * m;
}
return Result;
}

}
```

در دستورات بالا دو متد Power2 و Power در کلاس sum تعریف شده اند. متد Power2 یک پارامتر از نوع صحیح می گیرد و مجذور آن را به عنوان خروجی صحیح بر می گرداند. متد Power دو پارامتر صحیح می گیرد و اولی را به توان دومی می رساند و حاصل را به عنوان خروجی صحیح بر می گرداند.

نکته!

در زبان C# بایستی متغیر هایی را که در متد ها از آنها استفاده می کنید قبل از استفاده مقدار دهی نمایید و بهترین مکان برای این کار در زمان تعریف متغیر ها و متناسب با کاربرد مورد نظر تان است. در متد دوم، با توجه به همین اصل در هنگام تعریف متغیر Result، مقدار دهی اولیه صورت گرفته است.

فراخوانی متد

برای استفاده از متد، تنها تعریف آن کافی نیست، بلکه باید اجرا شود اجرای متد را فراخوانی متد می گویند. متد باید توسط متد دیگری فراخوانی شود تا اجرا گردد. متدی که عمل

فراخوانی را انجام می دهد به نام متد فراخوان و متدی که فراخوانی می شود را متد فراخوانی شونده می گویند.

اگر در هنگام تعریف متد ها، پارامتر هایی را برای آنها تعریف کنیم، در هنگام فراخوانی آن متد نیز باید پارامتر هایی را به آن ارسال کنیم. پارامتر هایی که در هنگام تعریف متد مشخص می شوند پارامتر های مجازی و پارامتر هایی که در هنگام فراخوانی به متد ارسال می شوند را پارامتر های واقعی می گویند.

برای فراخوانی متد، نام متد آورده می شود و در پرانتز های جلوی نام متد، لیست مقادیر پارامتر ها متناسب با ترتیب و نوع پارامتر های تعریف شده آورده می شوند (نوشتن پرانتز ها حتی زمانی که پارامتری ندارید نیز اجباری است). مقادیر می توانند شامل متغیر ها و مقادیر ثابت باشند. در صورتی که متد دارای خروجی باشد و تمایل داشته باشید تا مقدار خروجی را در متغیری ذخیره کنید، نام متغیر بایستی با علامت = قبل از فراخوانی آورده شود.

برای مثال فرض کنید قصد داریم متدی با نام Power3 در کلاس sum بنویسیم که مکعب یک عدد صحیح را (توان سوم عدد) را به عنوان خروجی صحیح برگرداند و در نوشتن این تابع می خواهیم از فراخوانی متد Power2 که قبلا در این کلاس نوشته ایم استفاده کنیم :

```
public int Power3 (int m)
{
    int Result = 0;
    Result = Power2 (m);
    Result = Result * m;
    return Result;
}
```

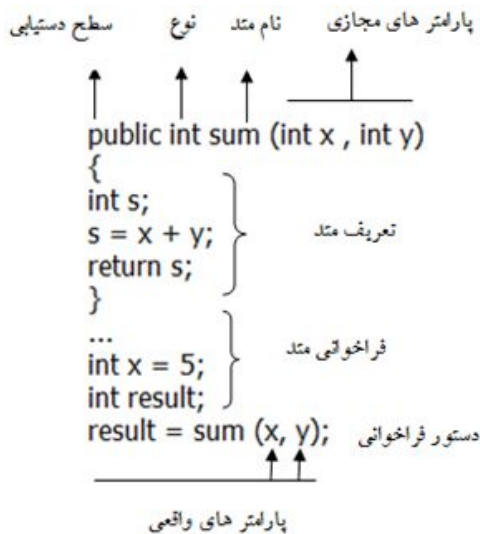
همانطور که در دستورات بالا مشاهده می کنید در بدنه این متد، متد Power2 با مقدار ارسالی پارامتر متد Power3 یعنی m فراخوانی شده است و خروجی آن یعنی m به توان 2 به متغیر Result تخصیص داده شده است و از حاصل ضرب $Result * m$ مقدار خروجی نهایی به دست آمده است.

وقتی متدی اجرا شد می تواند مقادیری را به متد فراخوان برگرداند. این مقادیر می توانند به دو طریق به متد فراخوان برگردند :

۱- از طریق نام متد

۲- از طریق پارامترها

اگر قرار باشد متد مقداری را در نامش برگرداند، باید نوع متد به نحوی انتخاب شود که با نوع برگشتی توسط متد یکسان باشد. به عنوان مثال اگر متدی یک مقدار صحیح را در نامش برمی گرداند، نوع این متد در هنگام تعریف باید `int` تعیین شود اگر متد مقداری را در نامش برنگرداند، از نوع `void` خواهد بود. در دستورات زیر نحوه فراخوانی متد `sum` را به همراه اجزای آن می توانید مشاهده کنید :



نحوه ارسال پارامترها به متد

برای ارسال پارامترها به متد، دو روش تعریف وجود دارد.

۱- ارسال از طریق مقدار (by Value) : در این روش تمام تغییراتی که در بدنه متد روی مقدار پارامتر ارسال شده اعمال می شود، بعد از فراخوانی متد، از بین می رود یا به عبارت دیگر پارامتر ارسال شده بعد از فراخوانی، مقدار قبل از فراخوانی را خواهد داشت و تنها مقادیر حاصل از تغییرات در بدنه متد معتبر هستند. این روش، روشی است که به طور پیش فرض در نظر گرفته می شود. در زمان فراخوانی، یک مقدار ثابت یا یک متغیر می تواند به عنوان مقدار پارامتر ارسال شود.

۲- ارسال از طریق ارجاع (by Reference) : تمام تغییرات روی پارامتر ارسال شده بعد از اجرای فراخوانی، موجود خواهد بود. یعنی پارامتر ارسال شده پس از فراخوانی، آخرین

مقداری را که در بدنه متد به دست آورده است خواهد داشت. دو روش ارسال پارامتر به صورت ارجاع وجود دارد که عبارتند از :

۱- استفاده از کلمه کلیدی ref : برای تعریف یک پارامتر به عنوان ارسال از طریق ارجاع می بایست کلمه را ref را قبل از تعریف پارامتر به کار برد و در زمان فراخوانی نیز به کار بردن ref قبل از متغیری که به عنوان مقدار پارامتر ارسال می شود، الزامی است. همچنین نمی توان از مقادیر ثابت به عنوان مقادیر پارامتر های ارسال با ارجاع استفاده کرد. برای مثال اگر متدی پارامتری از نوع صحیح و ارجاع با آدرس تعریف کرده است، در زمان فراخوانی این متد نمی توانید از عدد ثابت استفاده کنید ولی می توانید از ref i که در آن، i متغیری صحیح است استفاده نمایید.

نکته!

در حالت ref آرگومان های ارسالی به متد قبل از ارسال حتما باید دارای مقدار اولیه باشند. در غیر این صورت با خطای زمان کامپایل مواجه می شویم.

۲- استفاده از کلمه کلیدی out : اگر متغیری (نوع مقداری) را در متدی تعریف کردید ولی فاقد مقدار است و علاقه مند هستید این متغیر را به روش ارجاع به متدی بفرستید تا متد فراخوانی شده آن را مقدار دهد و شما از آن مقدار در متد فراخوان استفاده کنید برای این کار پارامتر متناظر آن را باید با واژه out مشخص کنید.

نکته!

در حالت out نیازی به مقدار دهی آرگومان ها قبل از ارسال به متد نمی باشد. بلکه باید پارامتر ها را در داخل بدنه متد مقدار دهی نمائیم در غیر اینصورت چنانچه پارامتر های نوع out را در درون متد مقدار دهی اولیه کنیم با خطای زمان کامپایل مواجه می شویم.

با توضیحات داده شده می توان این طور استنباط کرد که یکی از کاربردهای پارامتر های ارسال از طریق ارجاع، زمانی است که قصد دارید پارامتری نقش خروجی را نیز برای شما داشته باشد. یکی از موارد متداول، زمانی است که شما قصد دارید بیش از یک خروجی از متد خود دریافت کنید.

برای مثال فرض کنید می خواهیم متدی با نام `Power23` بنویسیم که مجذور و مکعب یک عدد را در اختیار ما بگذارد. برای این کار متد ما یک خروجی مجذور عدد خواهد داشت و خروجی دیگر یعنی مکعب عدد را از طریق یک پارامتر ارسال با ارجاع در اختیار ما خواهد گذاشت.

```
public int Power23 (int m, ref int m3)
{
    int Result = 0;
    Result = Power2 (m);
    m3 = Result * m;
    return Result;
}
```

پس از فراخوانی متد بالا، شما مقادیر مورد نظر را از خروجی متد و پارامتر دوم در اختیار خواهید داشت. در مواردی که پارامتری فقط نقش خروجی داشته و نیازی به ارسال مقدار از طریق آن متد وجود نداشته باشد مانند `m3` در مثال بالا بهتر است که پارامتر را از نوع پارامتر خروجی `out` تعریف کنید. پارامتر خروجی مشابه پارامتر ارسال از طریق ارجاع است با تفاوت های زیر :

- الف : در تعریف و فراخوانی به جای کلمه `ref` از کلمه `out` استفاده می شود.
 - ب : چون نیاز به ارسال مقدار از طریق آن وجود ندارد، اجباری به مقدار دهی اولیه متغیری که به عنوان پارامتر `out` ارسال می شود وجود نخواهد داشت.
 - ج : به خوانایی برنامه کمک می کند و برنامه نویس به راحتی در می یابد که پارامتر مربوطه تنها نقش خروجی را در متد ایفا می کند. کد زیر بازنویسی متد بالا را با استفاده از پارامتر خروجی نشان می دهد.
- پس از فراخوانی متد بالا، شما مقادیر مورد نظر را از خروجی متد و پارامتر دوم در اختیار خواهید داشت. در مواردی که پارامتری فقط نقش خروجی داشته و نیازی به ارسال مقدار از طریق آن متد وجود نداشته باشد مانند `m3` در مثال بالا بهتر است که پارامتر را از نوع پارامتر خروجی `out` تعریف کنید. پارامتر خروجی مشابه پارامتر ارسال از طریق ارجاع است با تفاوت های زیر :

- الف : در تعریف و فراخوانی به جای کلمه `ref` از کلمه `out` استفاده می شود.

ب : چون نیاز به ارسال مقدار از طریق آن وجود ندارد، اجباری به مقدار دهی اولیه متغیری که به عنوان پارامتر out ارسال می شود وجود نخواهد داشت.

ج : به خوانایی برنامه کمک می کند و برنامه نویس به راحتی در می یابد که پارامتر مربوطه تنها نقش خروجی را در متد ایفا می کند. کد زیر بازنویسی متد بالا را با استفاده از پارامتر خروجی نشان می دهد.

```
public int Power23 (int m, out int m3)
{
    int Result = 0;
    Result = Power2 (m);
    m3 = Result * m;
    return Result;
}
```

مثال زیر چگونگی استفاده از متدی که آرگومان ها به صورت ref به آن وارد شده اند را نشان می دهد و نیز چگونگی تغییر آرگومان های متد فراخواننده را در زمان تغییر پارامتر های متد فراخوانی شده را نشان می دهد.

```
using System;
class Program
{
    static void Main()
    {
        Console.Write("a=? ");
        int a = int.Parse(Console.ReadLine());
        Console.Write("b=? ");
        int b = int.Parse(Console.ReadLine());

        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Swap(ref a, ref b);
        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Console.Read();
    }

    static void Swap(ref int x, ref int y)
    {
        int tmp = x;
        x = y ;
        y = tmp ;
        Console.WriteLine("\n X={0}\tY={1}", x, y);
    }
}
```

همانطور که در دستورات بالا می بینید در ارسال آرگومان ها به صورت ref کلمه کلیدی ref را هم در زمان فراخوانی متد و هم در زمان تعریف متد بایستی ذکر کرد. همانطور که در

پنجره تصویر ۱ - ۵ در خروجی برنامه مشاهده می کنید اگر پارامتر ها از نوع ref معرفی شوند تغییر مقدار آنها بر روی آرگومان های نظیر تاثیر می گذارد.



تصویر ۱ - ۵

مثال زیر چگونگی استفاده از متدی را که آرگومان ها به صورت out به آن وارد شده اند را نشان می دهد و نیز چگونگی تغییر آرگومان های متد فرخواننده در هنگام تغییر پارامتر های متد فراخوانی شده را بررسی می کند.

```
using System;
class Program
{
    static void Main()
    {
        int a, b;

        Swap(out a, out b);
        Console.WriteLine("\n a={0}\tb={1}", a, b);
        Console.Read();
    }

    static void Swap(out int x, out int y)
    {
        x=10 ;;
        y=20 ;
        Console.WriteLine("\n X={0}\tY={1}", x, y);
    }
}
```

همانطور که در مثال بالا مشاهده می کنید پارامتر های نوع out درون متد مقدار دهی اولیه شده اند و چون پارامتر های از نوع out می باشند این مقادیر به آرگومان های نظیر یعنی a و b نیز اعمال می گردد. خروجی برنامه بالا را می توانید در پنجره تصویر ۲ - ۵ مشاهده کنید.



تصویر ۲ - ۵

فضای نام (Name Space)

یکی از قسمت های مهم چارچوب Net، کلکسیون عظیم کلاس های آن است. در چارچوب Net حدود ۳۵۰۰ کلاس در رابطه با موارد مختلف وجود دارد، اما سوال این است که چگونه به عنوان یک برنامه نویس می توانید کلاس مورد نظرتان را در بین این کلاس ها پیدا کنید؟

کلاس های موجود در چارچوب Net به چندین گروه مختلف به نام فضای نام تقسیم می شوند که هر کدام از آنها حاوی چندین کلاس مرتبط به هم است. به این ترتیب اگر به دنبال کلاسی برای یک کار خاص باشید می توانید فقط کلاس های داخل فضای نام مربوط به آن کار را جستجو کنید. خود این فضای نام ها به صورت سلسله مراتبی هستند، به این معنی که یک فضای نام خود نیز می تواند شامل چندین فضای نام دیگر باشد، که آنها نیز به نوبه خود کلاس های داخل آن فضای نام را دسته بندی می کنند.

بیشتر کلاس های موجود در چارچوب Net در فضای نام System و یا فضای نام های موجود در آن دسته بندی شده اند. برای مثال :

System.Data : شامل کلاس هایی برای دسترسی به اطلاعات ذخیره شده در یک بانک اطلاعاتی است.

System.Xml : شامل کلاس هایی برای خواندن و نوشتن سند های XML است.

System.Windows.Forms : شامل کلاس هایی برای ترسیم یک Form و کنترل های آن روی صفحه نمایش است.

System.Net : شامل کلاس هایی برای ایجاد ارتباطات شبکه ای در برنامه است.

در واقع فضای نام ساختاری است برای گروه بندی کلاس های مرتبط. کلاس های مرتبط به هم را می توان در یک فضای نام تعریف کرد تا ساختار برنامه و استفاده از کلاس ها،

روشن و گویا باشد. به طور مثال کلیه کلاس هایی که مربوط به عملیات خواندن و نوشتن اطلاعات به وسیله واسط ها هستند در فضای نامی به نام I/O تعریف می شوند. در داخل یک فضای نام می توان علاوه بر تعریف کلاس ها، فضای نام های دیگری را نیز تعریف کرد. البته امکان تعریف کلاس های هم نام در فضای نام های متفاوت وجود دارد. نحوه تعریف یک فضای نام به صورت زیر است :

```
namespace نام دلخواه
{
    تعریف اختیاری فضای نام های داخلی که در ادامه بررسی خواهند شد
    تعریف کلاس ها
}
```

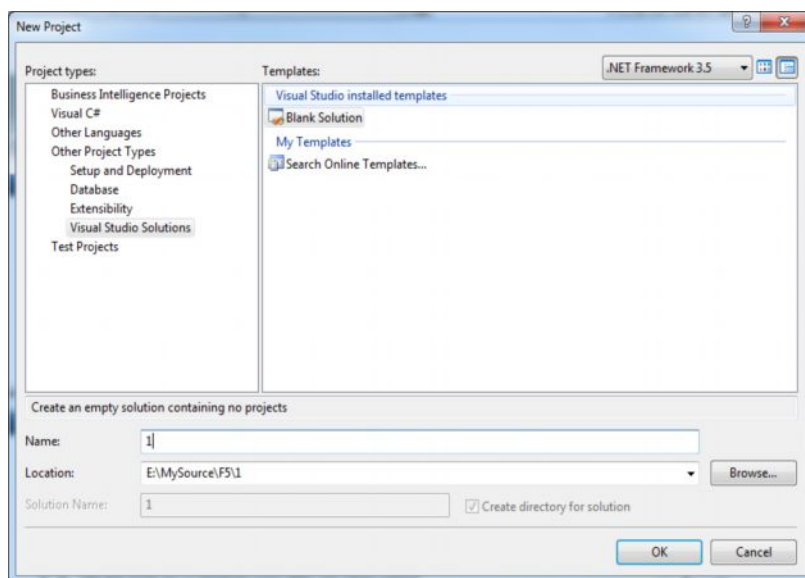
مثال :

```
namespace IO
{
    namespace IO1
    {
        public class class1;
        {
            -
            -
        } //end of class1
        public class class2;
        {
            -
            -
        } //end of class2
        -
        -
    } //end of name space
    public class class3;
    {
    } //end of class3
    public class class1;
    {
    } //end of class1
} //end of name space IO
```

همانطور که مشاهده می کنید کلاسی با نام class1 در هر دو فضای نام IO و IO1 وجود دارد. حال بهتر است با ساختار برنامه، پروژه ها و ارتباط آنها با فضای نام ها آشنا شوید. برنامه شما در واقع Soultion است که می تواند از یک یا چند پروژه تشکیل شده باشد. این

پروژه ها می توانند پروژه ویندوزی (Windows Application)، پروژه وب سایت (Web Site)، کتابخانه کلاس (Class Library) و ... باشند.

به طور پیش فرض با ایجاد هر پروژه در برنامه، یک فضای نام، با عنوان پروژه ایجاد می شود که تمام کلاس ها، فرم ها و اجزائی که به آن پروژه اضافه می شوند در آن فضای نام قرار می گیرند. ادامه مباحث را به صورت عملی بررسی می کنیم تا مطالب گویاتر باشد. برنامه ویژوال استودیو را اجرا کرده و از منوی File گزینه New و سپس Project را انتخاب کنید تا پنجره New Project مطابق پنجره تصویر ۳ - ۵ ظاهر گردد.

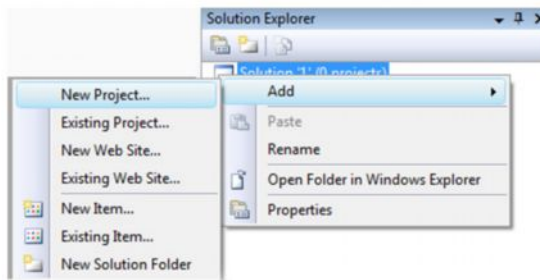


تصویر ۳ - ۵

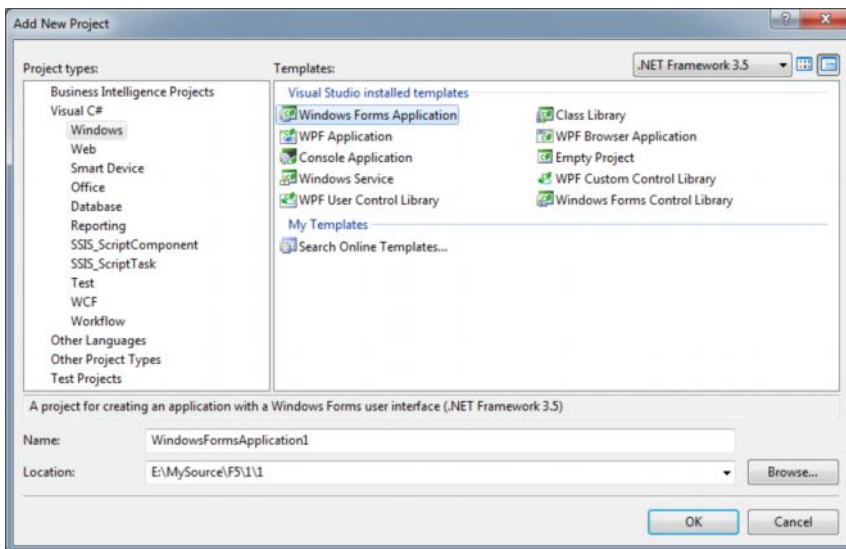
در پنجره New Project از بخش Project types بخش Other Project Types را انتخاب کرده و سپس گزینه Visual Studio Solutions را انتخاب کنید و از بخش Templates گزینه Blank Solution را انتخاب کنید. از بخش Name نام پروژه و از بخش Location محل ذخیره پروژه خود را انتخاب کرده و سپس بر روی دکمه OK کلیک کنید.

نکته! پروژه های از نوع Blank Solution همانند پروژه های Windows Application هستند با این تفاوت که در آغاز ساخت پروژه فاقد Form هستند.

اگر در سمت راست ویژوال استودیو، پنجره Solution Explorer را مشاهده نمی کنید، می توانید از منوی View گزینه Solution Explorer را انتخاب کنید. همانطور که در پنجره تصویر ۴ - ۵ مشاهده می کنید، برنامه شما (Solution) در حال حاضر شامل هیچ پروژه ای نیست یک پروژه از نوع Windows Application به آن اضافه می کنیم. برای این کار در پنجره Solution Explorer بر روی Solution کلیک راست کرده و گزینه Add و سپس New Project را کلیک می کنیم در پنجره New Project یک پروژه از نوع Windows Application ایجاد می کنیم (پنجره تصویر ۴ - ۵ و پنجره تصویر ۵ - ۵).



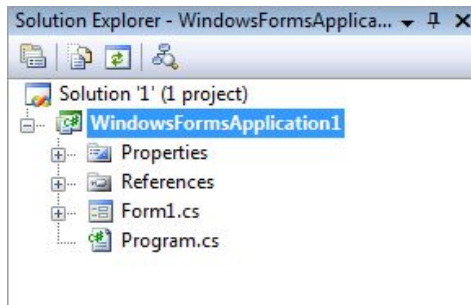
تصویر ۴ - ۵



تصویر ۵ - ۵

حال برنامه شما دارای یک پروژه از نوع پروژه ویندوزی است. همراه با این نوع پروژه یک Form به طور خودکار به پروژه شما اضافه می شود. این موارد در پنجره Solution Explorer قابل مشاهده

است.



تصویر ۶ - ۵

روی این Form با زدن کلید FV به محیط ویرایشگر کد بروید. همانطور که در پنجره تصویر ۶ - ۵ مشاهده می کنید یک فضای نام با عنوان پروژه اضافه شده است یعنی `WindowsFormApplication1` و کلاس `Form1.cs` را نیز در بر گرفته است. هر Form یا جزء دیگری را که به این پروژه اضافه کنید به طور پیش فرض همین فضای نام را خواهد داشت. برای مثال در پنجره Solution Explorer روی پروژه `Windows Application1` کلیک راست کرده و گزینه Add و سپس New Item را انتخاب کنید از پنجره باز شده Windows Form را انتخاب کرده و دکمه Add را کلیک کنید تا `Form2` به پروژه شما اضافه شود. اگر در ویرایشگر کد Form بروید، باز هم مشاهده می کنید که فضای نام `Windows Application1`، کلاس `Form2` را در بر گرفته است. در حال حاضر این فضای نام شامل دو کلاس `Form1` و `Form2` می باشد. همانطور که می بینید یک فضای نام می تواند در چند فایل، کلاس ها را به هم مرتبط و مربوط سازد. برای تفهیم بیشتر مطالب، یک مثال دیگر را مورد بررسی قرار می دهیم.

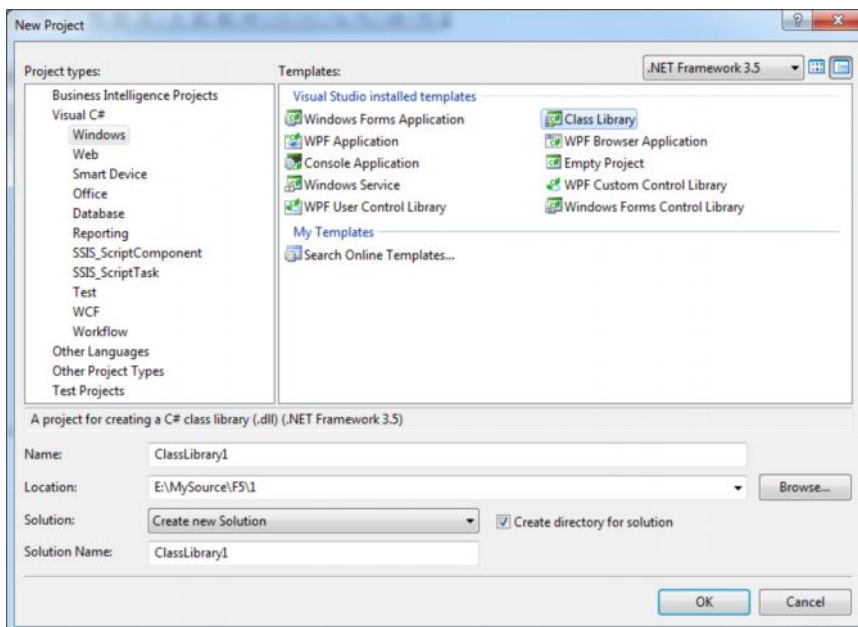
فرض کنید قصد دارید یک کتابخانه (Library) از کلاس های مفید در زمینه های توابع ریاضی، تاریخ و ... ایجاد کنید. برای این کار یک پروژه از نوع Class Library، با نام `ClassLibrary1` به برنامه خود اضافه می کنیم.

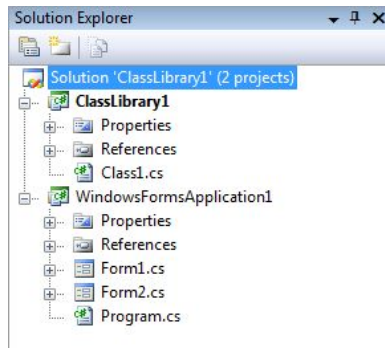
`Class Library` یکی از انواع پروژه های موجود در ویژوال استودیو است. برنامه یا کد هایی که در یک پروژه `Class Library` نوشته می شود دارای خروجی به شکل یک Application نیست یعنی خروجی قابل اجرا ندارد و فایلی که ساخته می شود با پسوند `dll` (Dynamic

Link Library) است. فایل های dll فایل هایی هستند که توسط فایل های Windows Application یا Console Application و یا سایر پروژه ها استفاده می شوند.

در فضای نام ClassLibrary^۱ قصد داریم دو فضای نام برای جهت قرار گیری کلاس های ریاضی و تاریخ به نام های MathSpace و DateSpace ایجاد کنیم. از طرف دیگر در فضای نام MathSpace می خواهیم توابع مربوط به کار با رشته و اعداد در کلاس و فایل جداگانه، به ترتیب با نام های StringClass و NumericClass قرار گیرند و به همین ترتیب در فضای نام DateSpace نیز، توابع مربوط به کار با تاریخ شمسی و میلادی در کلاس و فایل جداگانه، به ترتیب با نام های HijriClass و MiladClass ایجاد گردند. مراحل زیر را دنبال کنید :

ابتدا، از منوی File گزینه Add و سپس New Project را انتخاب کنید سپس در پنجره New Project و از بخش Template گزینه Class Library را انتخاب کنید (پنجره تصویر ۷ - ۵). توجه کنید که در بخش Name و Location عنوان و مسیر پروژه را تغییر ندهید. با این کار پروژه ای با نام ClassLibrary به برنامه شما اضافه می شود که در Solution Explorer قابل مشاهده است (پنجره تصویر ۸ - ۵).

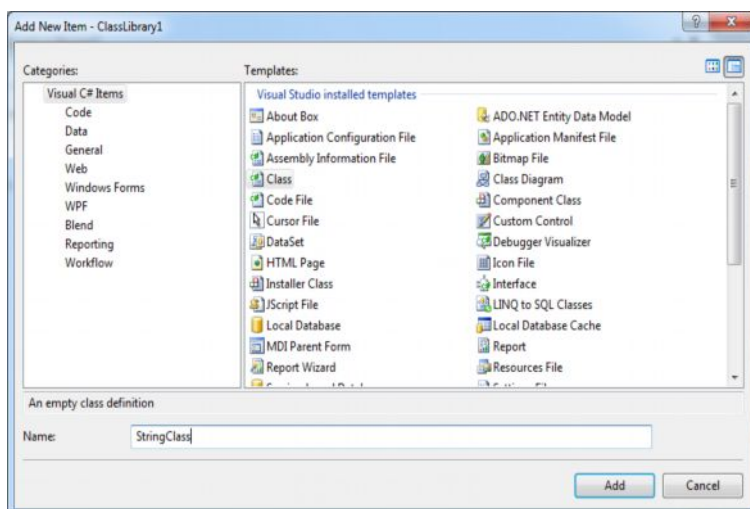




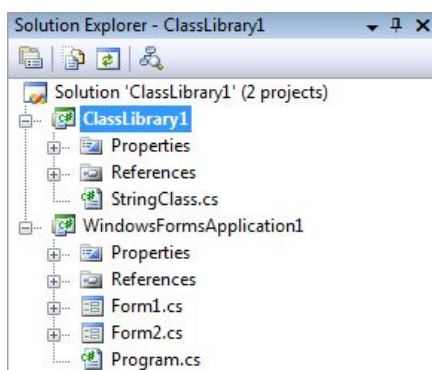
تصویر ۸ - ۵

در حال حاضر برنامه شما شامل دو پروژه از نوع های `WindowsFormsApplication` و `ClassLibrary1` می باشد. همانطور که در پنجره تصویر ۸ - ۵ مشاهده می کنید به طور خودکار یک کلاس به نام `Class1.cs` به پروژه اضافه شده است و در ویرایشگر کد در فضای نام `ClassLibrary1` قابل مشاهده است. برای ایجاد کلاس با نام و ساختار مد نظر، این کلاس را حذف می کنیم. البته امکان تغییر این کلاس نیز وجود دارد. برای این کار در پنجره `Solution Explorer` بر روی نام `Class1` راست کلیک کرده و گزینه `Delete` را کلیک می کنیم تا `Class1` حذف شود. حال برای اضافه کردن کلاس های مورد نظر خود، به ترتیب زیر عمل می کنیم :

در پنجره `Solution Explorer` برنامه بر روی پروژه `Class Library` کلیک راست کرده و گزینه `Add` و سپس `New Item` را کلیک می کنیم. در پنجره باز شده در قسمت `Template` گزینه `Class` را انتخاب کرده و در بخش `Name` عنوان کلاس را `StringClass` وارد می کنیم (پنجره تصویر ۹ - ۵).



تصویر ۹ - ۵



تصویر ۱۰ - ۵

حال اگر در ویرایشگر کد این کلاس دقت کنید می بینید که این کلاس در فضای نام ClassLibrary1 قرار دارد.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassLibrary1
{
    class StringClass
    {
    }
}
```

برای این که این کلاس در ساختار مورد نظر که درباره آن توضیح داده شد، قرار گیرد به شکل زیر عمل می کنیم :

در فضای نام ClassLibrary1 با نوشتن کد زیر فضای نام MathSpace را ایجاد می کنیم :

```
namespace MathSpace
{

}
```

حال کد مربوط به StringClass را به صورت زیر به داخل این فضای نام انتقال می دهیم :

```
class StringClass
{

}
```

نتیجه حاصل در دستورات زیر قابل مشاهده است.

```
namespace ClassLibrary1
{
    namespace MathSpace
    {
        class StringClass
        {
        }
    }
}
```

در حال حاضر در فضای نام ClassLibrary1، فضای نام دیگری با نام MathSpace ایجاد کرده و در این فضای نام نیز یک کلاس با نام StringClass ایجاد کرده ایم. به طور مشابه مراحل بالا را برای ایجاد کلاس NumericClass در فضای نام MathSpace طی می کنیم. بر روی ClassLibrary1 راست کلیک کرده و سپس گزینه Add و New Item را انتخاب می کنیم و در پنجره Add New Item و از بخش Template گزینه Class را انتخاب کرده و در بخش Name عنوان کلاس خود را NumericClass قرار می دهیم.

فضای نام MathSpace در فضای نام ClassLibrary1 ایجاد شده و کلاس StringClass نیز درون این فضای نام قرار دارد. و این فضای نام دو کلاس Numeric Class و String Class را در دو فایل جداگانه در بر می گیرد. البته ذکر این نکته نیز ضروری است که دو کلاس را در یک فایل نیز می توان قرار داد.

```
namespace ClassLibrary1
{
    namespace MathSpace
    {
        class NumericClass
        {
        }
    }
}
```

به روشی که کلاس NumricClass و StringClass را ایجاد کردیم به همین روش کلاس های HijriClass و MiladiClass را ایجاد می کنیم. با این تفاوت که برای این کلاس ها در فضای نام ClassLibrary1 در فایل های ایجاد شده، فضای نام DateSpace را در ایجاد می کنیم.

<pre>namespace ClassLibrary1 { namespace DateSpace { class HijriClass { } } }</pre>	<pre>namespace ClassLibrary1 { namespace DateSpace { class MiladiClass { } } }</pre>
---	--

با ایجاد این کلاس ها، فضای نام ClassLibrary1 شامل دو فضای نام با نام های MathSpace و DateSpace می باشد که هر کدام به ترتیب شامل کلاس های StringClass و NumericClass و کلاس های HijriClass و MiladiClass بوده و همچنین در یک فایل جداگانه قرار دارد.

نکته دیگری که باید به آن پردازیم این است که امکان ادغام کلاس ها در به صورت یک فایل یکسان نیز وجود دارد. بطور مثال فرض کنید در مثال بالا قصد داریم کلاس های StringClass و NumericClass را در یک فایل داشته باشیم. برای این کار می توانیم از پنجره Solution Explorer، کلاس یا فایل StringClass.cs را حذف کنیم و در ویرایشگر کد فایل NumericClass.cs دوباره کلاس StringClass را در فضای نام MathSpace و بعد از تعریف کلاس NumericaClass قرار دهیم. در این حالت کد حاصل مشابه زیر خواهد بود :

```
namespace ClassLibrary1
{
    namespace MathSpace
    {
        class NumericClass
        {
        }
        class StringClass
        {
        }
    }
}
```

در این صورت باز هم فضای نام MathSpace شامل کلاس های StringClass و NumericClass می باشد با این تفاوت که این دو کلاس در یک فایل وجود دارند. البته تغییر نام فایل ها نیز ممکن است. اتخاذ تصمیم برای جداسازی فایل ها معمولاً به نظر طراح بستگی دارد و با در نظر گرفتن ساختار فضا های نام و کلاس های مربوطه، طبقه بندی موضوعی کلاس ها و حجم کلاس های موجود در فایل انجام می شود.

توجه کنید هنگامی که یک کلاس در فضای نام قرار می گیرد، برای دسترسی به آن علاوه بر ذکر نام کلاس، باید فضای نام آن را نیز مشخص کنیم. البته تاکنون در برنامه ها از نام کوتاه شده آنها، یعنی فقط نام خود کلاس استفاده کرده و فضای نام آن را مشخص نمی کردید. تمام کلاس ها از کلاس object مشتق می شوند، در حقیقت نام کلاس را خلاصه کردیم زیرا کلاس object در فضای نام System قرار دارد و همه کلاس ها در واقع از کلاس System.object مشتق می شوند. همچنین کلاس Console در واقع نام خلاصه شده کلاس System.Console است و کد Console.ReadLine() با کد System.Console.ReadLine() برابر است.

هر کلاس فقط می تواند به یک فضای نام تعلق داشته باشد و همچنین نمی تواند عضو هیچ فضای نامی نباشد. به عبارت دیگر هر کلاس باید دقیقاً در یک فضای نام قرار داشته باشد.

استفاده از کلاس های موجود در فضا های نام دیگر

استفاده زیاد از فضا های نام گاهی خسته کننده است و قابلیت مشخص کردن یک کلاس مشخص با مشخصات خاص همیشه لازم نیست. زبان C# به شما اجازه می دهد تا نام کامل یک کلاس را خلاصه بندی کنید. برای استفاده از یک کلاس بدون ذکر فضای نام آن، باید فضای نام آن کلاس را با استفاده از دستور using به برنامه اضافه کنیم. اهمیت استفاده از این

دستور ممکن است هنگامی که در حال کار با برنامه های کوچک هستید نمایان نشود، اما فرض کنید بخواهید به کلاسی مانند ServiceDescription در فضای نام System.Web.Services.Description دسترسی پیدا کنید. واضح است که ذکر اسم کلاس و فضای نام آن در هر بار استفاده از این کلاس کار بسیار سختی است. بنابراین بهتر است که با اضافه کردن فضای نام آن، هر مرتبه فقط نام کلاس را ذکر کنید. تمام فضای نام هایی که می خواهید به یک برنامه اضافه کنید، باید در ابتدای کد و قبل از هر دستوری (قبل از دستور namespace که برای مشخص کردن فضای نام برنامه به کار می رود)، با استفاده از دستور using مشخص شوند.

به عنوان مثال فرض کنید که شما در کلاس HijriClass قصد استفاده از بعضی توابع کلاس NumericClass را دارید از آنجا که کلاس HijriClass در فضای نام DateSpace وجود دارد و کلاسی که قصد استفاده از آن را دارید در فضای نام دیگری یعنی فضای نام MathSpace قرار دارد در بالای فایل cs کلاس HijriClass مسیر کامل فضای نام داخلی MathSpace را جلوی کلمه using ذکر می کنیم یعنی :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ClassLibrary1.MathSpace;

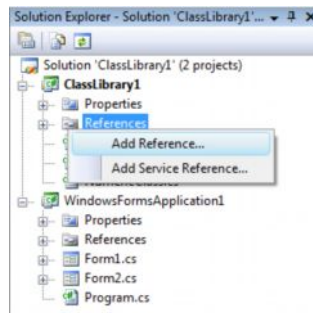
namespace ClassLibrary1
{
    namespace DateSpace
    {
        class HijriClass
        {
        }
    }
}
```

همانطور که مشاهده می کنید برای اشاره به فضای نام داخلی از نام فضای بیرونی استفاده شده است و سپس از نقطه استفاده شده و بعد فضای نام داخلی به آن اضافه شده است که این حالت می تواند برای فضا های نام داخلی تر در صورت وجود تکرار شود.

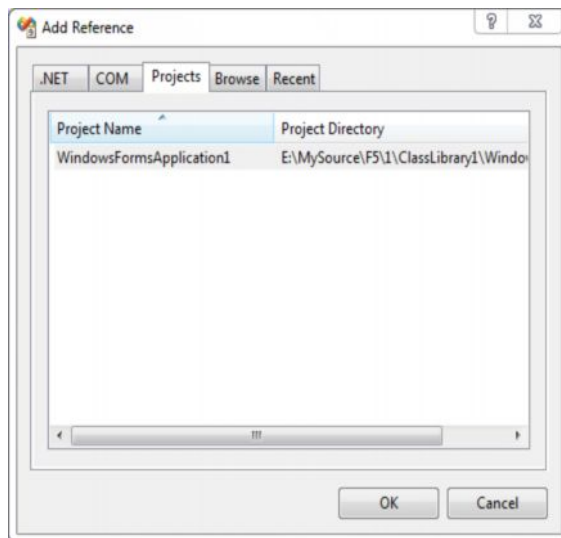
بعد از انجام این کار در هر جا از این فایل cs (در کلاس HijriClass یا ...) نیاز به استفاده از کلاس ها یا فضا های نام موجود در MathSpace باشد تنها ذکر نام کلاس یا فضای نام مربوطه کافی است. برای مثال برای دسترسی به کلاس کار با اعداد تنها استفاده از نام

NumericClass کافی است. در صورتی که از روش using استفاده نکنیم کل مسیر می بایست در هر بار استفاده ذکر شود. برای مثال در زمان استفاده، برای دسترسی به کلاس NumericClass می بایست مسیر کامل `ClassLibrary1.MathSpace.NumericClass` ذکر شود.

در صورتی که قصد استفاده از فضای نامی را دارید که جزء فضای نام های پیش فرض .Net Framework نمی باشد یا فضای نامی است که از پروژه دیگر می باشد خودتان یا دیگران ایجاد کرده اید قبل از مراحل بالا باید ADD Refrence را انجام دهید. بدین منظور در قسمت Solution Explorer بر روی پوشه Refrence راست کلیک کرده و گزینه Add Refrence را انتخاب کنید.



تصویر ۱۱ - ۵



تصویر ۱۲ - ۵

همانطور که در پنجره تصویر ۱۲ - ۵ مشاهده می کنید پنجره Add Reference باز شده است در این پنجره می توانید از طریق بخش Projects فضای نام پروژه مورد نظر خود را اضافه کنید و از دستور using استفاده کنید در بخش های بعدی به طور یک مثال این عملیات را بررسی خواهیم کرد.

تنها مشکلی که هنگام استفاده از دستور using ممکن است رخ دهد این است که اگر دو فضای نام توسط دستورات using مورد ارجاع قرار گیرد به نحوی که شامل دو نوع هم نام باشد از شکل کامل نام یا حداقل بلندترین شکل آن استفاده کرد تا از دستیابی کامپایلر به نوع صحیح آن اطمینان حاصل کرد. به طور مثال فرض کنید کلاسی با نام NamespaceExample در دو فضای نام Wrox.ProCSharp.Basics و Wrox.ProCSharp.OOP قرار دارند. اگر کلاسی با نام Test در فضای نام Wrox.ProCSharp ایجاد کرده و از یکی از کلاس های NamespaceExample در این کلاس نمونه سازی کنید، در این صورت باید مشخص کنید که از کدامیک از این دو کلاس استفاده کرده اید :

```
using Wrox.ProCSharp;
```

```
class Test
{
public static int Main()
{
Basics.NamespaceExample nEx = new Basics.NamespaceExample();
// do something with the nEx variable
return 0;
}
}
```

از آنجا که دستورات using در بالای فایل های C# قرار می گیرند، برنامه نویسان C++ و C ممکن است به راحتی فضا های نام را با سر فایل های (Header) و دستور #include در این دو زبان اشتباه بگیرند. مراقب باشید که این اشتباه را مرتکب نشوید. دستور using هیچ گونه پیوند فیزیکی بین فایل ها اعمال نمی کند و زبان C# در این زمینه هیچ شباهتی به فایل های Header در زبان C++ و C ندارد.

کاربرد دیگر کلمه کلیدی using تعیین اسامی مستعار برای کلاس ها و فضا های نام است. اگر عنوان فضای نام مورد نظر شما آنقدر طولانی باشد و شما قصد داشته باشید در کد خود

چندین بار به این نام رجوع کنید و نیز به منظور اجتناب از تداخل نام نوع، نمی خواهید که آن را در یک دستور using قرار دهید، می توانید برای آن یک نام مستعار مشخص کنید. برای تعریف نام مستعار برای یک کلاس به روش زیر عمل می شود :

مسیر کامل کلاس = نام مستعار دلخواه برای کلاس using

مسیر کامل کلاس شامل مسیر فضای نام های در برگرفته نام کلاس است. استفاده از نام مستعار در دستور using تنها در موردی است که انتهای using به نام یک کلاس ختم می شود و نه به یک فضای نام. به مثال زیر توجه کنید :

```
using System;
using Introduction = wrox.ProCSharp.Basics;
class test
{
    public static int Main()
    {
        Introduction : : NamespaceExample NSEx =
        new Introduction : : NamespaceExample ();
        console.WriteLine (NSEx.GetNamespace ());
        return 0;
    }
}
namespace wrox.ProCSharp.Basics
{
    class NamespaceExample
    {
        public string GetNamespace ()
        {
            return this.GetType ().Namespace;
        }
    }
}
```

دستورات بالا نسخه تصحیح شده مثال قبل است، نام مستعار Introduction را به فضای نام wrox.ProCSharp.Basics نسبت می دهد و از این نام برای نمونه سازی شیء NamespaceExample استفاده می کند که در این فضای نام تعریف شده است. به نحوه استفاده از (::) توجه کنید این کار نام مستعار Introduction را برای فضای نام ذکر شده انتخاب می کند. اگر کلاسی با همین نام در این حوزه تعریف شده باشد، تداخل نام به وجود خواهد آمد. عملگر :: به نام مستعار اجازه می دهد که حتی با وجود تداخل نام مورد ارجاع قرار گیرد. کلاس NamespaceExample متدی به نام GetNamespace() دارد که از متد

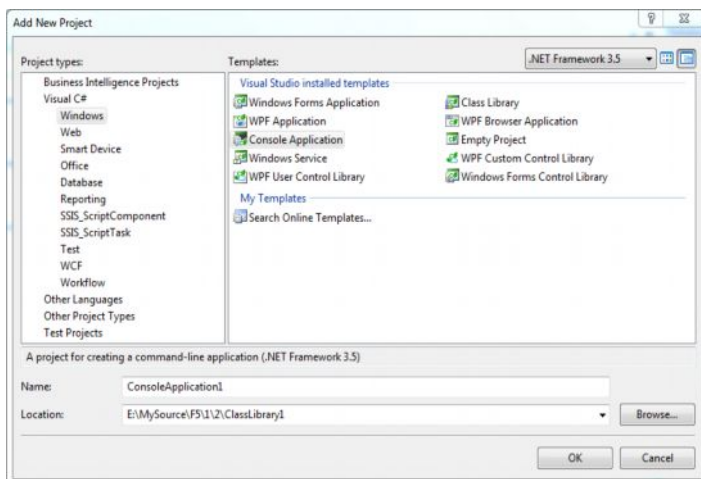
GetType() که توسط همه کلاس ها برای دسترسی به یک شیء Type که نشانگر نوع کلاس است، استفاده می کند. شما از این شیء به منظور بازگرداندن یک فضای نام یک کلاس استفاده می کنید.

مثال ۱ - ۵: برنامه ای بنویسید که ابتدا یک کلاس را تعریف کرده و با استفاده از یک متد ساده یک پیغام چاپ کند. سپس از این کلاس در پروژه های خود استفاده کنید.

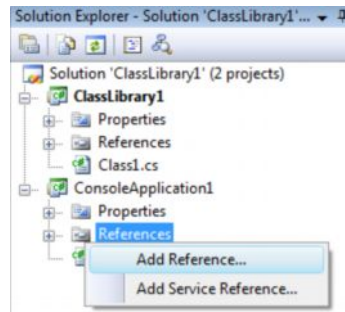
حل: ابتدا یک پروژه از نوع ClassLibrary ساخته و سپس از این کلاس در یک پروژه از نوع Console استفاده خواهیم کرد. ابتدا شروع به کد نویسی دستورات آن می کنیم.

```
namespace ClassLibrary1
{
    public class Class1
    {
        public void Method()
        {
            Console.WriteLine("Hello Ardebil");
        }
    }
}
```

در پنجره Solution Explorer بر روی Solution خود راست کلیک کرده و گزینه Add و سپس New Project را کلیک می کنیم. در پنجره New Project گزینه Console Application را مطابق پنجره تصویر ۱۳ - ۵ انتخاب کرده و مسیر و عنوان پروژه خود را به صورت پیش فرض قبول می کنیم.

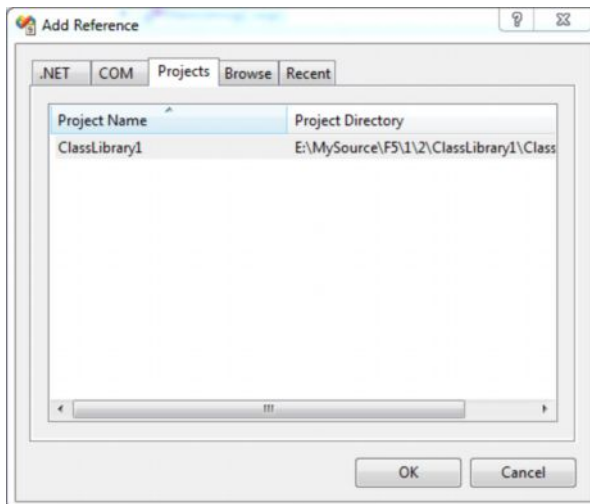


سپس در پروژه Console Application بر روی پوشه Refrence راست کلیک کرده و گزینه Add Refrence را کلیک می کنیم.



تصویر ۱۴ - ۵

در پنجره Add Refrence به تب Project رفته و ClassLibrary۱ را انتخاب کرده و بر روی گزینه OK کلیک کنید.

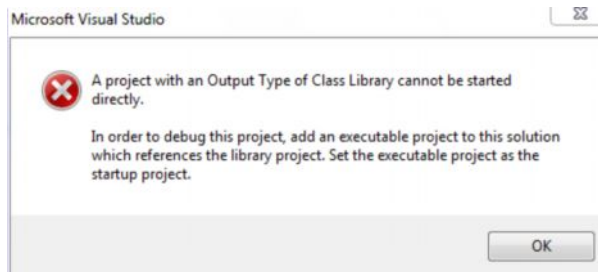


تصویر ۱۵ - ۵

با این کار ClassLibrary۱ به پروژه شما اضافه می شود و می توانید به راحتی از کلاس ها و متد های موجود در آن پروژه خود استفاده کنید. حال در پنجره Solution Explorer بر روی Program.cs دابل کلیک کنید و سپس دستورات زیر را بنویسید. از بخش using کد using ClassLibrary۱ را به پروژه خود اضافه می کنیم و از class۱ که در ClassLibrary۱ نوشتیم یک نمونه به نام test می سازیم. از متد Method در این نمونه استفاده می کنیم.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ClassLibrary1;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Class1 test = new Class1();
            test.Method();
        }
    }
}
```

حال اگر برنامه را اجرا کنید احتمالاً با خطای زیر مواجه خواهید شد.



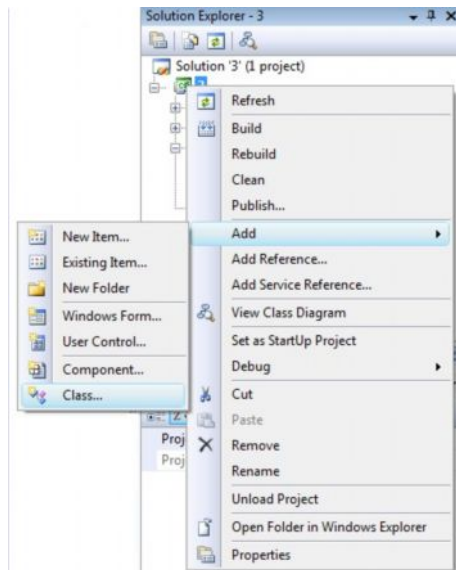
تصویر ۱۶ - ۵

اگر یادتان باشد گفتیم که پروژه های نوع Class Library خروجی ندارند. برای نمایش خروجی کافی است که در پنجره Solution Explorer بر روی پروژه Console Application راست کلیک کرده و گزینه Set as StartUp Project را انتخاب کنید تا خروجی پروژه بر روی آن قرار گیرد. حال اگر برنامه را اجرا کنید می توانید خروجی را مشاهده کنید.

مثال ۲ - ۵: در این مثال قصد داریم نحوه کارکرد دو کلاس از نوع public و private را نشان دهیم و نیز چگونگی استفاده از یک متد و تاثیر آن بر یکدیگر را بررسی کنیم.

حل : ابتدا یک پروژه از نوع Console Application ساخته و سپس در داخل این پروژه یک کلاس با نام myclass ایجاد می کنیم. در این بخش می خواهیم کلاس مزبور را در داخل فضای نام پروژه Console ایجاد کنیم. اکنون در پنجره Solution Explorer بر روی عنوان

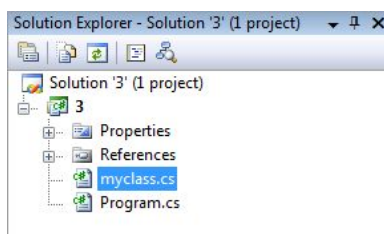
پروژه Console خود راست کلیک کرده و گزینه class را انتخاب می کنیم (مطابق پنجره تصویر ۱۷ - ۵).



تصویر ۱۷ - ۵

در پنجره Add New Item گزینه Class را انتخاب کرده و در بخش Name عنوان مورد نظر را تایپ کرده و سپس بر روی دکمه Add کلیک می کنیم. ما در اینجا عنوان کلاس داخلی خود را myclass انتخاب می کنیم.

دقت داشته باشید کلاسی که در این بخش ساختیم با کلاس پروژه قبلی تفاوت هایی دارد این کلاس در داخل همین پروژه ساخته شده است و به عنوان کلاس داخلی این پروژه محسوب می شود و نیازی به تعریف فضای نام ندارد چون این کلاس خود جزوه فضای نام همین پروژه هست. در پنجره تصویر ۱۸ - ۵ می توانید این عملیات رو مشاهده کنید.



تصویر ۱۸ - ۵

حالا دستورات زیر را می نویسیم.

```
namespace _3
{
    class myclass
    {
        private int Compute1()
        {
            return 1; // Private instance method that computes something.
        }

        public int Compute2()
        {
            return this.Compute1() + 1; // Public instance method.
        }

        private static int Compute3()
        {
            return 3; // Private static method that computes.
        }

        public int Compute4()
        {
            return Compute3() + 1; // Public static method.
        }
    }
}
```

بر روی Program.cs در پنجره Solution Explorer کلیک کرده و دستورات زیر تایپ می کنیم :

```
static void Main(string[] args)
{
    // Create new instance of the Test class.
    // ... You can only call the Compute2 public instance method.
    myclass test = new myclass();
    Console.WriteLine(test.Compute2());
    // Call the public static method.
    // ... You cannot access the private static method.
    Console.WriteLine(test.Compute4());
}
```

پس اگر برنامه را اجرا کنید می توانید خروجی را مشاهده کنید. متد های Compute2 و Compute4 نتیجه محاسبات انجام شده در کلاس myclass را منعکس می کنند.

مثال ۳ - ۵: برنامه ای بنویسید که طول و عرض مستطیلی را خوانده محیط و مساحت آن را محاسبه می کند.

حل : ابتدا یک پروژه از نوع Window ساخته و Form آن را مطابق پنجره تصویر ۱۹ - ۵ طراحی می کنیم :



تصویر ۱۹ - ۵

حال یک کلاس با عنوان rectangle از نوع داخلی ساخته مطابق روشی که در پروژه قبلی ساختیم و دستورات زیر را در آن تایپ می کنیم :

کلاس rectangle از اعضای زیر تشکیل شده است :

خاصیت X : طول مستطیل (فیلد x) را مقدار دهی و بازیابی می کند.

خاصیت Y : عرض مستطیل (فیلد y) را مقدار دهی و بازیابی می کند.

فیلد prime : محیط مستطیل را نگهداری می کند.

فیلد area : مساحت مستطیل را نگهداری می کند.

متد calculateArea : مساحت مستطیل را محاسبه می کند.

متد calculatePrime : محیط مستطیل را محاسبه می کند.

متد getArea : مساحت مستطیل را بر می گرداند.

متد getPrime : محیط مستطیل را بر می گرداند.

```
class rectangle
{
    private int x;
    private int y;
    private int area;
    private int prime;
    public int X
    {
        get
        {
            return x;
        }
        set
        {
            x = value;
        }
    }
    public int Y
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
        }
    }
    public void calculateArea()
    {
        area = x * y;
    }
    public void calculatePrime()
    {
        prime = (x + y) * 2;
    }
    public int getArea()
    {
        return (area);
    }
    public int getPrime()
    {
        return prime;
    }
}
```

بنابراین به Form طراحی برگشته و بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    rectangle rect = new rectangle();
    rect.X = System.Convert.ToInt32(textBox1.Text, 10);
    rect.Y = System.Convert.ToInt32(textBox2.Text, 10);
    rect.calculateArea();
    rect.calculatePrime();
    label4.Text = "مساحت = " + System.Convert.ToString(rect.getArea())
        + " محیط = " + System.Convert.ToString(rect.getPrime());
}
```

کافی است که برنامه را اجرا کرده و نتیجه را مشاهده کنید.

مثال ۴ - ۵: برنامه ای بنویسید که با دریافت مقادیر n و x مقدار سری زیر را محاسبه کند؟

$$S = \sum_{i=1}^n \frac{x^{3i+1}}{3i+1} = \frac{x^4}{4} + \frac{x^7}{7} + \frac{x^{10}}{10} + \dots + \frac{x^{3n+1}}{3n+1}$$

حل : در این بخش قصد داریم یک متد را درون خود پروژه ایجاد کرده و فراخوانی کنیم. ابتدا یک پروژه از نوع Console ایجاد می کنیم. برای نوشتن یک متد لازم نیست که کلاس جدیدی ایجاد کنیم در داخل کلاس Program.cs هم می توانیم این متد را بنویسیم. سپس یک متد به نام CalPow می نویسیم که مقدار پارامتر اول را به توان پارامتر دوم می رساند و لذا از این متد در محاسبه مقدار سری بهره می بریم.

```
static ulong calPow (uint m , uint n)
{
    ulong temp = 1;
    for (uint i = 1; i <= n; i++)
        temp *= m;
    return temp;
}
```

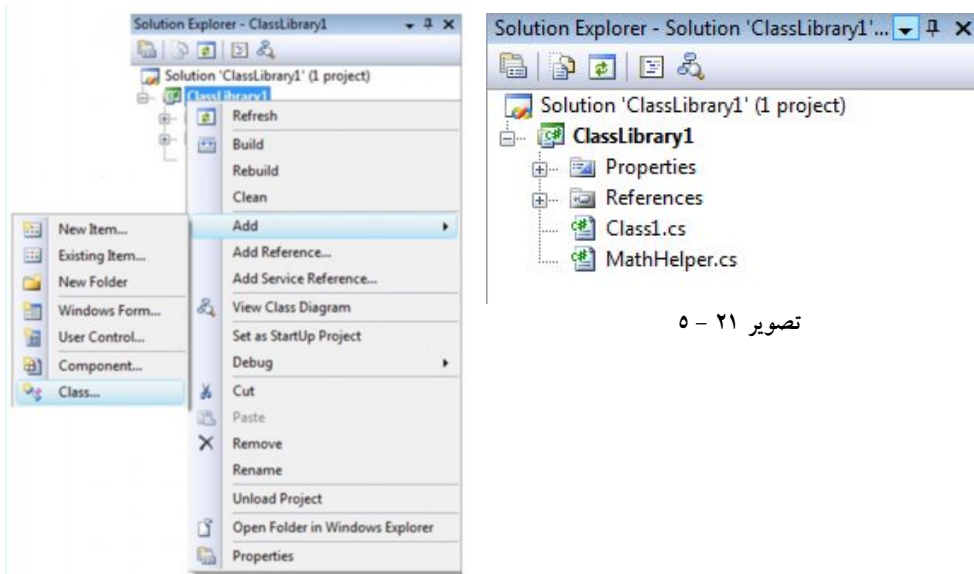
پس به متد اصلی برنامه رفته و دستورات زیر را می نویسیم.

```
static void Main(string[] args)
{
    Console.Write("Enter n?");
    uint n = uint.Parse(Console.ReadLine());
    Console.Write("Enter x?");
    uint x = uint.Parse(Console.ReadLine());
    double sum = 0.0;
    for (uint i = 1; i <= n; i++)
        sum += calPow(x, 3 * i + 1) / (3 * i + 1);
    Console.WriteLine(sum);
}
```

در این مثال پس از دریافت n و x در یک حلقه for با کمک گرفتن از متد calPow مجموع n جمله اول سری محاسبه شده است.

مثال ۵-۵: کلاسی تعریف کنید که چهار عمل اصلی ریاضی را انجام دهد. از این کلاس در یک پروژه ویندوزی استفاده کنید.

حل: ابتدا یک پروژه از نوع Class Library ساخته و سپس داخل آن یک کلاس جدید به نام MathHelper ایجاد می کنیم (پنجره تصویر ۲۰-۵ و ۲۱-۵).



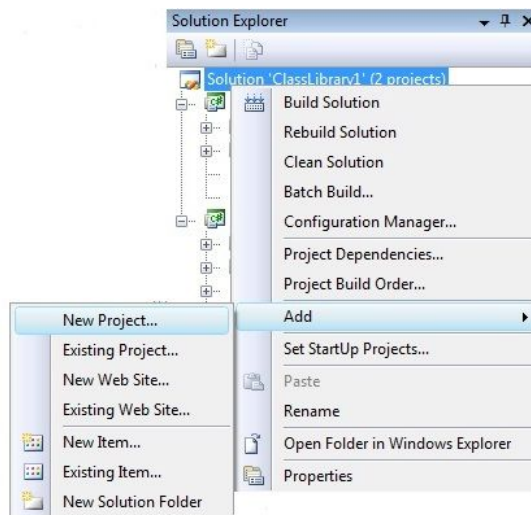
تصویر ۲۱-۵

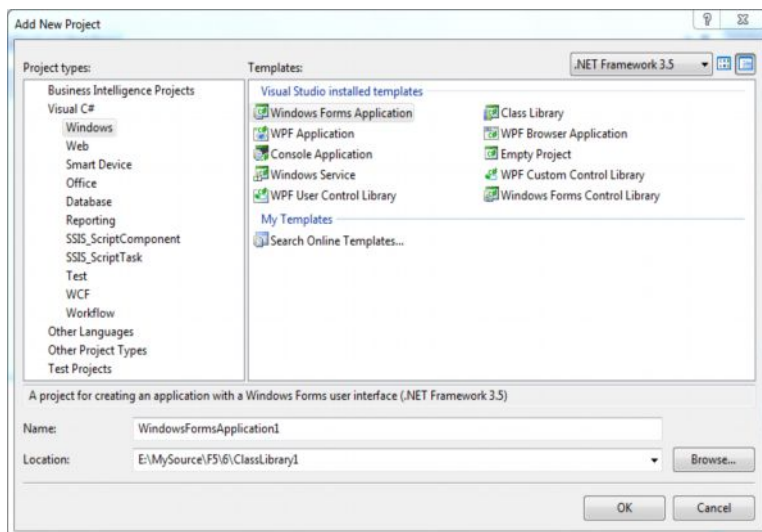
تصویر ۲۰-۵

همانطور که در پنجره تصویر ۲۱-۵ مشاهده می کنید کلاس MathHelper ایجاد شده است. بر روی آن دابل کلیک کنید تا وارد بخش ویرایشگر کد آن شوید، سپس دستورات زیر را بنویسید.

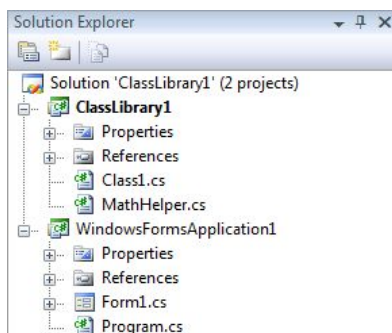
```
public class MathHelper
{
    public int Add(int x, int y)
    {
        return x + y;
    }
    public int Sub(int x, int y)
    {
        return x - y;
    }
    public int Mul(int x, int y)
    {
        return x * y;
    }
    public int Div(int x, int y)
    {
        return x / y;
    }
}
```

حال باید یک پروژه از نوع Windows Application ایجاد کرده و از کلاس MathHelper داخل پروژه ClassLibrary۱ استفاده کنیم. در پنجره Solution Explorer بر روی Solution مورد نظر راست کلیک کرده و از منوی Add گزینه New Project را انتخاب می کنیم. در پنجره New Project گزینه Windows Form Application را انتخاب کرده و بر روی OK کلیک می کنیم.





تصویر ۲۳ - ۵



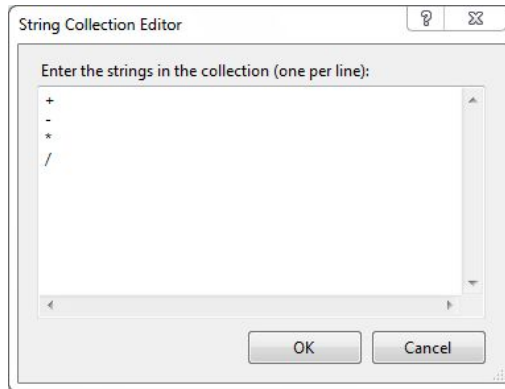
تصویر ۲۴ - ۵

در پروژه WindowsFormsApplication1 بر روی Form1.cs دابل کلیک کنید تا باز شود و آن را مطابق پنجره تصویر ۲۵ - ۵ طراحی کنید.



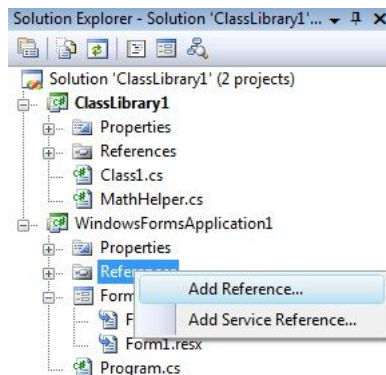
تصویر ۲۵ - ۵

اکنون کنترل ComboBox را انتخاب کرده و به پنجره Properties بروید و خاصیت DropDownStyle را روی DropDownList قرار دهید. و در خاصیت Items مقادیر +، -، *، / را مطابق پنجره تصویر ۲۶ - ۵ وارد کنید.

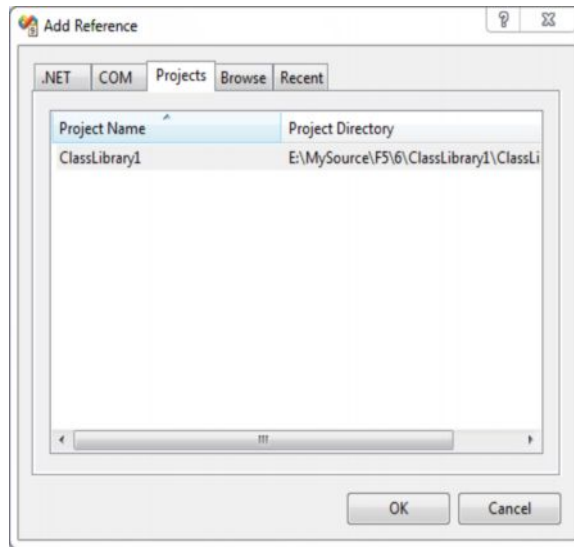


تصویر ۲۶ - ۵

باید پروژه ClassLibrary1 را به داخل پروژه WindowsFormApplication1 وارد کنیم چون کلاس MathHelper داخل پروژه ClassLibrary1 قرار دارد. در پنجره Solution Explorer و از پروژه WindowsFormApplication1 بر روی پوشه Refrence راست کلیک کرده و گزینه Add Refremce را انتخاب می کنیم و در پنجره Add Refrence از تب Project گزینه ClassLibrary1 را انتخاب کرده و دکمه OK را کلیک می کنیم (پنجره تصویر ۲۷ - ۵ و پنجره تصویر ۲۸ - ۵).



تصویر ۲۷ - ۵



تصویر ۲۸ - ۵

حال می توانیم از کلاس MathHelper موجود در پروژه ClassLibrary1 در داخل پروژه WindowsForm1Application استفاده کنیم. بنابراین در پنجره Solution Explorer و در پروژه WindowsForm1Application گزینه Form1 را انتخاب کرده و به Form1 می رویم و بر روی دکمه FV کلیک می کنیم تا وارد ویرایشگر کد برنامه شویم. در بخش using ها دستور using ClassLibrary1 را می نویسم.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using ClassLibrary1;
```

دوباره به Form1 برگشته و بر روی دکمه Click دابل کلیک می کنیم، سپس در رویداد Click این کنترل دستورات زیر را می نویسیم.

```
private void button1_Click(object sender, EventArgs e)
{
    string num1 = textBox1.Text;
    string num2 = textBox2.Text;
    string op = comboBox1.Text;
    int Result = 0;
    int n1 = int.Parse(num1);
    int n2 = int.Parse(num2);
    MathHelper mh = new MathHelper();
    if (op == "+")
    {
        Result = mh.Add(n1, n2);
    }
    if (op == "-")
    {
        Result = mh.Sub(n1, n2);
    }
    if (op == "*")
    {
        Result = mh.Mul(n1, n2);
    }
    if (op == "/")
    {
        Result = mh.Div(n1, n2);
    }
    label5.Text = Result.ToString();
}
```

برنامه را اجرا کنید، فقط قبل از اجرا در پنجره Solution Explorer بر روی پروژه WindowsFormApplication راست کلیک کرده و گزینه Set as Startup Project را کلیک کنید تا شروع پروژه بر روی این پروژه تنظیم شود.

مثال ۶ - ۵: برنامه ای که با استفاده از یک کلاس و چند متد، دو مقدار را از ورودی خوانده و در یک متغیر قرار می دهد سپس محتویات آن را عوض می کند. علاوه بر این، مقداری را در متدی به طور تصادفی تولید کرده و به متد فراخوان بر می گرداند و نمایش می دهد؟

(هدف این مثال کار با پارامترهای out و ref می باشد).

حل: یک پروژه ویندوزی جدید ساخته، سپس Form آن را مطابق پنجره تصویر ۲۹ - ۵ طراحی کنید.



تصویر ۲۹ - ۵

اکنون یک کلاس داخلی طبق روش های گفته شده ساخته و عنوان آن را refout قرار دهید و دستورات زیر را وارد کنید :

```
class refout
{
    public void swap(ref int x, ref int y)
    {
        int temp = (x);
        x = y;
        y = temp;
    }
    public void Rand(out int x)
    {
        Random r = new Random();
        x = r.Next(100);
    }
}
```

متد swap: دو پارامتر از نوع ref را دریافت کرده محتویات دو پارامتر را تعویض می کند.
متد Rand: یک مقدار از نوع int به صورت تصادفی ایجاد کرده و با پارامتر از نوع out بر می گرداند.

حال بر روی دکمه Swap دابل کلیک کنید تا وارد رویداد click آن شوید. سپس دستورات زیر را بنویسید.

```
private void button2_Click(object sender, EventArgs e)
{
    refout temp = new refout();
    int x = System.Convert.ToInt32(textBox1.Text, 10);
    int y = System.Convert.ToInt32(textBox2.Text, 10);
    temp.swap(ref x, ref y);
    textBox1.Text = x.ToString();
    textBox2.Text = y.ToString();
}
```

بر روی دکمه Random دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    refout temp = new refout();
    int x;
    temp.Rand(out x);
    label4.Text = x.ToString();
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال ۷ - ۵: می خواهیم برنامه ای بنویسیم که در آن عملکرد متد های بدون پارامتر و با پارامتر دار (out , ref) را نشان دهیم؟

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و متد های زیر را در کلاس Program.cs تعریف کنید.

```
static void Example1(int value)
{
    value = 1;
}

static void Example2(ref int value)
{
    value = 2;
}

static void Example3(out int value)
{
    value = 3;
}
```

در متد static void Main متد ها را به صورت زیر فراخوانی کنید.

```
static void Main(string[] args)
{
    int val = 0;
    Example1(val);
    Console.WriteLine(val); // Still 0!
    Example2(ref val);
    Console.WriteLine(val); // Now 2!
    Example3(out val);
    Console.WriteLine(val); // Now 3!
}
```

خروجی برنامه به صورت زیر خواهد بود.

0
2
3

اعضای استاتیک (static) کلاس

حافظه Ram برای هر نوع پروسه ای که در آن بارگذاری می شود به سه قسمت تقسیم می شود : Heap، Stack و Static.

(static در NET. در حقیقت قسمتی از Heap است که به آن High Frequency Heap نیز گفته می شود).

قسمت استاتیک حافظه، محل نگهداری متد ها و متغیر های استاتیک است. متد ها و یا متغیر هایی که نیاز به نمونه سازی از کلاس برای ایجاد شدن ندارند، به صورت استاتیک ایجاد می گردند. در زبان C# از واژه کلیدی static برای معرفی آنها کمک گرفته می شود. برای مثال :

```
class MyClass
{
    public static int a;
    public static void DoSomething();
}
```

در این مثال برای فراخوانی متد DoSomething نیازی به ایجاد یک نمونه جدید از کلاس MyClass نیست و تنها کافی است بنویسیم :

```
MyClass.DoSomething(); // and not -> new MyClass().DoSomething();
```

نکته مهمی که در اینجا وجود دارد این است که متد های استاتیک تنها قادر به استفاده از متغیر های استاتیک تعریف شده در سطح کلاس هستند. علت چیست؟

```

class MyClass
{
// non-static instance member variable
private int a;
//static member variable
private static int b;
//static method
public static void DoSomething()
{
//this will result in compilation error as "a" has no memory
a = a + 1;
//this works fine since "b" is static
b = b + 1;
}
}

```

در این مثال اگر متد DoSomething را فراخوانی کنیم، تنها متغیر b تعریف شده در حافظه حضور داشته (به دلیل استاتیک معرفی شدن) و چون باروش فراخوانی MyClass.DoSomething هنوز نمونه ای از کلاس مذکور ایجاد نشده به متغیر a نیز حافظه ای اختصاص داده نشده است و نامعین می باشد. بر این اساس کامپایلر نیز از کامپایل شدن این کد جلوگیری کرده و خطای لازم را اعلام خواهد کرد.

اکنون تعریف یک کلاس به صورت استاتیک چه اثری را خواهد داشت؟ با تعریف یک کلاس به صورت استاتیک مشخص خواهیم کرد که این کلاس تنها حاوی متد ها و متغیر های استاتیک می باشد. امکان ایجاد یک نمونه از آنها وجود نداشته و نیازی نیز به این امر ندارند. این کلاس ها امکان داشتن instance variables را نداشته و به صورت پیش فرض از نوع sealed به حساب خواهند آمد و امکان ارث بری از آنها نیز وجود ندارد. علت این امر هم این است که یک کلاس static هیچ نوع رفتاری را تعریف نمی کند.

پس چرا نیاز به یک کلاس استاتیک ممکن است وجود داشته باشد؟ همانطور که عنوان شد یک کلاس استاتیک هیچ نوع رفتاری را تعریف نمی کند بنابراین بهترین مکان برای تعریف متد های کمکی که به سایر اعضای کلاس های ما وابستگی نداشته، عمومی بوده، مستقل و متکی به خود هستند.

برای مثال متدی را در نظر بگیرید که بجز اعداد، سایر حروف یک رشته را حذف می کند. این متد عمومی است، وابستگی به سایر اعضای یک کلاس یا کلاس های دیگر را ندارد. بنابراین

در گروه متدهای کمکی قرار می گیرد. اگر از افزونه ی ReSharper استفاده نمائید این نوع متدها را به صورت خودکار تشخیص داده و راهنمایی لازم را جهت تبدیل آنها به متدهای استاتیک ارائه خواهد داد.

با کلاسهای استاتیک نیز همانند سایر کلاس های یک برنامه توسط JIT Compiler رفتار می شود، اما با یک تفاوت. کلاسهای استاتیک فقط یکبار هنگام اولین دسترسی به آنها ساخته شده و در قسمت High Frequency Heap حافظه قرار می گیرند.

پس در زبان C# دو نوع متد وجود دارد: متدهای Static و متدهای نمونه. متدهایی که در اعلان خود شامل کلمه کلیدی static باشند از نوع static هستند، بدین معنا که برای فراخوانی آن نیاز به نمونه سازی از کلاس نیست. از روی متدهای استاتیکی نمی توان شیء (Object) ایجاد کرد. از کلاسی که به صورت static تعریف می شود نمی توان توسط کلمه new یک نمونه (instance) گرفت در نتیجه در کل فقط یک نمونه از کلاس خواهیم داشت. این نوع کلاس، فقط می تواند حاوی اعضای static باشد، به عبارت دیگر بایستی همه متدها، فیلدها و خاصیت های تعریف شده باید از نوع static باشند. یکی از مزایای استفاده از این کلاس ها، این است که کامپایلر تضمین می کند به هیچ عنوان نمونه ای از این کلاس ساخته نشود. کلاس های static نمی توانند constructor داشته باشند ولی با این حال می توان از static constructor ها برای مقدار دهی به عناصر static کلاس، استفاده کرد. توجه کنید که static constructor ها پارامتر و modifier ندارند. همچنین برای اینکه بتوانید به اعضای static ای که در کلاس هستند دسترسی داشته باشید، کافی است ابتدا نام کلاس را نوشته و سپس توسط عملگر (.) به آنها دسترسی پیدا کنید.

نمونه ای از متدهای static، متدهای کلاس math هستند که شامل متدهای مربوط به اعمال ریاضی است. به عنوان مثال، متدهای sin() و cos() از کلاس math بدون نمونه سازی از آن قابل استفاده هستند.

مهمترین نکته در استفاده از کلاس های Static، زمان استفاده از آنهاست. زمانی از این نوع کلاس استفاده می کنم که از لحاظ مفهومی در کل پروژه فقط با یک نمونه از آن کلاس کار داشته باشیم و وجود نمونه های دیگه بی معنا باشد. واضح ترین و پر واضح ترین مثالی که

همیشه با آن روبرو هستیم کلاس Program.cs است که در اکثر پروژه های زبان C# وجود دارد.

در مورد کلاسهای static به این نکته هم باید توجه کنیم، با توجه به اینکه در حافظه مستقر هستند، پس نباید شامل شیء های فراوانی از کلاسهای سنگین دیگر باشند زیرا باعث کندی سرعت نرم افزار های تولیدی ما خواهند شد.

در ارتباط با اعضای استاتیک (Static) یک کلاس بایستی به موارد زیر توجه کنید :

۱- اعضای یک کلاس (فیلد ها، خاصیت ها، متد ها و...) می توانند یکی از انواع اعضای شیء (instance member) یا اعضای ایستا (static member) باشند. به طور مثال دستورات زیر دستور اول عضو شیء و دستور دوم عضو استاتیک را نشان می دهد.

```
public int f1;  
public static int f1;
```

۲- اعضای استاتیک عضو کلاس هستند.

۳- از طریق نام کلاس می توان به اعضای استاتیک دسترسی داشت و دسترسی از طریق شیء غیر مجاز بوده و در صورت دسترسی از طریق شیء پیغام خطا دریافت خواهیم نمود. ولی به اعضای شیء، از طریق نام کلاس دسترسی نداریم و اعضای شیء تنها از طریق شیء کلاس در دسترس قرار خواهند گرفت.

۴- برای تعریف اعضای استاتیک، کلمه static را بعد از آخرین سطح دسترسی آن عضو بیان می کنیم.

۵- در متد های استاتیک، this وجود نداشته و قابل استفاده نیست چون شیء نمی تواند آن متد را فراخوانی کند.

۶- متد های استاتیک تنها به پارامتر های خود، متغیر های محلی، اعضای استاتیک دیگر کلاس و اعضای هر شیء دیگری که در خود ایجاد می کنند، دسترسی دارند و به اعضای غیر استاتیک کلاس دسترسی ندارند.

۷- متد های غیر استاتیک به تمام اعضای استاتیک و غیر استاتیک کلاس دسترسی دارند.

۸- فیلد های استاتیک بین همه اشیای کلاس مشترک هستند، به این معنی که هر شیء می تواند با یک متد غیر استاتیک که مقدار فیلد استاتیک را تغییر می دهند فیلد استاتیک را نیز تغییر داده و تغییر مذکور برای تمام اشیای کلاس قابل رویت خواهد بود. یعنی بعد از تغییر تمام اشیا مقدار جدید فیلد استاتیک را از طریق متد های خود در اختیار دارند.

۹- در صورتی که در اعلان متد از کلمه کلیدی static استفاده نشده باشد بدین معناست که از روی آن می توان نمونه ایجاد کرد و شیء تولید نمود. هر یک از اشیاء ایجاد شده از روی این متد ها، تمامی عناصر آن متد را دارا می باشند.

۱۰- خواص نیز مانند فیلد ها و متد ها می توانند به صورت استاتیک تعریف شوند، یعنی برای استفاده از آنها نیازی به ایجاد شیء نیست. برای مثال در کد زیر خصوصیت setRadius که به صورت استاتیک تعریف شده از فیلد استاتیک Radius استفاده می کند.

```
public class Circle
{
    private static float Radius = 1;
    public Circle ()
    {
        double area;
        area = Radius * Radius * 3.14;
    }
    public static float setRadius
    {
        get
        {
            return Radius;
        }
        set
        {
            Radius = value;
        }
    }
}
```

۱۱- از سازنده های استاتیک جهت دادن مقادیر اولیه به فیلد های اشیاء در زمان ایجاد آنها استفاده می شود. این سازنده ها دارای نحوه دسترسی نیستند و پارامتر ندارند. توجه کنید که این سازنده ها فقط یکبار اجرا می شوند و به تعداد اشیاء ساخته شده از کلاس ارتباطی ندارند. سازنده های استاتیک زودتر از بقیه سازنده ها اجرا می شوند.

```

public class Bus
{
    // Static constructor:
    static Bus()
    {
        System.Console.WriteLine("The static constructor invoked.");
    }
    public static void Drive()
    {
        System.Console.WriteLine("The Drive method invoked.");
    }
}
class TestBus
{
    static void Main()
    {
        Bus.Drive();
    }
}

```

۱۲- کلاس ها نیز همانند فیلد ها، متد ها، خواص و سازنده ها می توانند استاتیک باشند. هنگامی که کلاسی را توسط کلمه کلیدی static معرفی می کنید دیگر نمی توان یک شیء توسط کلمه کلیدی new از آن ایجاد کرد. این کلاس ها فقط می توانند شامل اعضای استاتیک باشند و اگر اعضای غیر استاتیکی در این گونه کلاس ها تعریف شوند خطای کامپایلری دریافت خواهید کرد. مزیت استفاده از کلاس های استاتیک که دارای اعضای فقط استاتیکی هستند این است که لزومی ندارد در مرحله اول مقدار دهی شوند و فضا دریافت کنند.

```

static class test
{
    public static void print ()
    {
        console.write ("Enter a number for print : ");
    }
}
static void main ()
{
    test t = newtest (); // Can not create an instance of the static class
}

```

مثال ۸ - ۵: برنامه زیر عملکرد دو متد استاتیک و غیر استاتیک را نشان می دهد.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و چهار متد اصلی خود را می نویسیم دو متد به صورت استاتیک و دو مورد آن غیر استاتیکی تعریف شده اند.

```
static void MethodA()
{
    Console.WriteLine("Static method");
}
void MethodB()
{
    Console.WriteLine("Instance method");
}
static char MethodC()
{
    Console.WriteLine("Static method");
    return 'C';
}
char MethodD()
{
    Console.WriteLine("Instance method");
    return 'D';
}
```

در static void main() دستورات زیر را می نویسیم.

```
static void Main(string[] args)
{
    // Call the two static methods on the Program type.
    Program.MethodA();
    Console.WriteLine(Program.MethodC());
    // Create a new Program instance and call the two instance methods.
    Program programInstance = new Program();
    programInstance.MethodB();
    Console.WriteLine(programInstance.MethodD());
}
```

خروجی برنامه به صورت زیر خواهد بود.

```
Static method
Static method
C
Instance method
Instance method
D
```

مثال ۹-۵: برنامه زیر نحوه ارسال پارامتر ها را در متد های استاتیک با کلمات کلیدی ref

و out بیان می کند.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و سپس متد های ۱ SetString و ۲ SetString را در داخل کلاس Program.cs تعریف می کنیم.

```
static void SetString1(ref string value)
{
    if (value == "cat") // Test parameter value
    {
        Console.WriteLine("Is cat");
    }
    value = "dog"; // Assign parameter to new value
}
static void SetString2(int number, out string value)
{
    if (number == 1) // Check int parameter
    {
        value = "one"; // Assign out parameter
    }
    else
    {
        value = "carrot"; // Assign out parameter
    }
}
```

توضیحات لازم به صورت Comment بیان شده است. از این متد ها در داخل متد اصلی برنامه استفاده می کنیم.

```
static void Main(string[] args)
{
    string value1 = "cat"; // Assign string value
    SetString1(ref value1); // Pass as reference parameter
    Console.WriteLine(value1); // Write result
    string value2; // Unassigned string
    SetString2(1, out value2); // Pass as out parameter
    Console.WriteLine(value2); // Write result
}
```

اکنون کافی است که برنامه را اجرا کرده و نتیجه را در خروجی مشاهده کنید.

مثال ۱۰-۵: برنامه ای بنویسید که با بهره گرفتن از یک متد استاتیک تشخیص دهد که آیا پارامتر ورودی یک عدد کامل هست یا خیر؟ مجموع اعداد کامل چهار رقمی را چاپ می کند.

حل : ابتدا یک پروژه از نوع Console ساخته و سپس متد IsComplete را به صورت زیر تعریف کنید.

```
static bool IsComplete(uint n)
{
    uint sum = 0;
    for (uint i = 1; i < n / 2; i++)
        if (n % i == 0)
            sum += i;
    if (sum == n)
        return (true);
    else
        return (false);
}
```

متد IsComplete با دریافت عدد صحیح و مثبت و بررسی کامل بودن آن مقدار true یا false را بر می گرداند. در متد اصلی برنامه از متد IsComplete به صورت زیر استفاده می کنیم.

```
static void Main(string[] args)
{
    ulong sum = 0;
    for (uint i = 1000; i < 9999; i++)
        if (IsComplete(i))
            sum += i;
    Console.WriteLine("\n sum =" , sum);
}
```

متد های دارای پارامتر های متغیر

یکی از ویژگی های مفید زبان C#، وجود پارامتر های params است که بوسیله آنها می توان متدی را اعلان کرد که تعدادی متغیر را به عنوان پارامتر دریافت نماید. پارامترهای params حتماً باید یکی از انواع آرایه تک بعدی باشند. به این ترتیب زمان ارسال پارامتر ها به متد، می توان هر تعداد پارامتر را ارسال کرد و این پارامتر ها در عناصر آرایه یک بعدی قرار می گیرند و در متد فراخوانی شده دستیابی می شوند. اگر متد غیر از پارامتری که با واژه params مشخص می شود پارامتر دیگری نیز داشته باشد این پارامتر ها باید قبل از پارامتر params تعریف شوند. یعنی پارامتر params بایستی آخرین پارامتر باشد. به عنوان مثال دستورات زیر را در نظر بگیرید :

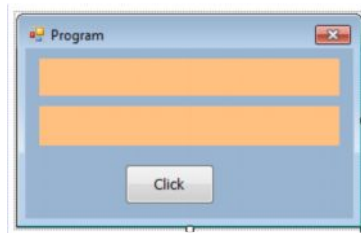
```
public void sum (params int [] array1 , int len)
{
    ...
}
```

چون در تعریف پارامترهای متد sum، ابتدا پارامتر params و سپس پارامتر len از نوع int تعریف شده است با خطا مواجه خواهید شد. شکل صحیح تعریف آن به صورت زیر است:

```
public void sum (int len , params int [] array1)
{
...
}
```

مثال ۱۱ - ۵: برنامه زیر نحوه استفاده از متدهایی با تعدادی متغیر از پارامترها را نشان می دهد. این برنامه مجموع پارامترهای ارسال شده به متد را محاسبه می کند. در این برنامه، یک بار متد را با سه پارامتر و بار دیگر با پنج پارامتر فراخوانی می کنیم.

حل: ابتدا یک پروژه جدید از نوع ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۳۰ - ۵ طراحی می کنیم.



تصویر ۳۰ - ۵

ابتدا متد sum را در بخش public partial class تعریف می کنیم. و کلاس جدیدی را تعریف نمی کنیم چون هنگام ساخت پروژه کلاسی با نام پروژه ایجاد می شود.

```
public int sum(int len, params int[] array1)
{
    int s = 0;
    for (int i = 0; i < len; i++)
        s += array1[i];
    return s;
}
```

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "نتیجه فراخوانی با سه پارامتر" + sum(3, 10, 20, 30);
    label2.Text = "نتیجه فراخوانی با پنج پارامتر" + sum(5, 10, 20, 30, 40, 50);
}
```

مثال ۱۲ - ۵: برنامه زیر نیز همانند مثال بالا نحوه استفاده از متد هایی با تعدادی متغیر از پارامترها را نشان می دهد. این برنامه مجموع پارامتر های ارسال شده به متد را محاسبه کنیم. توضیحات لازم در بخش کد های برنامه به صورت Comment آورده شده است.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده، سپس یک متد به نام SumParameters تعریف کنید.

```
static int SumParameters(params int[] values)
{
    // Loop through and sum the integers in the array.
    int total = 0;
    foreach (int value in values)
    {
        total += value;
    }
    return total;
}
```

در متد اصلی برنامه دستورات زیر را می نویسیم.

```
static void Main(string[] args)
{
    // Call params method with one to four integer constant parameters.
    int sum1 = SumParameters(1);
    int sum2 = SumParameters(1, 2);
    int sum3 = SumParameters(3, 3, 3);
    int sum4 = SumParameters(2, 2, 2, 2);
    // Write results of the method invocations.
    Console.WriteLine(sum1);
    Console.WriteLine(sum2);
    Console.WriteLine(sum3);
    Console.WriteLine(sum4);
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

نکته!

در زبان C# می توان دو یا چند متد را در داخل یک کلاس نوشت که همانم باشند ولی پارامتر این متد ها باید با هم فرق کنند (تعداد پارامتر ها، انواع پارامتر ها، یا ترتیب پارامتر ها). این عمل را در C# تعریف مجدد متد می نامند. وقتی یکی از متد های همانم فراخوانی می شود، کامپایلر C#، با بررسی تعداد، انواع، و ترتیب پارامتر های متد فراخوانی شده، متد مناسبی را انتخاب می نماید. همانمی متد ها معمولاً برای ایجاد چندین متد همانم که کار های مشابهی را انجام می دهند، به کار می رود. به عنوان مثال، متد های زیر را در نظر بگیرید :

```
public void myFunction (int a , int b);  
public void myFunction (float a , float b);
```

متد های بازگشتی

بعضی از مسائل طبیعت بازگشتی دارند. مثلاً اگر به ما بگویند ۵ برابر چند است با توجه به فرمول $n = n * (n - 1)$ می توانیم بگوئیم که اگر ۴ را بدانیم کافی است که آن را در ۵ ضرب کنیم. پس مساله ۵! تبدیل به مساله ۴! می شود و الی آخر.

متد هایی که تا به حال نوشتیم، توانستند متد های دیگر را فراخوانی کنند. به دلیل وجود بعضی از مسائل، مفید است که متدی خودش را فراخوانی کند. متد بازگشتی، متدی است که درون بدنه خود، خود را صدا می زند. متد های بازگشتی دارای دو ویژگی اصلی هستند :

۱- متد خودش، خودش را صدا می زند (معمولاً با آرگومان کمتر).

۲- یک شرط جهت اتمام فراخوانی ها وجود دارد.

عمل بازگشت در علم کامپیوتر یک روش فکر کردن برای حل مسائل است. در واقع بازگشت یکی از ایده های اصلی علم کامپیوتر است. حل یک مسئله به روش بازگشتی بدین معناست که راه حل بستگی به مدل کوچکتری از صورت مسئله داشته باشد.

مرحله بازگشتی می تواند چندین مرحله بازگشتی دیگر را ایجاد کند و این متد، مسئله را به دو مساله کوچکتر تقسیم می کند. هر بار متد خودش را با نسخه ساده تری از مسئله اصلی فراخوانی می کند، و دنباله ی مسئله های کوچکتر باید به حالت پایه ختم شوند تا اینکه بازگشتی به اتمام برسد.

الگوریتم محاسبه فاکتوریل و فیبوناچی مرسوم ترین مثال های بازگشتی می باشد. متدهای بازگشتی در زبان C# به دو صورت با پارامتر ورودی و بدون پارامتر ورودی تقسیم می شود که به صورت زیر تعریف می شوند :

() نام متد نوع پارامتر بازگشتی سطح دسترسی

{

Codes

. . .

return نام متغیر یا پارامتر بازگشتی

}

بدون پارامتر ورودی

(نام و نوع پارامتر ها) نام متد نوع پارامتر بازگشتی سطح دسترسی

{

Codes

. . .

return نام متغیر یا پارامتر بازگشتی

}

با پارامتر ورودی

نکته!

مرحله بازگشتی معمولاً حاوی دستور return است.

متدهای بازگشتی معمولاً به صورت () (پارامتر های ورودی) نام متد و یا () نام متد (فراخوانی می شود).

مثال ۱۳ - ۵: برنامه ای بنویسید که یک عدد را از کاربر دریافت کرده و فاکتوریل آن را به

صورت بازگشتی محاسبه کند؟

الگوریتم بازگشتی فاکتوریل به صورت زیر می باشد.

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n - 1) & \text{if } n > 0 \end{cases}$$

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و یک متد بازگشتی به صورت زیر تعریف

می کنیم :

```
static long Factorial(long number)
{
    long f;
    if (number == 1)
        return 1;
    else
        f = Factorial(number - 1) * number;
    return f;
}
```

در متد اصلی برنامه از متد Factorial به صورت زیر استفاده می کنیم.

```
static void Main(string[] args)
{
    Console.WriteLine("Please enter a number whose factorial is to be found: ");
    int number = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("The factorial of the number is : {0}", Factorial(number));
}
```

مثال ۱۴ - ۴ : برنامه ای بنویسید که ۱۰ جمله از سری فیبوناچی را به روش بازگشتی حل کند؟

الگوریتم بازگشتی سری فیبوناچی به صورت زیر است :

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{if } n \geq 2 \end{cases}$$

حل : ابتدا یک پروژه از نوع console ایجاد کرده و متد fibonacci را به صورت زیر در آن تعریف کنید.

```
static int fibonacci(int x)
{
    if (x <= 1)
    {
        return 1;
    }
    return fibonacci(x - 1) + fibonacci(x - 2);
}
```

از متد fibonacci در متد اصلی برنامه به صورت زیر استفاده کنید.

```
static void Main(string[] args)
{
    for (int i = 1; i <= 10; i++)
        Console.WriteLine("Fibonacci no. = {0}", fibonacci(i));
}
```

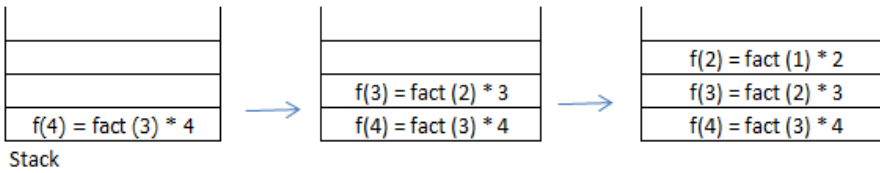
برنامه را اجرا کرده و نتیجه را مشاهده کنید.

نکته!

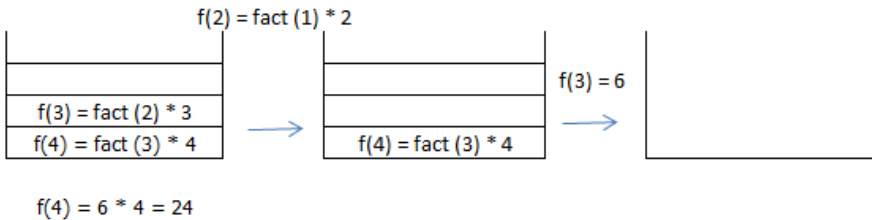
معمولا مسائلی را با تکنیک بازگشتی حل می کنیم که حالت n ام آن به کمک حالت $n - 1$ آن قابل بدست آوردن باشد. بسیاری از مسائلی را که الگوریتم حل آنها به ظاهر پیچیده است به کمک تکنیک بازگشتی به راحتی می توان حل کرد. مزیت روش بازگشتی به روش معمولی، سادگی برنامه نویسی آن است و عیب آن مصرف زیاد حافظه (به خاطر مصرف حافظه Stack و فراخوانی های مکرر) و نیز زمان زیادی است که برای فراخوانی ها صرف می شود. هر مسئله ای که بتواند با روش بازگشتی حل شود، به روش تکراری نیز حل خواهد شد. اما وقتی حل مساله به صورت بازگشتی، طبیعی به نظر برسد و با ماهیت مساله مطابقت داشته باشد، خوانایی برنامه بالاتر خواهد بود.

به طور مثال مساله فاکتوریل را در مثال قبلی در نظر بگیرید، تمام متغیر های محلی و پارامتر های آن دوباره فضائی را از پشته می گیرند و مقدار جدید پارامتر ها در این فضا ذخیره می شوند و متد با این مقادیر جدید اجرا می شود. با فراخوانی های مکرر متد، هیچ فضائی برای ذخیره سازی مجدد کد متد اشغال نمی شود. برای سادگی فرض می کنیم عملیاتی که قرار است هم اکنون اجرا شوند ولی به علت فراخوانی مجدد متد نمی توان آنها را اجرا کرد، در داخل حافظه پشته ذخیره می گردند تا بعدا اجرا شوند. حافظه پشته به صورت (LIFO) می باشد (Last In First Out) یعنی آخرین اطلاعات وارد شده ابتدا خارج می شود مانند برداشتن بشقاب هایی که روی هم چیده می شوند. Stack نقش دفترچه یادداشت یا چرکنویس را برای برنامه ها دارد. هنگامی که واسط اجرای یک متد به ابتدای متد می رود (با فراخوانی مجدد)، دستوراتی که اجرا نشده اند (دستورات بعد از صدا زدن) در Stack ذخیره می شوند تا بعدا اجرا گردند.

اگر در برنامه فاکتوریل مثال قبل مقدار number را برابر ۴ دهید پشته به صورت زیر ابتدا تا هنگامی که شرط برقرار نباشد پر می شود. سپس هنگام رسیدن به شرط if و درست بودن آن، Stack از بالا به پائین خالی می شود.



در حالتی که $n = 1$ باشد شرط if درست شده و تابع $\text{fact}(1)$ برابر ۱ می گردد و در این حال Stack از بالا به پایین خالی می شود.



پس خروجی نهایی تابع عدد ۲۴ می شود (یعنی ۴!).

متغیرهای محلی و سراسری

هر متغیر دارای خصوصیتی مانند نام، نوع، اندازه و مقدار است. یک متغیر علاوه بر خصوصیات ذکر شده خصوصیات دیگری مانند طول عمر (Duration or Life time) و حوزه (Scope) دارند. طول عمر متغیر مدت زمانی است که در حافظه وجود دارد. بعضی از متغیرها طول عمر کوتاهی دارند، بعضی از آنها دائماً حذف و ایجاد می شوند، و بعضی دیگر تا انتهای برنامه وجود دارند. حوزه متغیر، جایی است که به نام متغیر می تواند در برنامه مراجعه شود. بعضی از متغیرها در سراسر برنامه قابل دستیابی هستند در حالی که به بعضی دیگر فقط در بخشی از برنامه می توان مراجعه کرد.

متغیرهایی که در داخل یک متد تعریف می شوند متغیرهای محلی (Local) بوده و فقط در همان تابع قابل استفاده هستند و بیرون تابع شناخته شده نمی باشند. متغیرهای سراسری (Global) یا عمومی خارج از بدنه تمامی متد ها تعریف می شوند و هر متغیر سراسری از محل تعریف آن به بعد در دسترس است. در واقع متغیر سراسری مانند منبع آبی است که در بالای ساختمانی چند طبقه قرار دارد و آب مورد نیاز تمام طبقات را تامین می کند.

متغیرهای محلی در زبان C# می تواند در ابتدای هر بلوکی (یعنی بلافاصله بعد از {) تعریف کرد. این آکولاد باز می تواند مربوط به if, while, for, Class, Method و یا هر بلوک دیگری باشد. متغیر محلی در همان بلوکی که در آن تعریف شده، شناخته شده می باشد. یعنی در زمان ورود به بلوک، آن متغیر در Stack ساخته شده و هنگام خروج از بلوک متغیر از بین رفته و فضای آن به پشته برگردانده می شود.

متغیرهای محلی در Stack ساخته می شوند و اگر مقدار دهی نشوند مقدار اولیه آنها نامشخص است. متغیرهای سراسری در Data Segment ساخته شده و اگر مقدار دهی نشوند مقدار اولیه آنها صفر است.

متغیرهای سراسری در ابتدای برنامه ساخته شده و تا انتهای برنامه فضای آن از بین نمی رود. اگر در متدی متغیری محلی همنام با یک متغیر سراسری تعریف شود، هنگامی که در آن تابع نام متغیر مذکور استفاده شود منظور متغیر محلی است نه سراسری. نوع دیگری از متغیرهای محلی، متغیرهای محلی static می باشند که دارای ویژگی های زیر می باشند :

- ۱- فقط در همان متدی که تعریف شده اند، شناخته شده می باشند و قابل استفاده هستند.
 - ۲- داخل Data Segment ساخته شده و فقط یکبار مقدار دهی اولیه می شوند و هنگام خروج از متد، آخرین مقدار خودشان را حفظ می کنند.
 - ۳- اگر مقدار دهی اولیه نشوند مقدار اولیه آنها صفر است.
- این متغیرها از آن نظر که فقط در درون متد شناخته شده می باشند مشابه متغیرهای محلی هستند ولی از آنجا که محتوای خود را حفظ می کنند مشابه متغیرهای سراسری می باشند. در زیر یک نمونه از متغیر محلی static را می توانید مشاهده کنید.

```
static int i = 1;
```

مثال : نمونه ای از تعریف متغیر های محلی را در زیر می بینید.

```
private void chap()  
{  
    string name,lastname,fullname;  
    name = "Reza";  
    lastname = "bahramirad";  
    fullname = name + lastname;  
    textBox1.Text = fullname;  
}  
  
private void chap1()  
{  
    string name,lastname,fullname; //error  
    name = "Reza"; //error  
    lastname = "bahramirad"; //error  
    fullname = name + lastname; //error  
    textBox1.Text = fullname; //error  
}
```

توجه کنید که متغیر های name, lastname, fullname به صورت محلی در متد chap() تعریف شده اند، لذا هنگام دسترسی به آنها در تابع chap۱() با پیغام خطا روبرو خواهید شد.

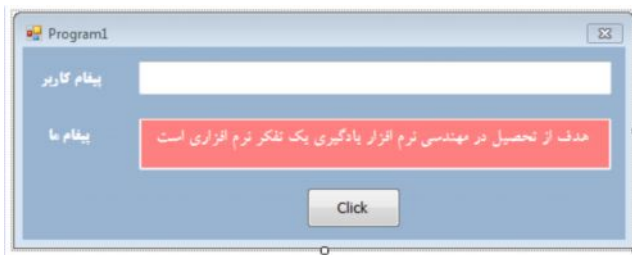
مثال : نمونه ای از متغیر های سراسری را در زیر می بینید.

```
string name, lastname, fullname;  
private void chap()  
{  
    string name,lastname,fullname;  
    name = "Reza";  
    lastname = "bahramirad";  
    fullname = name + lastname;  
    textBox1.Text = fullname;  
}  
  
private void chap1()  
{  
    string name,lastname,fullname;  
    name = "Reza";  
    lastname = "bahramirad";  
    fullname = name + lastname;  
    textBox1.Text = fullname;  
}
```

توجه کنید که متغیر های name , lastname , fullname به صورت سراسری تعریف شده اند، لذا توابع chap() و chap۱() بدون هیچ مشکلی می توانند به این متغیر ها دسترسی داشته باشند.

مثال ۱۵ - ۴: برنامه ای بنویسید که مقادیر کنترل های textBox۱ و textBox۲ موجود در Form۱ را برای Form۲ ارسال کند؟

حل : ابتدا یک پروژه جدید از نوع ویندوزی ایجاد کرده علاوه بر Form۱ که به صورت پیش فرض ساخته می شود، Form۲ را هم مطابق روش های گفته شده در فصل های گذشته به پروژه خود اضافه کنید. Form۱ را مانند پنجره تصویر ۳۱ - ۵ و Form۲ را مطابق پنجره تصویر ۳۲ - ۵ طراحی می کنیم.



تصویر ۳۱ - ۵



تصویر ۳۲ - ۵

حال یک کلاس داخل جدید با عنوان MyClass ایجاد کرده و دو متد a و b را به صورت زیر در آن تعریف کنید.

```
class MyClass
{
    public static string a;
    public static string b;
}
```

اکنون به Form۱ برگشته و بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس این دستورات را وارد کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    MyClass.a = textBox1.Text;
    MyClass.b = textBox2.Text;
    Form a = new Form2();
    a.Show();
}
```

در پنجره Solution Explorer بر روی Form2 دابل کلیک کنید تا باز شود. در رویداد Form2_Load دستورات زیر را بنویسید.

```
private void Form2_Load(object sender, EventArgs e)
{
    label1.Text = MyClass.a;
    label2.Text = MyClass.b;
}
```

برنامه را اجرا کرده و نتیجه را مشاهده کنید. در واقع یکی از مسائل بسیار مهم در برنامه های چند Form ای، ارسال مقادیر کنترل ها برای Form های دیگر می باشد. برای این منظور می توان از فیلدها و متغیرهای static در کلاس استفاده کرد. به نحوی که قبل از نمایش Form جدید، مقداری که باید به Form جدید ارسال شود در فیلد static کلاس قرار داده شود، سپس از مقدار آن فیلد در Form جدید استفاده گردد.

مفهوم سازنده (Constructor)

سازنده ها، متدهایی هستند که وظیفه ایجاد شیء از کلاس را بر عهده دارند. در صورتی که شما هیچ سازنده ای تعریف نکرده باشید، هر کلاس به طور خودکار دارای سازنده پیش فرض (سازنده ای که پارامتر نداشته باشد) می باشد. سازنده متدی است همانم با کلاس که چه در صورت نوشتن و چه در صورت ننوشتن بر روی کلاس ها وجود دارد.

سازنده پیش فرض (Default constructor)

سازنده پیش فرض، سازنده ای است که پارامتر نداشته باشد. با این تعریف به سازنده ای که طراح کلاس (برنامه نویس) بنویسد و پارامتر نداشته باشد نیز سازنده پیش فرض، گفته می شود. هر کلاس می تواند چند سازنده با پارامترهایی متفاوت داشته باشد که به آن، سربارگذاری سازنده ها گویند.

سازنده ها، متد هایی از کلاس هستند که برای آنها خروجی مشخص نمی شود (زیرا خروجی یک سازنده همیشه مشخص و ثابت بوده و به صورت یک شیء از کلاس مربوطه است، ذکر خروجی برای آن لازم نیست و گفته می شود که خروجی ندارد).

سازنده ها همانند متد ها دارای سطوح دسترسی هستند. آنها معمولاً برای مقدار دهی اولیه اشیاء در زمان ایجاد (با استفاده از عملگر new) استفاده می شوند. زمان ایجاد شیء از کلاس، دیدید که به صورت زیر عمل می شود :

```
className objName = new className();
```

className نام کلاس و objName نام شیء ای است که باید از این کلاس ایجاد شود. new واژه کلیدی است که به همین صورت استفاده می شود و className نام سازنده کلاس است. اگر سازنده ای دارای پارامتر باشد، در هنگام ایجاد شیء ای از آن کلاس، باید پارامتر ها را نیز ارسال کنید :

```
className objName = new className (parameters);
```

مقدار اولیه می تواند به صورت ایستا و ثابت یا به صورت پویا و پارامتری نیز انجام شود. برای مقدار دهی به صورت ایستا، روش جایگزین مقدار دهی اولیه فیلد ها در هنگام تعریف فیلد در کلاس (initializer) نیز وجود دارد ولی برای مقدار دهی اولیه به صورت پویا که در واقع مقدار دهی اولیه بر حسب پارامتر دریافتی (ارسالی به سازنده) انجام می دهد، سازنده ها تنها گزینه خواهند بود.

برای مثال کلاس HijriClass را در نظر بگیرید. فرض این است که شیء این کلاس مقادیر سال، ماه، و روز یک تاریخ هجری شمسی را در خود نگه خواهد داشت، ضمن اینکه متد های مربوط به کار با تاریخ شمسی را نیز دارد. با این توضیح سه فیلد Year , Month , Day از نوع int و با سطح دسترسی private در کلاس تعریف شده اند. قصد داریم دو سازنده برای این کلاس تعریف کنیم. یک سازنده ایستا و یک سازنده پویا. سازنده ایستا می بایست زمانی که شیء با استفاده از آن ایجاد می شود مقادیر سال، ماه و روز را در فیلد های شیء با ۱۳۹۱ و ۷ و ۱۶ مقدار دهی اولیه کند و سازنده پویا باید بر حسب سه پارامتر دریافتی، فیلد های

مذکور را مقدار دهی نماید. کد زیر کلاس و فیلدها و سازنده های آن را مطابق با توضیحی که داده شده است نشان می دهد :

```
public class HijriClass
{
    private int Year;
    private int Month;
    private int Day;
    public HijriClass()
    {
        Year = 1391;
        Month = 7;
        Day = 16;
    }
    public HijriClass (int pYear, int pMonth, int pDay)
    {
        Year = pYear;
        Month = pMonth;
        Day = pDay;
    }
}
```

با استفاده از این سازنده ها قصد داریم دو شیء ایجاد کنیم. این کار را می توانیم در یک پروژه ویندوزی و در رویداد Click یک کنترل button انجام دهیم. کد مربوط به ایجاد دو شیء از طریق سازنده ایستا و سازنده پویا در زیر آمده است :

```
HijriClass HijriObj1 = new HijriClass();
HijriClass HijriObj2 = new HijriClass (1391,3,4);
```

پس از اجرای دستور اول شیء HijriObj1 حاوی شیء ای از کلاس HijriClass است که مقدار فیلدهای سال، ماه، و روز آن به ترتیب برابر ۱۳۹۱، ۷ و ۱۶ می باشد که این مقدار دهی توسط سازنده ایستا که برای ایجاد آن استفاده شده، انجام گرفته است. از طرفی با اجرای دستور دوم شیء HijriObj2 در فیلدهای سال، ماه و روز خود به ترتیب دارای مقادیر ۱۳۹۱، ۳ و ۴ را خواهد داشت که این مقادیر را از سازنده پویا در زمان ایجاد دریافت کرده است و امکان ارسال هر مقدار صحیح ثابت یا متغیر دیگری نیز وجود دارد.

هنگامی که بعد از کلمه new اسم کلاس HijriClass (نام سازنده) را وارد کرده و پارانتر را باز می کنید، ویرایشگر متن یک پنجره کوچک باز می کند که نشان می دهد دو متد با این نام

در کلاس وجود دارند و این امکان برای شما وجود دارد که یکی از آنها را انتخاب نمایید. آنگاه بر حسب انتخاب شما، پارامتر های آن متد نشان داده می شود.

```
public partial class Form1 : Form
{
    HijriClass HijriObj1 = new HijriClass();

    HijriClass HijriObj2 = new HijriClass(1391,3,4);
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        HijriClass HijriObj1 = new HijriClass(|
        1 of 2 HijriClass.HijriClass 0
    }
}
```

سازنده تنها یک بار در زمان حیات شیء و آن هم در زمان ایجاد شیء (new شدن شیء از کلاس)، اجرا می شود. بعد از آن کلیه عملیات روی شیء (شامل تغییر مقادیر فیلدها و...) با سایر متدها (یا خاصیت ها و یا فیلدها) انجام می شود.

اگر هیچ سازنده ای تعریف نشود، برای ایجاد شیء باید از سازنده ای که به طور خودکار و تلویحی برای کلاس هایی که برای آنها سازنده نوشته نشده اند در نظر گرفته می شود، استفاده کرد که همان سازنده بدون پارامتر است. در این هنگام مقدار دهی اولیه فیلدها به طور پیش فرض، بر حسب نوع فیلد (عددی، رشته ای، کلاس و...)، مقدار پیش فرض آن نوع (صفر، خالی، null و...) می شود.

برای مثال اگر برای کلاس HijriClass هیچ سازنده ای تعریف نکنید برای ایجاد شیء باید از دستور زیر استفاده کنید که بعد از اجرای آن مقادیر فیلدهای سال، ماه و روز شیء برابر صفر خواهد بود.

```
HijriClass HijriObj1 = new HijriClass();
```

نکته!

اگر کلاس دارای سازنده هایی باشد، اما هیچ کدام از سازنده های public آن، سازنده پیش فرض نباشند، و برنامه نویس سعی کند سازنده ای بدون پارامتر را برای مقدار اولیه دادن به

اشیاء کلاس فراخوانی کند، خطایی توسط کامپایلر صادر می شود. سازنده فقط وقتی می تواند بدون پارامتر فراخوانی شود که هیچ سازنده ای برای کلاس تعریف شده نباشد (که در این صورت کامپایلر سازنده ای را تعریف می کند)، یا سازنده ای بدون پارامتر با سطح دستیابی public تعریف شده باشد.

نکته!

متد های سازنده هیچ مقداری را بر نمی گردانند (حتی از نوع void). متد های سازنده می توانند هیچ، یک و یا چندین پارامتر ورودی داشته باشند.
در مثال زیر کلاسی با متد سازنده اش تعریف شده است.

```
public class myClass
{
private int a;
public myClass(); //Constrauctor
public void show();
}
```

مقدار دهی اولیه فیلدها هنگام تعریف در کلاس (Initializer)

این روش جایگزینی برای سازنده ها در حالت ایستا است که در این حالت فیلدها در زمان تعریف در کلاس، مقدار دهی اولیه می شوند. در این روش می توان به جای تعریف سازنده ایستا برای مقدار دهی اولیه فیلدها، سازنده تعریف نکرد و به جای آن در تعریف فیلدها در کلاس برای آنها مقدار اولیه تخصیص داد.

در بخش قبل به جای تعریف سازنده ایستا می توانستیم مقدار دهی اولیه را به صورت زیر در زمان تعریف فیلدها در کلاس انجام دهیم :

```
public HijriClass()
{
Year = 1391;
Month = 7;
Day = 16;
}
```

در این صورت بعد از اجرای دستور `HijriClass HijriObj1 = new HijriClass();` شیء با استفاده از سازنده پیش فرض ایجاد می شود ولی این بار فیلدهای سال، ماه و روز به جای مقادیر ۱۳۹۱، ۷ و ۱۶ را خواهند داشت.

نکته ای که اینجا تاکید داریم این است که سازنده پیش فرض، تنها هنگامی برای کلاس به صورت تلویحی در نظر گرفته می شود که ما هیچ سازنده ای برای کلاس ننوشته باشیم. در مثال قبلی، فرض کنید که شما، تنها سازنده پارامتری را تعریف کرده باشید و سازنده دیگری در کلاس تعریف نشده باشد. در این حالت با توجه به این که شما یک سازنده تعریف کرده اید، دیگر سازنده بدون پارامتر پیش فرض برای کلاس در نظر گرفته نمی شود و در نتیجه شما مجاز به استفاده از دستور `HijriClass HijriObj1 = new HijriClass();` نیستید و تنها می بایست از سازنده پارامتری برای ایجاد شیء خود استفاده کنید که در این حالت مقادیر فیلد ها توسط آن مقدار دهی می شوند و مقادیر تنظیم شده توسط `initializer` رونویسی خواهند شد.

سازنده از روی شیء کلاس خود (Copy Construcor)

این سازنده، سازنده ای است که پارامتری از نوع کلاس خود می گیرد و شیء جدیدی با مقادیر فیلد های پارامتر گرفته شده ایجاد می کند. در واقع این سازنده شیء ای از کلاس خود را می گیرد و شیء جدید با مقادیر فیلد های برابر آن شیء ایجاد می کند، گویا یک کپی از شیء دیگر ایجاد شده است. با این توضیحات چنین سازنده ای برای مثال کلاس `HijriClass` به شکل زیر خواهد بود :

```
public class myClass
{
private int a;
public myClass(); //Constractor
public void show();
}
```

در کد بالا سازنده، شیء ای از کلاس `HijriClass` را به عنوان پارامتر دریافت می کند و مقادیر فیلد های آن را در فیلد های شیء جدیدی که در حال ایجاد است کپی می کند. دستورات زیر از این سازنده استفاده کرده اند :

```
HijriClass HijriObj2 = new HijriClass (1391,3,4);
.
HijriClass HijriObj3 = new HijriClass (HijriObj2);
```

در اینجا دستور اول شیء `HijriObj2` با سازنده پارامتری ایجاد کرده است و مقادیر فیلد های آن بر حسب مقادیر ورودی برابر ۱۳۹۱، ۳ و ۴ شده اند. در زمان دیگر، دستور دوم اجرا شده

است با اجرای این دستور شیء جدید HijriObj۳ با مقادیر کنونی فیلد های شیء HijriObj۲ ایجاد می شود. در واقع گویا یک شیء کپی و مشابه شیء (تشابه مقادیر فیلد ها) HijriObj۲ ایجاد کرده ایم.

پس یک متد در کلاس HijriClass با نام SetDate تعریف می کنیم که سه پارامتر از نوع صحیح می گیرد و فیلد های سال، ماه و روز را مقدار دهی می کند. از این متد می توان برای تغییر مقادیر فیلد های Year, Month, Day بعد از ایجاد شیء و در هر زمان که مورد نیاز بود استفاده کرد.

```
public void SetDate (int pYear, int pMonth, int pDay)
{
    Year = pYear;
    Month = pMonth;
    Day = pDay;
}
```

مثال ۱۶ - ۵: برنامه ای بنویسید که پیغام (I am a programmer) را با استفاده از سازنده ها چاپ کند؟

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و در داخل کلاس program سازنده زیر را تعریف کنید.

```
class Program
{
    public Program() //create a constructor with the same name of the class.
    {
        Console.WriteLine("I am a programmer");
    }
}
```

در متد اصلی برنامه این سازنده را فراخوانی کنید.

```
static void Main(string[] args)
{
    Program p = new Program();//create a object when the constructor is called
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

سازنده های ایستا

یکی از ویژگی های جدید زبان C# این است که می توان در آن سازنده های ایستای بدون پارامتر را ایجاد کرد. همانطور که در مورد سازنده هایی که تاکنون نوشته شده اند مطرح شد،

این سازنده ها سازنده های نمونه هستند و زمانی که یک شیء از آن کلاس ایجاد می شود اجرا می شوند. سازنده های ایستا نیز تنها یک بار اجرا می شوند. در زبان های C++ و VB6 معادلی برای سازنده ایستا وجود ندارد.

```
class MyClass
{
static MyClass ()
{
// initialization code
}
// reset of class definition
}
```

در زیر نکاتی در مورد سازنده های ایستا و کاربردهای آن بیان شده است :

۱- کد تعریف شده در سازنده ایستا، بعد از اجرای برنامه و قبل از ایجاد اولین شیء اجرا می شود و زمان دقیق اجرای آن مشخص نیست.

۲- قبل از سازنده ایستا، نمی توان سطح دسترسی قرار داد.

۳- همانند یک متد ایستا، سازنده ایستا نیز به اعضای غیر استاتیک دسترسی دارد.

یک دلیل برای نوشتن یک سازنده ایستا، می تواند موقعیتی باشد که در آن، کلاس شما دارای بعضی فیلدهای ایستا یا ویژگی هایی است که باید از یک منبع خارجی قبل از اولین استفاده از کلاس، مقدار دهی شوند.

زمان اجرا NET. هیچ تضمینی را درباره زمانی که یک سازنده ایستا اجرا می شود، نمی دهد، بنابراین شما نباید هیچ کدی را در آن قرار دهید به این امید که در زمان خاصی اجرا خواهد شد (به طور مثال، زمانی که یک اسمبلی بارگذاری می شود). همچنین نباید ترتیبی از اجرای سازنده های ایستای کلاس های مختلف را تصور کنید. تنها تضمینی که وجود دارد این است که سازنده ایستا تنها یک بار اجرا خواهد شد و نیز قبل از اینکه کد شما هر گونه ارجاعی را ایجاد کند، فراخوانی خواهد شد. در زبان C# به نظر می رسد که سازنده ایستا معمولاً بلافاصله قبل از اولین فراخوانی به هر عضوی از کلاس اجرا خواهد شد.

توجه کنید که سازنده ایستا به هیچ تغییر دهنده ای دسترسی ندارد. این سازنده هرگز توسط کد دیگری از زبان C# فراخوانی نمی شود بلکه تنها همواره به وسیله زمان اجرای NET.

هنگامی که کلاس بار شده است فراخوانی می شود بنابراین هر اصلاح کننده دسترسی مانند public یا private بی معنی خواهد بود.

به همین دلیل، سازنده ایستا حتی نمی تواند هیچ پارامتری را بگیرد و یک کلاس تنها می تواند یک سازنده ایستا داشته باشد. بنابراین واضح است که سازنده های ایستا تنها می توانند به اعضای استاتیک کلاس دسترسی داشته باشند و دسترسی به اعضای اشیاء نمونه سازی شده از کلاس برای آنها امکان پذیر نیست.

در صورتی که یک سازنده ایستا و یک سازنده بدون پارامتر در یک کلاس تعریف شوند. با اینکه تعداد پارامتر های هر دو یکسان است، ولی تداخلی بوجود نخواهد آمد زیرا سازنده ایستا زمانی اجرا شده است که کلاس بار شده است اما سازنده معمولی زمانی اجرا شده است که یک نمونه ایجاد شده است. بنابراین هیچ ابهامی درباره اینکه چه سازنده ای در چه زمانی اجرا شده است وجود نخواهد داشت.

توجه داشته باشید که اگر شما پیش از یک کلاس دارید که در آن سازنده ایستا تعریف کرده اید سازنده ایستایی که در ابتدا اجرا خواهد شد در واقع تعریف نشده است. این بدان معناست که شما نباید هیچ کدی را داخل یک سازنده ایستا قرار دهید که به اجرا یا عدم اجرای سازنده های ایستای دیگر بستگی دارد. از طرف دیگر اگر هر فیلد ایستایی با مقادیر پیش فرض مقدار دهی شده باشد این مقادیر قبل از فراخوانی سازنده ایستا اعمال خواهند شد.

در زیر سناریویی از یک مثال کاربردی و عملی یک سازنده ایستا را بررسی می کنیم :

فرض کنید یک پروژه از dll های نوشته شده زیادی توسط شرکت یا شخص ثالث استفاده می کند. در زمانی که حجم dll ها زیاد بوده و به کارگیری (Load) یکباره آنها، فضای زیادی از حافظه بگیرد بهتر است که هر dll زمانی بارگذاری شود که به آن نیاز داریم. فرض کنید در یک یا چند متد از متد های ClassA یکی از dll ها استفاده می شود. این طور به نظر می رسد که روش منطقی این است که باید قبل از ایجاد اولین شیء از کلاس ClassA این dll در حافظه بارگذاری شود. با تعریف یک سازنده ایستا در کلاس ClassA و نوشتن دستور بارگذاری آن، dll مربوطه NET Framework. این امکان و تضمین را به شما می دهد که قبل

از اجرای اولین دستور `new ClassA()` که منجر به پیدایش اولین شیء از کلاس مذکور می شود سازنده ایستا اجرا شود که منجر به بارگذاری dll مربوطه در حافظه می شود.

مثال ۱۷-۵: برنامه زیر نحوه استفاده از متد های سازنده را نشان می دهد.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و یک کلاس داخلی با عنوان Constructor ایجاد کرده و دستورات زیر را در آن بنویسید.

```
class Constructor
{
    private static int id ;
    //Static constructor, value of data member id is set conditionally here.
    //This type of initialization is not possible at the time of declaration.
    static Constructor()
    {
        if (Constructor.id < 10)
        {
            id = 20;
        }
        else
        {
            id = 100;
        }
        Console.WriteLine("Static<Class> Constructor for Class Constructor Called..");
    }
    public static void print()
    {
        Console.WriteLine("Constructor.id = " + id);
    }
}
```

حال در متد اصلی کلاس `program.cs` دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    //Print the value of id
    Constructor.print();
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال ۱۸-۵: برنامه زیر به صورت خیلی ساده تفاوت سازنده استاتیک و غیر استاتیک را نشان می دهد.

حل : ابتدا یک پروژه از نوع Console ساخته و سپس دو کلاس داخلی به نام های `HasStaticConstructor` و `NoStaticConstructor` ساخته و دستورات زیر را در آنها تایپ کنید.

```
static class HasStaticConstructor
{
    public static int _test;
    // Static constructor initializes public field.
    static HasStaticConstructor()
    {
        _test = 1;
    }
}

static class NoStaticConstructor
{
    public static int _test = 1;
}
```

اکنون در متد اصلی کلاس program.cs، دو سازنده بالا را فراخوانی کنید.

```
static void Main(string[] args)
{
    System.Console.WriteLine(HasStaticConstructor._test);
    System.Console.WriteLine(NoStaticConstructor._test);
}
```

استفاده از مرجع this

هر شیء از طریق مرجع this می تواند به خودش مراجعه کند. مراجعه this به طور ضمنی می تواند به متغیرهای نمونه (فیلدها)، خواص و متدهای شیء مراجعه کند. در متدهای یک کلاس بعد از نوشتن واژه this باید از . استفاده نمود. ویرایشگر کد لیست فیلدها، خاصیت ها، متد و ... را که در کلاس مرتبط تعریف شده اند در اختیار ما قرار می دهد. در زیر چند بیان دیگر از این مطلب را بررسی می کنیم :

this فقط در داخل متدهای کلاس معنی دارد. اگر بخواهیم بدانیم در یک قطعه کد this به چه معنی است باید ببینیم در متد کدام کلاس قرار دارد زیرا this به معنی شیء جاری آن کلاس می باشد.

this تنها در زمان تعریف متدهای یک کلاس قابل استفاده می باشد.

برای مثال در متد SetDate() که بیان شد دستور Year = pYear; در واقع همان دستور Year = pYear; this می باشد. یعنی هر شیء که این متد را فراخوانی کرد (this) فیلد Year آن، برابر pYear قرار بگیرد.

در داخل هر قطعه کدی که this مشاهده گردد، در واقع this معرف شیء جاری کلاس در

برگیرنده آن قطعه کد (متد) است. مثلاً چنانچه در متد اداره کننده رویداد click یک کنترل button روی Form\ یعنی button\click از this استفاده شود، با توجه به اینکه this درون متد های کلاس Form است لذا this بیانگر شیء جاری Form مذکور می باشد.

یکی از کاربرد های مهم this عبارت است از :

زمانی که در متدی، پارامتری با نام یکی از فیلد های کلاس داریم this برای متمایز کردن فیلد کلاس از پارامتر همنام به کار می رود. برای مثال فرض کنید در متد SetDate پارامتر ها به همان نام فیلد ها یعنی Year, Month, Day باشند. در این صورت استفاده از this الزامی می شود تا تمایزی بین فیلد های کلاس و پارامتر ها ایجاد کند :

```
SetDate (int Year , int Month , int Day)
{
    this.Year = Year;
    this.Month = Month;
    this.Day = Day;
}
```

در کد بالا Year, Month, Day که بعد از this به کار رفته اند بیانگر فیلد های کلاس هستند و بدون this نشان دهنده پارامتر متد هستند.

فراخوانی سازنده ها از سازنده های دیگر

ممکن است بعضی اوقات خود را در موقعیتی بیابید که در یک کلاس چندین سازنده دارید این کار ممکن است به منظور اصلاح بعضی از پارامتر های اختیاری انجام گیرد که سازنده ها به طور معمول برای انجام این کار دارای کد هستند. به طور مثال کد زیر را در نظر بگیرید :

```
public class car
{
    private string description;
    private uint nWheels;
    public car (string model , uint nWheels)
    {
        this.description = description;
        this.nWheels = nWheels;
    }
    public car (string model)
    {
        this.description = description;
        this.nWheels = 4;
    }
}
```

هر دو سازنده فیلد های یکسانی را مقدار دهی می کنند. با قرار دادن تمام کد در یک جا

مرتب تر جلوه می کند و زبان C# نیز قاعده خاصی به نام یک مقدار دهنده سازنده برای انجام این کار دارد.

```
public class car
{
    private string description;
    private uint nWheels;
    public car (string description , uint nWheels)
    {
        this.description = description;
        this.nWheels = nWheels;
    }
    public car (string description) : this (description , 4)
    {
    }
}
```

در این میان کلمه کلیدی this منجر به فراخوانی سازنده با نزدیک ترین تطابق از نظر پارامتر می شود. توجه داشته باشید که هر مقدار دهنده سازنده، قبل از اجرای بدنه سازنده اجرا شده است. کد زیر را در حال اجرا در نظر بگیرید :

```
car mycar = new car ("Proton Persona");
```

در این مثال، سازنده دو پارامتری قبل از هر کدی در بدنه سازنده تک پارامتری اجرا می شود (تنها در این حالت که در بدنه سازنده تک پارامتری کدی وجود ندارد فرقی نمی کند که کدام یک زودتر اجرا شوند).

یک مقدار دهنده اولیه سازنده C# ممکن است همچنین شامل یک فراخوانی به سازنده دیگر در یک کلاس یکسان یا یک فراخوانی به یک سازنده در کلاس بالاتر از خود باشد (با استفاده از دستوری که ذکر شد، اما با استفاده از کلمه کلیدی base به جای this) با این حال امکان قرار دادن بیش از یک فراخوانی در مقدار دهنده وجود ندارد.

استفاده از سازنده های با سطح دسترسی خصوصی (private)

در زبان C# مانند سایر زبان های برنامه نویسی سطح بالا، ماژول سراسری (Global Module) نداریم. ممکن است بخواهیم کلاس های نرم افزاری کوچکی ایجاد کنیم که متد ها و یا فیلد های آن کاربردی سراسری (مشترک و قابل استفاده برای خیلی از کلاس ها) داشته باشند. در این حالت فیلد ها و متد های مذکور، استاتیک تعریف می شوند که همانطور که گفته شد بدون ایجاد شیء از طریق کلاس قابل دسترسی هستند.

برای اینکه شبیه سازی کاملی از مازول سابق داشته باشیم بایستی جلوی ایجاد شیء از این کلاس ها (کلاس هایی که تمام اعضای آن استاتیک هستند) را از برنامه نویس بگیریم. برای این کار، یک سازنده با سطح دسترسی خصوصی (private) در کلاس مذکور تعریف می کنیم تا برنامه نویس حتی اگر به اشتباه قصد ایجاد شیء ای را داشت با خصوصی کردن سطح دسترسی سازنده مانع این کار شویم زیرا از کلاسی که سازنده آن با سطح دسترسی خصوصی تعریف شده باشد، نمی توان شیء ایجاد کرد.

سازنده های همنام

همانند متد ها، سازنده های کلاس نیز می توانند همنام باشند. به عبارت دیگر می توان چند سازنده با نام یکسان برای یک کلاس تعریف کرد. روشن است که این سازنده ها باید در نوع، تعداد یا ترتیب پارامتر ها با هم فرق کنند.

اعضای ثابت و فقط خواندنی کلاس

فیلد های ثابت در داخل یک کلاس همانند یک ثابت متغیری است که شامل مقداری تغییر ناپذیر است و این چیزی است که علاوه بر زبان C# در سایر زبان های دیگر نیز وجود دارد. ثابت ها لزوماً تمام خواسته های ما را برآورده نمی کنند. بسته به موقعیت ممکن است متغیر هایی داشته باشیم که مقدار آنها نباید تغییر کند و نیز مقدار آنها تا زما اجرا مشخص نیست. زبان C# اجازه می دهد که برنامه نویسان بتوانند اعضای کلاس را ثابت و یا فقط خواندنی تعریف کنند. این اعضا به ترتیب با واژه های کلیدی const و readonly تعریف می شوند. اعضای داده ای که به صورت readonly تعریف می شوند دارای ویژگی های زیر هستند :

- ۱- اعضای داده ای می توانند در زمان اعلان یا در سازنده کلاس مقدار اولیه بگیرند.
- ۲- اعضای داده ای static و readonly پس از مقدار دهی اولیه، قابل تغییر نیستند، تنها متغیر های readonly که می توانند در چندین سازنده مقدار اولیه بگیرند و تنها یکی از آنها در زمان ایجاد شیء فراخوانی می شود.
- ۳- اعلان عضو کلاس به صورت readonly و تلاش برای استفاده از آن قبل از مقدار دهی اولیه، منجر به خطای منطقی می شود.

۴- تنها یک بار می توان در زمان اعلان و یا در داخل سازنده کلاس به اعضای readonly مقدار اولیه داد.

۵- زمان مقدار دهی اولیه به عضو استاتیک که readonly تعریف شده، در داخل سازنده باید از سازنده static استفاده کرد.

اعضای داده ای که به صورت const تعریف می شوند دارای ویژگی های زیر هستند :

۱- اعضای داده ای که به صورت const تعریف می شوند، باید در زمان ترجمه مقدار بگیرند. بنابراین، اعضای const فقط می توانند با مقادیر ثابت دیگر، مثل مقادیر صحیح، لیترال های رشته ای، کاراکترها و سایر اعضای ثابت تعیین شوند.

۲- اگر عضو کلاسی را const، تعریف کنید، ولی در زمان اعلان آن کلاس، به آن مقدار اولیه ندهید با خطا مواجه خواهید شد.

۳- اگر پس از مقدار اولیه دادن به عضو const، سعی کنید مقدار آن را عوض کنید، با خطا مواجه می شوید.

۴- اگر عضو const را استاتیک اعلان کنید با خطای نحوی مواجه می شوید، زیرا عضو const به طور ضمنی استاتیک است.

مثال ۱۹ - ۵: برنامه زیر زمان جاری سیستم را در زمان اجرای برنامه به کاربر نشان می دهد. هدف این برنامه آشنایی با فیلدهای از نوع readonly می باشد.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده، سپس یک کلاس داخلی با عنوان Manager ایجاد کرده، دستورات زیر را در آن بنویسید.

```
class Manager
{
    readonly DateTime _startup; // <-- the readonly field
    public Manager()
    {
        // Initialize startup time.
        this._startup = DateTime.Now;
    }
    public DateTime GetStartup()
    {
        // We cannot modify the DateTime here.
        return this._startup;
    }
}
```

در متد اصلی کلاس program.cs دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    Manager manager = new Manager();
    Console.WriteLine(manager.GetStartup());
}
```

اکنون برنامه را اجرا کرده و نتیجه را مشاهده کنید.

مثال ۲۰ - ۵: برنامه زیر یک عدد تصادفی از کامپیوتر دریافت کرده و محیط دایره را محاسبه می کند؟

حل: ابتدا یک پروژه جدید از نوع ویندوزی ایجاد کنید، سپس Form آن را مطابق پنجره تصویر ۳۳ - ۵ طراحی کنید.



تصویر ۳۳ - ۵

یک کلاس داخلی با عنوان readonlyconst ساخته و دستورات زیر را در آن تایپ کنید:

```
class readonlyconst
{
    //create constant PI
    public const double P = 3.14;
    //radius is a constant that ont initialized
    public readonly int radius;
    public readonlyconst(int radiusValue)
    {
        radius = radiusValue;
    }
}
```

حال به Form۱ برگشته و روی دکمه Click دابل کلیک کنید تا وارد رویداد click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    Random rand = new Random();
    readonlyconst conValue = new readonlyconst(rand.Next(1,20));
    label3.Text = "شعاع = " + conValue.radius + "\n محیط = " +
        2 * readonlyconst.P * conValue.radius;
}
```

اکنون برنامه را اجرا کرده و نتیجه را مشاهده کنید.

کلاس های تو در تو (Nested Classes)

کلاس های تو در تو، کلاس هایی هستند که در کلاس دیگری تعریف می شوند. کلاس اتومبیل را در نظر بگیرید اشیای این کلاس همواره اشیایی از چرخ را در خود دارند و معمولاً اشیای کلاس چرخ، موجودیت مجزا ندارند. در محیط فرضی برنامه شما مثلاً در کارخانه ماشین سازی، فروش چرخ به تنهایی ممکن نیست و چرخ همیشه درون ماشین در اختیار می باشد. در چنین مواردی که اشیای یک کلاس به صورت جزئی از اشیای کلاس دیگر، موجودیت می یابند (چرخ در ماشین) کلاس شیء داخلی را در درون کلاس شیء بیرونی تعریف می کنند. با این حال بر حسب سطح دسترسی کلاس های داخلی، امکان یا عدم امکان ایجاد شیء از کلاس داخلی مستقل از کلاس خارجی (بدون ایجاد شیء از کلاس خارجی)، به برنامه نویس داده می شود.

```
public class Machine
{
    =
    =    تعریف اعضای کلاس ماشین
    =
    private class wheel
```

```
    در داخل کلاس ماشین، تعریف کلاس چرخ هم انجام شده است
    {
    =
    =    تعریف اعضای کلاس چرخ
    =
    }
```

```
public wheel w1;
public wheel w2;
public wheel w3;
public wheel w4;
}
```

در مثال فوق در صورتی که کلاس wheel با سطح دسترسی خصوصی (private) تعریف

شود، امکان ایجاد شیء از این کلاس (کلاس wheel) مستقل از کلاس Machine وجود ندارد و ما چرخ ها را تنها در ماشین می بینیم ولی چنانچه این کلاس با سطح دسترسی عمومی (public) تعریف شود، علاوه بر اینکه اشیای چرخ می توانند به صورت جزئی از شیء ماشین وجود داشته باشند می توانند به طور مستقل نیز وجود داشته باشند. در این حالت برای ایجاد اشیاء کلاس چرخ (کلاس داخلی) به صورت مستقل، از روش زیر استفاده می شود :

اسم کلاس داخلی. اسم کلاس خارجی به عنوان نام کلاس، در دستور ایجاد شیء به کار می رود.

با این توضیح، دستور ایجاد شیء چرخ به صورت مستقل به صورت زیر است :

```
Machine.wheel w = new Machine.wheel();
```

معمولاً فیلدهایی از کلاس را که از نوع کلاس دیگری هستند، در سازنده کلاس مقدار دهی و ایجاد (new) می کنند تا زمانی که شیء کلاس ایجاد شد، تمام فیلدهای آن، آماده استفاده باشند (هر متغیری یا فیلدی که از نوع کلاس تعریف شده باشد، قبل از استفاده باید با شیء ای ایجاد شده یا شیء ای به آن تخصیص داده شود).

در مثال قبلی برای این که با ایجاد شیء ماشین، چرخ های آن آماده استفاده باشند باید در سازنده کلاس ماشین، فیلدهای چرخ را از کلاس چرخ ایجاد کنیم :

```
public Machine ()
{
    w1 = new wheel ();
    w2 = new wheel ();
    w3 = new wheel ();
    w4 = new wheel ();
}
```

حال فرض کنید که شیء ماشین را با دستور زیر ایجاد کنیم :

```
Machine m = new Machine ();
```

در این حالت با ایجاد شیء ماشین، w1 الی w4 هم در این شیء آماده استفاده است (مستقیم یا غیر مستقیم توسط متدها)، چون در هنگام ایجاد شیء ماشین در سازنده آن، چرخ ها ایجاد شده اند. در صورتی که فیلدهای چرخ مذکور (w1 الی w4) در سازنده کلاس چرخ ایجاد

نشوند، باید پس از ایجاد شیء اصلی (اینجا m) و قبل از استفاده از آنها، یک بار ایجاد شوند. البته این روش استفاده توصیه نمی شود و بهتر است ایجاد فیلد هایی که از نوع کلاس هستند در سازنده کلاس انجام پذیرد.

مثال ۲۱ - ۵: برنامه زیر نحوه استفاده از کلاس های تو در تو را نشان می دهد. در این برنامه کلاس B در داخل کلاس A تعریف شده است.

حل: ابتدا یک پروژه از نوع Console ایجاد کرده و سپس یک کلاس داخلی به نام A ایجاد کنید، سپس دستورات زیر را بنویسید.

```
class A
{
    public int v1;

    public class B
    {
        public int v2;
    }
}
```

حال در کلاس program.cs در متد اصلی برنامه دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    A a = new A();
    a.v1++;
    A.B ab = new A.B();
    ab.v2++;
}
```

سربار گذاری (Overloading) متد ها و سازنده ها

تعریف چند متد هم نام با ساختار متفاوت را سربار گذاری متد ها می گویند. ساختار یک متد توسط نام آن و لیست پارامتر ها (تعداد، نوع و ترتیب پارامتر ها) مشخص می شود. به طور ضمنی از این تعریف بر می آید که خروجی متد و نام پارامتر ها در ساختار متد تاثیری ندارند. امکان تعریف چند متد همنام در یک کلاس، تنها در صورتی امکان پذیر است که ساختار آنها متفاوت باشند. مثالی از سربار گذاری را در بحث سازنده ها در کلاس HijriClass دیدیم که همان سربار گذاری سازنده ها بود. در آنجا شما متد هایی (سازنده هایی) همنام کلاس و با

پارامتر های متفاوت داشتید که این امکان را به کلاس می داد تا اشیای آن با روش های مختلف بتوانند ایجاد شوند.

مثال سربارگذاری متد، نوشتن دو متد تقسیم همنام است که یکی در صورتی که پارامتر های آن، مقادیر صحیح باشند، خارج قسمت را بر می گرداند و دیگری در صورتی که پارامتر ها اعشاری باشند، مقدار دقیق تقسیم اعشاری را بر می گرداند. هر دو متد همنام هستند اما براساس پارامتر ها می خواهیم مقادیر مختلفی را برگردانند.

دستورات زیر، تعریف دو متد سربار شده Divide را در کلاس NumericClass نشان می دهد که با توجه به تفاوت نوع پارامتر ها ساختار متفاوتی دارند و در زمان فراخوانی بر حسب نوع پارامتر های ارسال شده، یکی از دو متد مناسب فراخوانی خواهند شد :

```
public int Divide (int p1 , int p2)
{
    return p1 / p2;
}
public double Divide (double p3 , double p4)
{
    return p3 / p4;
}
```

مثال ۲۲ - ۵ : برنامه زیر پیغام (Hello Iran) و (Hello Ardebil) را چاپ می کند. این برنامه آشنایی با نحوه سربار گذاری متد ها را بیان می کند.

حل : ابتدا یک پروژه از نوع Console ایجاد کنید، نیازی به ساخت یک کلاس داخلی نیست. متد ShowString را در خود کلاس program.cs تعریف می کنیم.

```
class Program
{
    static void ShowString()
    {
        // Send default argument to overload.
        ShowString("Hello Iran");
    }
    static void ShowString(string value)
    {
        // We don't need an if check here, which makes
        // ... calling this method directly faster.
        Console.WriteLine(value);
    }
}
```

در متد اصلی کلاس Program.cs از این متد های به صورت زیر استفاده می کنیم :

```
static void Main(string[] args)
{
    ShowString();
    ShowString("Hello Ardebil");
}
```

اکنون برنامه را اجرا کرده و نتیجه را مشاهده کنید.

مثال ۲۳ - ۵: برنامه زیر جمع دو عدد را محاسبه می کند. این برنامه نحوه سر بار گذاری متد ها را نشان می دهد.

حل : ابتدا یک پروژه جدید ویندوزی ایجاد کرده و یک کلاس داخلی به نام Math در آن ایجاد کنید، سپس Form آن را مطابق پنجره تصویر ۳۴ - ۵ طراحی کنید.



تصویر ۳۴ - ۵

حال دستورات زیر را در کلاس Math به صورت زیر بنویسید.

```
class Math
{
    public int Plus(int number1, int number2)
    {
        return Plus(number1, number2, 0);
    }

    public int Plus(int number1, int number2, int number3)
    {
        return number1 + number2 + number3;
    }
}
```

به Form برگشته و بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را وارد کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    Math a = new Math();
    int x = Convert.ToInt32(textBox1.Text);
    int y = Convert.ToInt32(textBox2.Text);
    int z;
    z=a.Plus(x, y);
    label4.Text = z.ToString();
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

تعریف مجدد عملگر ها

تا به اینجا از عملگر های استاندارد مانند + یا - برای کار بر روی نوع های داده ای خاص مانند اعداد صحیح، اعداد اعشاری و یا رشته ها استفاده کردیم. این عملگر ها به صورت درونی برای کار با نوع های داده ای برنامه ریزی شده اند. بعضی از زبان های برنامه نویسی مانند زبان C# به برنامه نویس اجازه می دهند که عملکرد این عملگر ها را به گونه ای تغییر دهند تا با ساختار ها و کلاس های نوشته شده توسط آنها نیز کار کنند.

برای مثال ممکن است بخواهیم ساختاری را ایجاد کنیم که بتواند اعداد مختلط را در خود نگهداری کند. همانطور که می دانید هر عدد مختلط از یک قسمت حقیقی و یک قسمت موهومی تشکیل شده است. فرض کنید در یک برنامه، کلاسی را برای نگهداری از اعداد مختلط به صورت زیر تعریف کرده اید :

```
public class ComplexNumber
{
    public double real;
    public double imaginary;
}
```

برای جمع دو شیء از این کلاس، باید قسمت های حقیقی هر کدام را با هم و قسمت های موهومی را نیز با هم جمع کنیم. اما عملگر + به صورت عادی نمی تواند اشیایی که از این کلاس نمونه سازی می شوند را با یکدیگر جمع کند. در C# این قابلیت وجود دارد تا برای عملگر + تعریف کنیم که چگونه باید دو شیء از این نوع را با یکدیگر جمع کند. بعد از انجام این کار، این عملگر قادر خواهد بود همانطور که دو عدد عادی را با هم جمع می کند، دو

شیء از کلاس ComplexNumber را نیز با هم جمع کند. به این کار سربارگذاری عملگر می گویند.

سربار گذاری عملگر (به نام چندریختی ادهاک نیز شناخته می شود) حالت ویژه ای از چند ریختی است که در آن بخشی و یا همه عملگرها مانند +، = یا == بسته به نوع پارامتر هایشان، پیاده سازی های متفاوتی دارند. گاهی سربار گذاری توسط زبان برنامه نویسی تعریف می شود. همچنین برنامه نویس می تواند برای پشتیبانی انواع جدید، عملگرها را دوباره پیاده سازی کند (سربار گذاری کند).

ادعا می شود سربار گذاری عملگر ها مفید است چون اجازه می دهد روند توسعه برنامه با استفاده از نشانه گذاری «نزدیک تر به هدف» انجام شود و این امکان را فراهم می سازد که انواع داده تعریف شده توسط کاربر با سطح نحوی همسانی با انواع داده توکار زبان برنامه نویسی به کار برده شوند. این کار به آسانی می تواند با فراخوانی متد ها شبیه سازی شود. برای مثال سه عدد صحیح a, b, c را در نظر بگیرید :

$$a + b * c$$

در یک زبان برنامه نویسی که از سربار گذاری عملگر ها پشتیبانی می کند با فرض این که تقدم عملگر * بیشتر از + است. در واقع عبارت بالا روش کارآتر نوشتن عبارت زیر است :

`add (a, multiply (b,c))`

همانطور که می دانید هر عملگر در برنامه نویسی دارای الویت خاصی است. برای مثال الویت عملگر * بیشتر از عملگر + است. پس برای ارزیابی مقدار عبارت $A + B * C$ ، ابتدا مقدار B در مقدار C ضرب شده و حاصل آن با مقدار A جمع می شود.

همچنین هر عملگر دارای شرکت پذیری خاصی است که مشخص می کند آن عملگر از سمت چپ به راست ارزیابی می شود و یا از سمت راست به چپ. برای مثال عملگر = دارای شرکت پذیری چپ است. به بیان دیگر برای ارزیابی مقدار $A = B = C$ مقدار C در B و سپس در A قرار می گیرد.

برای سربار گذاری عملگر ها در زبان C# از واژه کلیدی operator به صورت زیر استفاده می کنیم :

```

TypeName operator op (parameters)
{
...
}

```

TypeName نوعی است که توسط متد برگردانده می شود. operator واژه کلیدی است که به همین صورت استفاده می شود. op عملگری است که باید دوباره تعریف شود، و parameters لیست پارامترهایی است که باید به عملگر ارسال شوند. به عنوان مثال، دستور زیر عملگر + را برای جمع کردن دو شیء از کلاس Complex تعریف می کند.

```

public static Complex operator +(complex c1 , complex c2)
{
...
}

```

Complex نوعی است که توسط عملگر + برگردانده می شود، و این عملگر دارای دو پارامتر c1 و c2 از نوع Complex است.

هنگام سربار گذاری عملگر ها باید همواره نکات زیر را در نظر داشته باشید :

۱- هنگام سربار گذاری عملگر ها نمی توانید الویت و یا ترتیب شرکت پذیری آنها را تغییر دهید. زیرا این موارد به مفهوم عملگر بستگی دارند و براساس نوع داده ای که با آنها مورد استفاده قرار می گیرد نباید تغییر کنند. برای مثال عبارت $A + B * C$ صرف نظر از نوع داده ای متغیرهای A , B , C همواره به صورت $A + (B * C)$ ارزیابی می شود.

۲- هنگام سربار گذاری عملگر ها مجاز نیستید که تعداد عملوند های مورد نیاز آن را تغییر دهید. برای مثال عملگر * همواره دو عملوند را دریافت می کند و نمی توانید عملکرد آن را برای یک کلاس به گونه ای تغییر دهید که به سه عملوند نیاز داشته باشد.

۳- هنگام نوشتن یک برنامه مجاز نیستید که عملگر های جدیدی را ایجاد کرده و مفهومی را به آن اختصاص دهید. برای مثال نمی توانید عملگری را به صورت * تعریف کنید.

۴- از تعریف مجدد عملگر ها وقتی استفاده کنید که انجام عملیات، نسبت به فراخوانی عادی متد، روشن تر انجام شود. از تعریف مجدد عملگر ها به طور سازگار استفاده کنید به طوری که به قابلیت خوانایی برنامه آسیب نرساند. به عنوان مثال، عملگر + را طوری تعریف نکنید که دو کلاس را از هم تفریق کند.

مثال ۲۴ - ۵: برنامه زیر چگونگی سربرار گذاری عملگر های + و ++ را نشان می دهد.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و سپس یک کلاس داخلی به نام Widget به پروژه خود اضافه کنید، سپس دستورات زیر را بنویسید.

```
class Widget
{
    public int _value;
    public static Widget operator +(Widget a, Widget b)
    {
        // Add two Widgets together.
        // ... Add the two int values and return a new Widget.
        Widget widget = new Widget();
        widget._value = a._value + b._value;
        return widget;
    }
    public static Widget operator ++(Widget w)
    {
        // Increment this widget.
        w._value++;
        return w;
    }
}
```

حال در متد اصلی برنامه در کلاس program.cs دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    // Increment widget twice.
    Widget w = new Widget();
    w++;
    Console.WriteLine(w._value);
    w++;
    Console.WriteLine(w._value);

    // Create another widget.
    Widget g = new Widget();
    g++;
    Console.WriteLine(g._value);

    // Add two widgets.
    Widget t = w + g;
    Console.WriteLine(t._value);
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال ۲۵ - ۵: برنامه ای که با سربرار گذاری عملگر ها برای اشیاء، دو عدد مختلط را از

کاربر دریافت کرده و حاصل جمع و تفریق آنها را محاسبه کند؟



تصویر ۳۵ - ۵

در داخل کلاس ComplexNumber دستورات زیر را بنویسید.

```
class ComplexNumber
{
    public double Real = 0;
    public double Imaginary = 0;
    //override ToString () to display a complex number
    // in the traditional format
    public override string ToString()
    {
        return (Real + " + " + Imaginary + "i");
    }
    //overloading '+' operator :
    public static ComplexNumber operator +(ComplexNumber a, ComplexNumber b)
    {
        //Create a new ComplexNumber object to store the sum
        ComplexNumber sum = new ComplexNumber();
        sum.Real = a.Real + b.Real;
        sum.Imaginary = a.Imaginary + b.Imaginary;
        return sum;
    }
    //overloading '-' operator
    public static ComplexNumber operator -(ComplexNumber a, ComplexNumber b)
    {
        //Create a new ComplexNumber object to store the minus
        ComplexNumber minus = new ComplexNumber();
        minus.Real = a.Real - b.Real;
        minus.Imaginary = a.Imaginary - b.Imaginary;
        return minus;
    }
}
```

برای محاسبه جمع و تفریق دو عدد مختلط که کاربر در برنامه وارد کرده است، به چهار شیء از نوع ComplexNumber نیاز داریم. بر روی دکمه Click دابل کلیک کرده تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    ComplexNumber number1 = new ComplexNumber();
    ComplexNumber number2 = new ComplexNumber();
    number1.Real = double.Parse(textBox1.Text);
    number1.Imaginary = double.Parse(textBox3.Text);
    number2.Real = double.Parse(textBox2.Text);
    number2.Imaginary = double.Parse(textBox4.Text);
    ComplexNumber sum = new ComplexNumber();
    ComplexNumber minus = new ComplexNumber();
    sum = number1 + number2;
    minus = number1 - number2;
    label7.Text = sum.ToString();
    label8.Text = minus.ToString();
}
```

کافی است که برنامه را اجرا کرده و نتیجه را مشاهده کنید.

توضیح : برنامه را با ایجاد کلاسی به نام ComplexNumber برای نگهداری اعداد مختلط شروع کردیم. برای نگهداری این اعداد به دو فیلد، یکی برای نگهداری قسمت حقیقی و دیگری برای نگهداری قسمت موهومی نیاز داریم. بعد از ایجاد این دو فیلد، برای اینکه بتوانیم راحت تر این اعداد را نمایش دهیم متد ToString را به گونه ای override می کنیم تا این اعداد را به فرم عادی نمایش دهد.

در مرحله بعد بایستی عملگر های جمع و تفریق را برای این کلاس سر بار گذاری کنیم. برای سر بار گذاری یک عملگر باید متدی با یک قالب خاص به صورت static و public تعریف کنیم. نام این متد باید با کلمه کلیدی operator شروع شده و سپس عملگر مورد نظر را وارد کرد. برای مثال نام متدی که برای سر بار گذاری عملگر جمع به کار می رود باید برابر با + operator باشد. پارامتر های این متد عملوند هایی هستند که برای آن عملگر لازم است. مثلاً عملگر جمع به دو عملوند نیاز دارد تا بتواند آن را با یکدیگر جمع کند، پس باید دو پارامتر را برای متد مربوط به سر بار گذاری این عملگر مشخص کنیم. خروجی این متد هم باید هم نوع با نتیجه ی آن عملگر باشد. برای مثال در عملگر جمع، خروجی متد مربوط به سر بار گذاری آن باید برابر با نوع متغیری باشد که به عنوان حاصل جمع مشخص می شود. عملوند هایی که در این قسمت به عملگر + فرستاده می شوند، هر دو از نوع مختلط هستند. به بیان بهتر هر دوی عملوند ها، شیء ای از نوع ComplexNumber هستند. پس پارامتر های ورودی متد نیز از نوع ComplexNumber خواهند بود.

توجه کنید که اگر می خواستیم عملگر + را به گونه ای سربار گذاری کنیم که بتوانید یک شیء از نوع ComplexNumber را با یک عدد صحیح جمع کنیم، باید متد را به گونه ای تعریف می کردیم که شیء از نوع ComplexNumber و یک عدد از نوع int را به عنوان ورودی دریافت کند. به این ترتیب می توانستیم از دستوری مانند زیر برای جمع یک عدد مختلط با یک عدد صحیح استفاده کنیم.

```
sum = number + 20;
```

بعد از اینکه دو شیء از نوع ComplexNumber را با یکدیگر جمع کردیم، نتیجه نیز از نوع ComplexNumber خواهد بود، پس باید خروجی (مقدار برگشتی متد) را از نوع ComplexNumber مشخص کنیم. سپس می توانیم نحوه جمع کردن دو شیء از نوع مشخص شده را در بدنه ی متد معین کنیم. در اینجا برای جمع دو مقدار فرستاده شده، شیء جدیدی از نوع ComplexNumber تعریف کرده، قسمت های حقیقی دو عدد را با یکدیگر و قسمت های موهمی آن را نیز با هم جمع کرده حاصل را در شیء ComplexNumber جدید قرار می دهیم، سپس این شیء را به عنوان نتیجه بر می گردانیم.

برای سربارگذاری عملگر - نیز از همین روش استفاده می کنیم. این عملگر نیز مانند + دو شیء از نوع ComplexNumber را دریافت کرده و حاصل را نیز به صورت ComplexNumber بر می گرداند.

حال از کلاس ComplexNumber و دستورات آن در رویداد Click کنترل button استفاده می کنیم، که کد های مربوط به آن را در صفحه قبل می توانید مشاهده کنید.

نکته!

به تعریف مجدد عملگر ها سربارگذاری می گویند.

توجه! در مورد مفاهیم چند ریختی در بخش های بعدی صحبت خواهیم کرد.

کلاس های ناقص (Partial)

کلمه کلیدی partial به کلاس اجازه می دهد تا در چند فایل گسترش یابد. معمولاً یک کلاس به طور کامل در یک فایل جای خواهد گرفت. در مواقعی که چندین برنامه نویس نیاز به

دستیابی به یک کلاس دارند یا در اکثر اوقات که یک تولید کننده بخشی از یک کلاس را تولید می کند، این کلاس را در چندین فایل قرار می دهد.

کلمه کلیدی `partial` به سادگی قبل از کلاس به کار می رود. در مثال زیر، کلاس `TheBigClass` در دو فایل منبع جداگانه `BigClassPart1.cs` و `BigClassPart2.cs` قرار می گیرد.

```
//BigClassPart1.cs
partial class TheBigClass
{
    public void MethodOne ()
    {
        ...
    }
}

//BigClassPart2.cs
partial class TheBigClass
{
    public void MethodTwo ()
    {
        ...
    }
}
```

وقتی پروژه ای که این دو فایل را به عنوان بخشی از خود دارد کامپایل می شود یک نوع یکتا به نام `TheBigClass` با دو متد `MethodOne()` و `MethodTwo()` ایجاد خواهد شد.

اگر هر یک از کلمات کلیدی `public`, `private`, `protected`, `internal`, `abstract`, `sealed`, `new`, `generic constraint` در تعریف یک کلاس به کار رفته باشند برای تمام قسمت هایی از یک کلاس یکسان خواهند بود.

قسمت های تودرتو در مورد کلاس های ناقص تا زمانی قابل استفاده هستند که کلمه کلیدی `partial` قبل از کلاس تودرتو آمده باشد. خصوصیات، توضیحات، رابط ها و خصوصیات پارامتری، رابط ها نوع کلی و اعضا در هنگام کامپایل انواع ناقص به نوع مورد نظر، ترکیب خواهند شد.

نوع کلاس partial نوع جالبی است نمونه بارزی که از آن استفاده می شود و می توان به آن اشاره کرد در بخش کلاس طراحی Form در پروژه های از نوع ویندوزی است. در صورتی که در یک پروژه ویندوزی هستید بر روی دکمه F7 دابل کلیک کنید تا وارد بخش ویرایشگر کد شوید و به کد های زیر توجه کنید :

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
}
```

به کلمه partial در خط اول کد های بالا توجه کنید اگر به محیط طراحی Form خودتان برگردید در آن می توانید رنگ، عنوان، اندازه و خیلی از خصوصیات دیگر مربوط به Form را مشاهده کنید. همه این خصوصیات بایستی در جایی Set شوند. اما مساله اینجاست که این خصوصیات در کدام قسمت از کلاس تعریف شده اند که ما مشاهده نمی کنیم. همه این تنظیمات در یک متدی به نام public Form1() نوشته شده اند. پس بر روی دستور InitializeComponent راست کلیک کرده و گزینه Go To Definition را کلیک کنید.

چون کلاس ما از نوع partial تعریف شده توانستیم با یک متد داخلی به نام InitializeComponent فراخوانی کرده و دستورات آن را در جای دیگر بنویسیم. یکی از مزایای استفاده از این نوع کلاس ها آن است که از شلوغی کد های برنامه نویسی می کاهد و مانع از نامفهوم بودن کد ها می شود.

می توانیم کلاس های خود را از نوع partial تعریف کنیم و قسمتی از کد هایی را که لازم داریم نوشته و قسمت های دیگر را در جا های دیگر بنویسیم.

مثال ۲۶ - ۵: برنامه زیر نحوه استفاده از کلاس های partial را نشان می دهد.

```

public partial class myPartialClass
{
    public myPartialClass(string pString)
    {
        Console.WriteLine("I am in a partial class in Partial.cs.The parmeter passed is: "
            + pString);
    }
    public void doSomethingElse()
    {
        Console.WriteLine(" I am in Partial.cs ");
    }
}
public partial class myPartialClass
{
    public myPartialClass()
    {
        Console.WriteLine(" I am in a partial class in Program.cs");
    }
    public void doSomething()
    {
        Console.WriteLine(" I am in Progam.cs ");
    }
}

```

از کلاس myPartialClass به صورت زیر در متد اصلی برنامه استفاده می کنیم :

```

static void Main(string[] args)
{
    myPartialClass myCompleteObject = new myPartialClass();
    myCompleteObject.doSomething();
    myCompleteObject.doSomethingElse();
}

```

کلاس Object

تمام کلاس های NET. نهایتا از کلاس System.Object مشتق شده اند. در حقیقت اگر در هنگام تعریف یک کلاس، یک کلاس مبنا را مشخص نکنید، کامپایلر به طور خودکار فرض را بر این می گذارد که این کلاس از کلاس Object مشتق شده است.

یک برنامه نویس می تواند علاوه بر متد های عضو عمومی و محافظت شده که برای کلاس Object تعریف شده دسترسی پیدا کند، همچنین به سایر متد ها که در کلاس های دیگری تعریف شده نیز می تواند دسترسی داشته باشد.

لیست زیر متد های مربوط به کلاس System.Object را شرح می دهد :

`ToString()`: این متد به عنوان یک نماینده ابتدایی سریع و آسان برای ساختن یک رشته از روی اطلاعات یک شیء است. هنگامی که می خواهید از محتویات یک شیء با خبر شده و نیز گاهی برای اهداف اشکال زدایی از این متد استفاده کنید. این متد انتخاب های کمی برای قالب بندی داده ها دارد. به طور مثال، تاریخ ها می توانند طبق قاعده با تنوع عظیمی از قالب های داده ای مختلف ذکر شوند اما `DateTime.ToString()` هیچ انتخابی را برای این منظور در اختیار شما قرار نمی دهد.

`GetHashCode()`: این متد در صورتی به کار می رود که اشیاء در یک ساختار داده، مانند یک نگاشت جا گرفته باشند (همچنین مانند یک جدول درهم یا فرهنگ لغت). این متد توسط کلاس هایی که این ساختار ها را دستکاری می کنند، به کار می رود تا موقعیت یک شیء در ساختار مشخص شود. اگر تصمیم دارید تا کلاس خود را به نحوی بسازید که به عنوان کلیدی برای یک فرهنگ لغت به کار رود باید متد `GetHashCode()` را سربارگذاری کنید.

`Equals()` و `ReferenceEquals`: با جمع بندی این دو متد مختلف که جهت مقایسه تساوی اشیاء ایجاد شده اند استنباط خواهید کرد، چارچوب .NET، شمای پیچیده ای برای بررسی تساوی دارد. تفاوت های ظریفی در چگونگی استفاده از این دو متد در مقایسه با عملگر `==` وجود دارد. علاوه بر این محدودیت هایی در چگونگی سربار گذاری متد مجازی و تک پارامتری `Equals()` در صورت استفاده وجود دارد، زیرا بعضی از کلاس ها مبنای خاصی در فضای نام `System.Collections` دارند و متد را فراخوانی کرده و انتظار دارند این متد به طریق معینی رفتار کند.

`Finalize()`: این متد نزدیک ترین متد `C#` به مخرب ها در `C++` است و زمانی فراخوانی می شود که یک شیء مرجع به عنوان زباله جمع آوری شده است تا منابع را پاک کند. پیاده سازی `Object` از `Finalize()` واقعا هیچ کاری انجام نداده و توسط زباله روب نادیده گرفته می شود. معمولا وقتی به سربار گذاری متد `Finalize()` نیاز دارید که می خواهید منابع مدیریت نشده ای را که یک شیء به آنها ارجاع کرده است هنگام حذف آن شیء آزاد کنید. زباله روب نمی تواند این کار را به طور مستقیم انجام دهد زیرا تنها درباره منابع مدیریت شده اطلاع دارد، بنابراین هر پایان دهنده ای که شما فراهم می کنید اعتماد خواهد کرد.

GetType() : این متد یک نمونه از یک کلاس مشتق شده از Syetem.Type را باز می گرداند. این شیء می تواند یک محدوده وسیع از اطلاعات را درباره کلاس فراهم کند که شیء شما عضوی از آن بوده و شامل نوع پایه، متد ها، ویژگی ها و غیره است. System.Type همچنین نقطه ورودی به تکنولوژی انعکاس Net را فراهم می آورد.

MemberwiseClone() : این متد به سادگی یک کپی از شیء را ایجاد کرده و یک مرجع (یا در موردی که انواع مقدار به کار می روند، یک مرجع تبدیل شده) را به آن کپی باز می گرداند. توجه داشته باشید که کپی ایجاد شده یک کپی سطح است به این معنا که تمام انواع مقداری موجود در کلاس را کپی می کند. اگر کلاس شامل هر گونه مراجع تعبیه شده ای باشد، آنگاه تنها مراجع کپی خواهند شد و نه اشیایی که عمل ارجاع به آنها انجام می شود. این متد از نوع محافظت شده بوده و نمی تواند برای کپی اشیای خارجی فراخوانی شود. همچنین این متد مجازی نیست بنابراین شما نمی توانید آن را سربار گذاری کنید.

کلاس Object آنقدر مهم است که کلمه کلیدی Object در حقیقت اسم مستعار برای System.Object است. به جای System.Object از کلمه کلیدی Object استفاده کنید، زیرا ساده تر است و نیز با استفاده از کلمات کلیدی دیگر که معادل های طولانی تری دارند سازگاری دارد. اگر برنامه شما شامل یک دستور using System; باشد، می توانید از Object با یک O بزرگ بدون نوشتن System استفاده کنید. ولی بهتر این است که آن را سازگار با بقیه کلمات کلیدی بنویسید. همیشه از کلمه کلیدی object که ساده تر است، استفاده کنید.

گفتیم که تمام کلاس ها به طور ضمنی از کلاس System.Object ارث می برند و لازم نیست تعریف کنید که کلاس شما از System.Object ارث برده است (اگر چه در صورت تمایل می توانید این کار را انجام دهید). این واقعیت که کلاس شما یک کلاس است، به این معنی است که این کلاس به صورت خودکار از System.Object مشتق شده است (ارث برده است). یعنی اگر بنویسید :

```
class Circle
{
:
}
```

درست مثل این است که بنویسید :

```
class Circle : System.Object
{
:
}
```

درباره وراثت در بخش های بعدی صحبت خواهیم کرد. فعلاً کافی است بدانید که نتیجه این واقعیت که تمام کلاس ها به طور ضمنی از System.Object مشتق می شوند (ارث می برند) این است که یک متغیر از نوع object می تواند به یک نمونه از هر کلاس ارجاع کند. در مثال زیر، متغیر های c و o هر دو به یک شیء Circle ارجاع می کنند. بر این اساس که c از نوع Circle و o از نوع object است، دو دید از یک شیء را ارائه می کند :

```
Circle c;
c = new Circle (42);
object o;
o = c;
```

Box کردن (Boxing)

باکس کردن (Boxing)، فرایندی است که باعث می شود تا با یک نوع داده مقدار (Value Type) به صورت نوع داده ارجاع (Reference Type) رفتار شود. در این حالت به اصطلاح گفته می شود که شیء Box شده است.

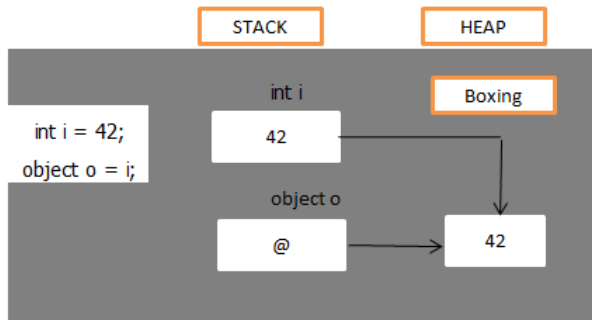
نوع داده مقدار با حافظه Stack و نوع داده ارجاع با حافظه Heap کار می کند. در واقع عمل Boxing و عمل UnBoxing نوع حافظه استفاده شده را تغییر می دهد.

همانطور که دیدید متغیر های از نوع object می توانند به هر شیء از یک نوع ارجاعی، ارجاع کنند. یعنی می توانند به هر نمونه از یک کلاس ارجاع کنند. متغیر های از نوع object می توانند به هر مقداری از هر نوعی نیز ارجاع کنند. برای مثال دو دستور زیر، متغیر i از نوع int که یک نوع مقداری است را با ۴۲ مقدار دهی اولیه می کند و متغیر o از نوع object که یک نوع ارجاعی است را با i مقدار دهی اولیه می کند :

```
int i = 42;
object o = i;
```

اثر این مقدار دهی دوم، مستلزم دقت و توجه است. ارجاعی که در داخل متغیر o قرار دارد، به متغیر i ارجاع نمی کند. باید به خاطر داشته باشید که i از نوع مقداری است و در پشته قرار

دارد. اگر ارجاع داخل 0 به i انجام می شد این ارجاع به پشته بود. از آنجایی که استحکام برنامه را به طور جدی تضعیف می کند و ممکن است یک نقص امنیتی ایجاد کند مجاز نیست. به جای آن، یک کپی دقیق از مقدار داخل i روی Heap ایجاد می شود و ارجاع داخل 0، به این کپی ارجاع می کند. این کپی خودکار Boxing نامیده می شود. در تصویر ۳۶ - ۵ آنچه که رخ می دهد نشان داده شده است.



تصویر ۳۶ - ۵

در مورد Boxing به خاطر داشته باشید که اگر مقدار اصلی یک متغیر را تغییر دهید، مقداری که در Heap وجود دارد تغییر نمی کند، زیرا این فقط یک کپی است. به خاطر عمل Boxing، یک متغیر از نوع Object می تواند به یک از این دو چیز ارجاع کند:

- ۱- این متغیر می تواند به هر نمونه از یک نوع ارجاعی (به یک object) ارجاع کند.
- ۲- این متغیر می تواند از طریق Boxing، به یک کپی از یک نوع مقداری (به یک مقدار) ارجاع کند. یعنی، یک متغیر از نوع object می تواند به هر چیزی بدون توجه به نوع آن ارجاع کند.

از Box بیرون کردن (UnBoxing)

عکس فرایند Box کردن است که موجب می شود مجدداً با شیء به صورت نوع داده مقدار (Value Type) رفتار شود. Box کردن به صورت تلویحی است و به طور خودکار انجام می شود ولی عمل UnBoxing به صورت صریح انجام می شود. توجه داشته باشید از آنجایی که یک متغیر از نوع object می تواند به یک کپی Box شده از

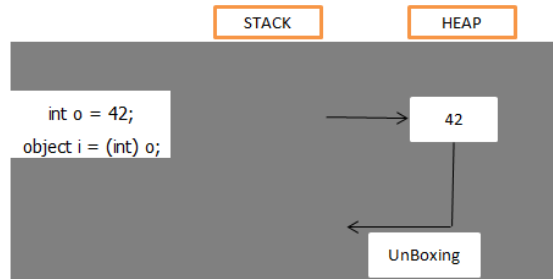
یک مقدار ارجاع کند، منطقی است که فقط به شما اجازه داده شود که به آن مقدار Box شده از طریق متغیر دستیابی پیدا کنید. ممکن است انتظار داشته باشید که بتوانید با استفاده از `int i` `i = 0` به مقدار Box شده از نوع `int` که متغیر `o` به آن ارجاع می کند دستیابی پیدا کنید. ولی، اگر این دستور را بنویسید، یک خطای زمان کامپایل ایجاد خواهد شد. به جای این دستور، باید از دستور `object o = 42` استفاده کنید که در آن `o` به یک مقدار Box شده از نوع `int` که مقدار 42 را در خود دارد، ارجاع می کند. در اینجا چند مثال آورده شده است :

```
object o = 42; // Okay
int i = o; // compile time error
```

اگر کمی فکر کنید، می فهمید که کاملاً منطقی است که نمی توانید از `int i = o` استفاده کنید. همانطور که می دانید `o` می تواند به هیچ چیز ارجاع نکند. با این حال یک راه حل برای بدست آوردن مقدار یک کپی Box شده وجود دارد. باید از تبدیل نوع استفاده کنید. برای این کار باید قبل از اسم متغیر از نوع `object`، اسم نوع را در داخل پرانتز بنویسید. برای مثال :

```
object o = 42; // Boxes
int i = (int) o; // compile okay
```

اثر این تبدیل نوع، مستلزم دقت و توجه است. کامپایلر متوجه می شود که نوع `int` را در تبدیل نوع مشخص کرده اید. بعد، چک می کند که `o` واقعا به چه چیزی ارجاع می کند. `o` می تواند به هیچ چیز ارجاع نکند. همین که تبدیل نوع بگوید `o` به یک `int` ارجاع می کند، به این معنی نیست که `o` واقعا به یک `int` ارجاع می کند. اگر `o` واقعا به یک مقدار Box شده از نوع `int` ارجاع کند و همه چیز مطابقت کند، تبدیل نوع با موفقیت انجام می شود و کامپایلر مقدار را از متغیر Box شده استخراج می کند. در این مثال، مقدار Box شده برای مقدار دهی اولیه `i` مورد استفاده قرار می گیرد. به این کار `UnBoxing` می گویند. در این مثال، این کار با موفقیت انجام شده است. در اینجا مثالی از یک `UnBoxing` موفقیت آمیز آورده شده است (تصویر ۳۷ - ۵).



تصویر ۳۷ - ۵

ولی اگر `o` به یک متغیر `Box` شده از نوع `int` ارجاع نکند، انواع با هم مطابقت نمی کنند و باعث می شود که تبدیل نوع با موفقیت انجام نشود. بلکه کامپایلر یک `InvalidCastException` ایجاد می کند. در اینجا مثالی از یک تبدیل نوع `UnBoxing` آورده شده است که با موفقیت انجام نمی شود :

```
Circle c = new Circle (42);
object o = c; //doesn't box because Circle is a class
int i = (int) o; // compiles okay, but throws at runtime
```

نوعی که در تبدیل نوع `UnBoxing` مشخص می کنید باید دقیقا با نوع مقدار `Box` شده مطابقت کند. برای مثال، اگر این متغیر یک کپی از یک `int` در خود داشته باشد و بخواهید آن را به یک نوع `long`، `UnBox` کنید باعث ایجاد یک `InvalidCastException` می شوید. این واقعیت که یک تبدیل نوع ضمنی از `int` به `long` به صورت نهادین وجود دارد نادرست است، زیرا تطابق باید به صورت کامل و دقیق باشد.

مخرب ها (Destructors)

مخرب ها، عکس سازنده ها هستند. بر خلاف سازنده ها که در زمان ایجاد شیء فراخوانی می شوند، مخرب ها در زمان نابودی شیء فراخوانی می شوند و فراخوانی آنها خودکار است. مخرب ها مانند سازنده ها در تعریف نام کلاس را دارند ولی علامت `~` قبل از نام کلاس، تعریف می شود. نحوه تعریف مخرب ها به صورت زیر است :

```
~ نام کلاس
{
=
= کد های مربوطه
=
}
```

در قطعه کد زیر کلاسی با نام ClassA تعریف شده و سازنده و مخربی نیز برای آن نوشته شده است. بررسی کنید که در هر زمان دلخواه، مقدار فیلد استاتیک C معرف و بیانگر چه چیزی می باشد؟

```
public class ClassA
{
    private static int C = 0;
    public ClassA ()
    {
        C = C + 1;
    }
    ~ClassA()
    {
        C = C - 1;
    }
    public static int GetC ()
    {
        return c;
    }
}
```

در کلاس ClassA فیلد استاتیکی به نام C از نوع صحیح و با مقدار اولیه صفر تعریف شده است. سطح دسترسی این فیلد private قرار داده شده است تا به طور مستقیم خارج از کلاس تغییر داده نشود. در سازنده کلاس، مقدار این فیلد استاتیک یک مقدار افزایش می یابد (این سازنده تنها سازنده کلاس است و برای ایجاد شیء با دستور new ClassA() این سازنده فراخوانی می شود).

در مخرب کلاس از مقدار C یکی کم می شود. بدین معنی که هر گاه شیء ای از کلاس ClassA نابود شود، مقدار C یکی کم می شود. با کنار هم گذاشتن موارد بالا، یعنی افزایش C با ایجاد شیء و کاهش C با نابودی شیء مقدار C در هر زمان معرف تعداد اشیاء حال حاضر کلاس ClassA در آن زمان می باشد.

مثال ۲۷ – ۵ : در مورد کلاس System.Object صحبت کردیم و متد های آن را معرفی کردیم. در این بخش قصد داریم با برنامه های ساده و کوتاه عملکرد هر یک از متد های مربوطه را نشان دهیم :

متد ToString():

```
class Employee
{
    string m_name;

    public Employee(string name)
    {
        m_name = name;
    }
    public override string ToString()
    {
        return String.Format("[Employee: {0}]", m_name);
    }
}

static void Main(string[] args)
{
    Employee emp = new Employee("Reza");
    Console.WriteLine(emp.ToString());
}
```

متد GetHashCode():

```
class Employee
{
    string m_name;
    public Employee(string name)
    {
        m_name = name;
    }
    public override int GetHashCode()
    {
        string uniqueString = ToString();

        return uniqueString.GetHashCode();
    }
    public override string ToString()
    {
        return String.Format("[Employee: {0}]", m_name);
    }
}

static void Main(string[] args)
{
    Employee emp = new Employee("Reza");
    Console.WriteLine(emp.GetHashCode());
    Console.ReadLine();
}
```

متد Equals() :

```
class Employee
{
    string m_name;
    public Employee(string name)
    {
        m_name = name;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Program eqDemo = new Program();
        eqDemo.InstanceEqual();
        Console.ReadLine();
    }
    public void InstanceEqual()
    {
        string name = "Reza";
        Employee employee1 = new Employee(name);
        Employee employee2 = new Employee(name);
        // comparing references to separate instances
        bool isEqual = employee1.Equals(employee2);
        Console.WriteLine("employee1 == employee2 = {0}", isEqual);
        employee2 = employee1;
        // comparing references to the same instance
        isEqual = employee1.Equals(employee2);
        Console.WriteLine("employee1 == employee2 = {0}", isEqual);
    }
}
```

متد Finalize() :

```
class Employee
{
    string m_name;
    public Employee(string name)
    {
        m_name = name;
    }
    ~Employee()
    {
        Console.WriteLine("Employee destructor executed.");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Employee emp = new Employee("Reza");
        Console.ReadLine();
    }
}
```

متد ReferenceEquals():

```
class Employee
{
    string m_name;
    public Employee(string name)
    {
        m_name = name;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Program refEqDemo = new Program();
        refEqDemo.InstanceEqual();
        Console.ReadLine();
    }
    public void InstanceEqual()
    {
        string name = "Reza";
        Employee employee1 = new Employee(name);
        Employee employee2 = new Employee(name);
        // comparing references to separate instances
        bool isEqual = Object.ReferenceEquals(employee1, employee2);
        Console.WriteLine("employee1 == employee2 = {0}", isEqual);
        employee2 = employee1;
        // comparing references to the same instance
        isEqual = Object.ReferenceEquals(employee1, employee2);
        Console.WriteLine("employee1 == employee2 = {0}", isEqual);
    }
}
```

متد GetType():

```
class Employee
{
}
class Program
{
    static void Main(string[] args)
    {
        object emp1 = new Employee();
        Employee emp2 = new Employee();
        Console.WriteLine(emp1.GetType());
        Console.WriteLine(emp2.GetType());
        Console.ReadLine();
    }
}
```

متد MemberwiseClone():

```

public class Address
{
}
class Employee
{
    Address m_address = new Address();
    string m_name;

    public Employee(string name)
    {
        m_name = name;
    }
    public Employee ShallowCopy()
    {
        return (Employee)MemberwiseClone();
    }
    public Address EmployeeAddress
    {
        get
        {
            return m_address;
        }
    }
}

static void Main(string[] args)
{
    Employee emp1 = new Employee("Reza");
    Employee emp2 = emp1.ShallowCopy();
    // compare Employee references
    bool isEqual = Object.ReferenceEquals(emp1, emp2);
    Console.WriteLine("emp1 == emp2: {0}", isEqual);
    // compare references of Address object in each Employee object
    isEqual = Object.ReferenceEquals(emp1.EmployeeAddress, emp2.EmployeeAddress);
    Console.WriteLine("emp1.EmployeeAddress == emp2.EmployeeAddress: {0}", isEqual);
    Console.ReadLine();
}

```

توجه! دستورات موجود در مثال های بالا اغلب شامل موارد گفته شده تا به اینجا می باشد. سعی کنید مباحث گفته شده را مرور کنید و روی مثال های گفته شده تامل نموده و در محیط Visual Studio چندین بار برنامه نویسی کنید تا ملکه ذهن تان شود. این کار به تکمیل درک شما از مباحث فوق الذکر کمک بیشتری می کند.

مثال ۲۸ - ۵: برنامه زیر نحوه استفاده از مخرب ها را نشان می دهد.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و یک کلاس داخلی به عنوان Example برای آن ایجاد کنید، سپس دستورات زیر را در آن بنویسید.

```
class Example
{
    public Example()
    {
        Console.WriteLine("Constructor");
    }
    ~Example()
    {
        Console.WriteLine("Destructor");
    }
}
```

در متد اصلی برنامه کلاس program.cs یک نمونه با عنوان x ساخته و برنامه را اجرا کنید.

```
static void Main(string[] args)
{
    Example x = new Example();
}
```

مثال ۲۹ – ۵: برنامه زیر کارکرد یک سازنده و مخرب را نشان می دهد. توضیحات اضافی در داخل برنامه به صورت Comment بیان شده است.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و سپس یک کلاس داخلی به نام CalssRom ایجاد کرده و دستورات زیر را در آن بنویسید.

```
class ClassRoom
{
    public int boycount;    //field
    public ClassRoom()      //default constructor
    {
        boycount = 30;
    }
    ~ ClassRoom()
    {
        boycount = 0;
    }
}
```

حال در متد اصلی کلاس program.cs دستورات زیر را بنویسید.

```
class Program
{
    static void Main(string[] args)
    {
        ClassRoom r;        //creating a class variable, object
        r = new ClassRoom(); //initializing class variable
        //or direct use "ClassRoom r = new ClassRoom();"
        int x = r.boycount;
        Console.WriteLine(x);
        Console.ReadKey();
    }
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

زباله روب (Garbage Collector)

.NET برای مدیریت حافظه و نیز بازیابی حافظه مورد نیاز برنامه های کاربردی از زباله روب یا Garbage Collector استفاده می کند. تا به حال دو تکنیک برای آزاد سازی و استفاده مجدد از حافظه ای که به طور پویا از طرف سیستم تقاضا می شود توسط سیستم عامل ویندوز مورد استفاده قرار گرفته است :

۱- وارد کردن برنامه به انجام این کار

۲- وادار کردن اشیا به نگهداشتن تعداد مرجع ها

واگذاری مسئولیت آزاد سازی حافظه به کد برنامه تکنیکی است که توسط زبان های سطح پایین و پر بازده مانند C++ مورد استفاده قرار می گیرد. این کار مفید بوده و سودش این است که منابع بیشتر از حد نیاز اشغال نمی شوند. بزرگترین عیب این کار، افزایش اشکالات است. کدی که درخواست حافظه می کند باید به وضوح سیستم را در هنگام عدم نیاز به آن حافظه باخبر کند در غیر این صورت باعث کمبود حافظه می شود.

اگر چه محیط های توسعه پیشرفته، ابزار هایی را برای کمک به کشف کمبود های حافظه پیشنهاد کرده اند، اما اشکالات بزرگی را بوجود آورده اند زیرا آنها تا زمان بوجود آمدن کمبود حافظه به شکلی که سیستم عامل ویندوز نتواند حافظه ای به فرایند ها اختصاص دهد هیچ تاثیری ندارد. با به وجود آمدن این شرایط، کامپیوتر تا فراهم آمدن میزان حافظه مورد نیاز کند می شود.

بر خلاف زبان های C و C++ که برنامه نویس باید حافظه را مدیریت کند، C# مدیریت حافظه را به طور داخلی انجام می دهد.

محیط کاری .NET، زباله رویی حافظه را انجام می دهد. این برنامه با هدف پاکسازی حافظه ساخته شده است و ایده آن این است که تمام حافظه ای که به صورت پویا درخواست شده روی Heap جای دارد (بدین صورت که Heap CLR مدیریت شده خود را به منظور استفاده برنامه های .NET نگه می دارد).

هنگامی که .NET تشخیص می دهد Heap مدیریت شده یک فرایند، پر شده و نیاز به پاک

سازی دارد، زباله روب را فراخوانی می کند. زباله روب از میان متغیر هایی که در حال حاضر در محدوده کد شما قرار دارند اجرا شده و مراجع اشیای ذخیره شده روی Heap را به منظور تشخیص مورد استفاده بودن هر یک از آنها از طریق کد شما امتحان می کند. این کار به منظور تشخیص اشیایی است که به آنها ارجاعی صورت گرفته است. هر شیء که بدان ارجاعی صورت نمی گیرد در کد شما غیر قابل دسترس تشخیص داده شده و می تواند حذف شود.

در واقع وقتی زباله روب اجرا می شود اشیایی را می یابد که برنامه به آنها مراجعه ای ندارد. این شیء را می توان در همان لحظه یا اجرای بعدی زباله روب جمع آوری کرد. بنابراین نشت های حافظه که در زبان هایی مثل C و C++ وجود دارد و حافظه به طور خودکار بازیابی نمی شود، در C# بوجود نخواهد آمد.

تخصیص و آزاد سازی منابع، مثل اتصال های شبکه، اتصال های بانک اطلاعاتی و فایل ها، صریحا باید توسط برنامه نویس صورت گیرد. یک روش اداره کردن این منابع (همراه با زباله روب)، تعریف مخرب است که منابع را به سیستم بر می گرداند. قبل از اینکه زباله روب، حافظه اشیاء را بازیابی کند مخرب شیء را فراخوانی می کند تا کار های نهایی را روی آن شیء انجام دهد.

وقتی زباله روب در حال حذف شیء ای از حافظه است، ابتدا مخرب شیء را فراخوانی می کند تا منابع مورد استفاده آن کلاس را آزاد کند. اما نمی توان دقیقا مشخص کرد که مخرب کی فراخوانی می شود، زیرا نمی توانیم دقیقا مشخص کنیم که کی زباله روب فراخوانی می گردد.

نکته اساسی در مورد مخرب ها این است که نمی توانید آنها را فراخوانی کنید. زیرا در زبان C#، خودتان نمی توانید یک شیء را از بین ببرید و هیچ دستور زبانی برای این کار وجود ندارد. دلایل خوبی وجود دارد که چرا طراحان C# به شما اجازه نداده اند که اشیاء را با نوشتن کد و به طور صریح از بین ببرید. اگر قرار بود که شما اشیاء را از بین ببرید ممکن بود : فراموش کنید که شیء را از بین ببرید. یعنی مخرب شیء در صورت وجود اجرا نمی شد و حافظه مربوط به آن به Heap برگردانده نمی شد و خیلی راحت، دیگر حافظه ای باقی نمی ماند.

یک شیء را بیش از یک بار از بین می بردید.

این مشکلات در زبانی مثل C# که استحکام و امنیت را جزء اهداف اصلی طراحی خود دارد قابل قبول نیست. Garbage Collector که به اختصار gc نیز می گویند تضمین می کند که :
خواه برنامه شما به صورت صریح تمام اشیایی که ایجاد کرده است را از بین برده باشد و
خواه از بین نبرده باشد، هر شیء از بین می رود. مثلاً، وقتی که برنامه به اتمام برسد تمام اشیاء
جاری از بین خواهند رفت.

هر شیء فقط یک بار از بین می رود.

هر شیء تنها زمانی از بین می رود که هیچ ارجاعی به آن نشود. این نوع از اشیاء را اشیاء غیر
قابل دسترس (Unreachable) می گویند.

این موارد بسیار سودمند هستند و شما را به عنوان یک برنامه نویس از کار های خسته کننده
که احتمال اشتباه در آن زیاد است، بی نیاز می کند و به شما امکان می دهد تا روی منطق
برنامه تمرکز کنید و خلاقانه تر کار کنید. با این حال، همه اینها به یک قیمتی انجام می شوند.
شما ترتیب از بین رفتن اشیاء را نمی دانید. همچنین نمی دانید که چه زمانی gc تصمیم می
گیرد که اشیاء را از بین ببرد. یک شیء درست در لحظه ای که غیر قابل دسترس می شود، از
بین نمی رود. از آنجایی که از بین بردن اشیاء یک عملیات وقت گیر است، gc فقط وقتی
اشیاء را از بین می برد که لازم باشد وقتی که حافظه heap تمام شده باشد و یا وقتی که
صریحا از آن بخواهید، این کار با فراخوانی متد System.GC.Collect قابل انجام است.
مسلماً زبان C#، برای برنامه هایی که در آنها مسئله وقت خیلی مهم و حساس است، زبان
مناسبی نیست.

نحوه اجرای Garbage Collector

gc در فرایند خود اجرا می شود و تنها زمانی اجرا می شود که بقیه پردازش ها در یک حالت
ایمن (مثلاً وقتی که در حالت معلق هستند) باشند. این مساله بسیار مهم است زیرا وقتی که gc
اجرا می شود، نیاز دارد که اشیاء را حرکت دهد و ارجاعات اشیاء را بروز کند. مراحل که
gc در هنگام اجرای می کند شامل موارد زیر است :

۱- یک گراف از اشیای قابل دسترسی ایجاد می کند و این کار را با دنبال کردن پشت سر هم فیلد های ارجاعی در اشیاء انجام می دهد. gc این گراف را بسیار با دقت می سازد و اطمینان حاصل می کند که ارجاعات حلقوی موجب یک جریان بازگشتی بی نهایت نمی شوند. هر شیء ای که در این گراف نباشد باید یک شیء غیر قابل دسترس باشد.

۲- بعد بررسی می کند تا ببیند که آیا هیچ کدام از اشیای غیر قابل دسترس به نهایی شدن احتیاج دارند یا خیر. به عبارت دیگر، بررسی می کند که آیا این اشیاء مخرب دارند یا خیر. هر شیء غیر قابل دسترسی که به نهایی شدن احتیاج داشته باشد، در یک صف ویژه به نام صف Freachable قرار می گیرد.

۳- بقیه اشیاء غیر قابل دسترس که احتیاج به نهایی شدن ندارند آزاد می شوند. gc یک شیء قابل دسترس را حرکت می دهد و ارجاعات مربوط به آن را نیز بروز می کند.

۴- حال gc به بقیه پردازش ها اجازه می دهد که از سر گرفته شوند.

۵- اشیاء غیر قابل دسترس که به نهایی شدن احتیاج دارند و در صف Freachable قرار دارند در یک پردازش جداگانه از بین می روند.

نوشتن کلاس هایی که دارای مخرب باشند، به پیچیدگی کد و روند gc می افزایند و اجرای برنامه را کند تر می کنند. اگر در برنامه هیچ مخربی وجود نداشته باشد، gc مجبور نیست که مراحل ۳ و ۵ را انجام دهد. اولین توصیه این است که سعی کنید از مخرب استفاده نکنید مگر در شرایطی که واقعا به آن احتیاج داشته باشید. برای مثال به جای آن از دستور using استفاده کنید که در بخش های بعدی به آن می رسیم.

اگر مخرب می نویسید، باید آن را با دقت بنویسید. به ویژه، باید آگاه باشید که اگر مخرب شما اشیای دیگر را فراخوانی می کند، ممکن است gc مخرب مربوط به این اشیاء را فراخوانی کرده باشد (به یاد داشته باشید که ترتیب از بین رفتن مشخص نیست).

مطمئن شوید که هر مخربی یک منبع را بر می گرداند و نه چیز دیگری را. این امر می تواند مستلزم این باشد که یک کلاس را به دو کلاس تقسیم کنیم که یکی از این کلاس ها برای مدیریت منابع در نظر گرفته شود.

مدیریت منابع

گاهی درست نیست که یک منبع را در یک مخرب آزاد کنیم. برخی از منابع آنقدر ارزشمند و نادر هستند که نباید به مدت دلخواهی به صورت آزاد نشده باقی بمانند. همانطور که می دانید، شما دقیقاً نمی دانید که gc چه زمانی، مخرب یک شیء را فراخوانی می کند. منابع باید تا آنجایی که ممکن است، سریع آزاد نشوند. در این شرایط تنها انتخاب شما این است که خودتان منابع را آزاد کنید. متد آزاد سازی، متدی است که یک منبع را آزاد می کند. اگر کلاسی، یک متد آزاد سازی داشته باشد، می توانید آن را فراخوانی کنید و به این ترتیب بر زمان آزاد شدن منبع، کنترل داشته باشید.

مدیریت خطا در زبان C#

یکی از قسمت های مهم در هر زبان برنامه نویسی، کنترل و مدیریت خطا هاست. دقت کنید خطا ها به دو دسته کلی تقسیم می شوند :

۱- خطاهایی که در اثر Syntax نادرست دستورات در زمان برنامه نویسی توسط برنامه نویس رخ می دهد و برنامه اجازه کامپایل را از برنامه نویس می گیرد.

۲- خطاهایی که ممکن است بعد از اتمام پروژه رخ دهد. بطور مثال پایگاه داده برنامه شما به هر دلیلی از بین برود. برنامه می خواهد به پایگاه داده متصل شود به دلیل عدم پیدا کردن پایگاه داده خطا رخ می دهد. و یا وقتی که یک عمل تقسیم بر صفر رخ دهد که در این حالت نیز در صورت برخورد نامناسب با خطا روبرو خواهید شد.

پس بهتر است که خطا ها را کنترل کنیم و در صورت بروز خطا فرایندی را تشکیل دهیم که مانع از بوجود آمدن مشکلات جدی در برنامه های خودمان شویم. در فصل های بعدی به طور کامل پردازش خطا و استثنا ها را مورد بررسی قرار خواهیم داد ولی با توجه به نیاز بخش های بعدی این فصل به دستورات کنترل خطا مقدمه ای از آن را در این قسمت بیان می کنیم.

اکنون به بررسی قسمت های مختلف دستورات مدیریت خطا در زبان C# می پردازیم. ابتدا کلمه کلیدی try را می نویسیم و یک بلاک برای آن قرار می دهیم حال دستورات عادی که قرار است در برنامه اجرا شوند را درون این بلاک می نویسیم. حال اگر قرار باشد در صورت بروز خطا یک سری دستورات خاص اجرا شوند دوباره از کلمه کلیدی catch استفاده می

کنیم و یک بلاک جدید برای آن بوجود می آوریم و دستوراتی که در زمان بوجود آمدن خطا بایستی اجرا شوند درون این بلاک می نویسیم. و باز هم از کلمه کلیدی finally استفاده کرده و یک بلاک جدید ایجاد می کنیم و دستوراتی که قرار است بعد از catch اجرا شوند درون این بلاک می نویسیم.

دستورات بخش catch می تواند پیشرفته تر باشد می توانیم برای این بخش پارامتر های ورودی، استثنا (Exception) تعریف کرده و از آن استفاده کنیم. ساختار کلی استفاده از دستورات ذکر شده به صورت زیر می باشد.

```
try
{
    ;دستورات اصلی برنامه
    ...
}
catch
{
    ;دستورانی که در صورت وجود خطا در بخش try بایستی اجرا شود
    ...
}
finally
{
    ;دستورانی که پس از بخش catch اجرا می شود
    ...
}
```

در برنامه زیر می خواهیم نحوه استفاده از دستورات بالا را به صورت یک پروژه عملی نشان دهیم. در این برنامه کاربر عددی را وارد کرده و عدد مزبور بر ۱۰۰ تقسیم می شود و در صورت بروز خطا دستورات بخش catch و finally اجرا خواهد شد.

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int a = 100 / Convert.ToInt32(textBox1.Text);
    }
    catch (Exception Ex)
    {
        MessageBox.Show(Ex.Message);
    }
    finally
    {
        MessageBox.Show("Finally was Raised");
    }
}
```

برنامه را اجرا کنید و در زمان اجرا در کنترل TextBox عدد صفر را وارد کنید خطای تقسیم نمایش داده می شود و پیام بخش finally نمایش داده می شود. اگر عدد ۲ را هم وارد کنید باز هم پیام بخش finally نمایش داده خواهد شد.

نکته! می توان یکی از دو بخش catch و finally را با توجه به کاربرد برنامه نوشته و یا از آن صرف نظر کرد.

در جلد دوم کتاب به طور کامل پردازش استثنا و مدیریت خطا در زبان C# بررسی شده است.

الگوی متد آزاد سازی

مثالی از یک کلاس که شامل یک متد آزاد سازی است، کلاس TextReader از یک فضای نام به نام System.IO می باشد. TextReader یک متد مجازی به نام Close دارد. کلاس های StreamReader (که کاراکتر ها را از یک Stream می خواند) و StringReader (که کاراکتر ها را از یک رشته می خواند) از کلاس TextReader ارث می برند و هر دو متد، متد close را Override می کنند. در اینجا مثالی به کار برده ایم که با استفاده از کلاس StreamReader، یک فایل را خط به خط می خواند :

```
TextReader reader = new StreamReader (filename);
string line;
while (line = reader.ReadLine ()) != null
{
    console.WriteLine (line);
}
reader.close ();
```

وقتی که کارتان با reader تمام شد باید close را فراخوانی کنید تا منابع و رمزنگاری handle فایل را آزاد کنید. با این حال، این مثال یک مساله دارد و آن این است که در برابر Exception ها ایمن نیست. اگر فراخوانی Readline یا Writeline باعث ایجاد یک Exception شود close فراخوانی نخواهد شد.

نکته!

Exception ها معمولا خطاهایی هستند که در دستورات رخ می دهد ولی همه Exception ها لزوما خطا نیستند. در فصل بعد راجب Exception ها به طور کامل صحبت خواهیم کرد.

آزاد سازی ایمن در برابر Exception

یک روش برای اینکه مطمئن شوید که متد آزاد سازی، صرف نظر از اینکه آیا یک Exception ایجاد می شود یا خیر، فراخوانی شود، این است که متد آزاد سازی را در یک بلاک finally فراخوانی کنید. در اینجا مثال قبلی را با استفاده از این روش بازنویسی کرده ایم :

```
TextReader reader = new StreamReader (filename);
try
{
    string line;
    while (line = reader.ReadLine ()) != null)
    {
        console.WriteLine (line);
    }
}
finally
{
    reader.close ();
}
```

استفاده از بلاک finally چند عیب هم دارد :

۱- اگر مجبور شوید که بیش از یک منبع را آزاد کنید، این روش ناکار آمد می شود (در پایان، بلاک های try و finally تو در تو خواهید داشت).

۲- در بعضی موارد، ممکن است مجبور شوید که کد را تغییر دهید. برای مثال، ترتیب تعریف ارجاع به منبع را تغییر دهید، به خاطر داشته باشید که ارجاع را با null مقدار دهی اولیه کنید و در بلاک finally بررسی کنید که ارجاع مساوی null نباشد.

۳- فهم راه حل مشکل است و هر بار باید تکرار شود.

۴- ارجاع به منبع، حتی بعد از بلاک finally هم باقی می ماند. یعنی بعد از اینکه منبعی آزاد شد، ممکن است تصادفاً آن را به کار گیرد. دستور using برای حل تمامی این مسائل طراحی شده است.

دستور using

ساختار دستور using به صورت زیر می باشد :

`using (try variable = initialization) embeddedStatement`

دستور using بالا دقیقاً معادل تبدیل زیر است :

```
{
type variable = initialization
try
{
embeddedStatement
}
finally
{
if (variable != null)
{
((IDisposable)variable).Dispose ();
}
}
}
```

به محدوده بلاک بیرونی توجه کنید. این بدان معنی است که متغیری که در یک دستور using تعریف می کنید در پایان embeddedStatement از محدوده خارج می شود.

اینکه دو کد معادل هستند به این معنی است که در دستور using تعریف می کنید باید از نوعی باشد که یک رابط به نام IDisposable را پیاده سازی کند. این رابط در یک فضای نام به نام System قرار دارد و فقط یک متد به نام Dispose دارد :

```
namespace System
{
interface IDisposable
{
void Dispose ();
}
}
```

می توان از دستور using به عنوان یک روش مستحکم و ایمن در برابر Exception استفاده کرد تا مطمئن شوید که منابع همیشه آزاد می شوند. فقط باید مطمئن شوید که :

- ۱- کلاسی که شامل متد آزاد سازی (برای مثال، close در TextReader) است، IDisposable را پیاده سازی می کند همانطور که TextReader این کار را انجام می دهد.
 - ۲- کلاس، Dispose را پیاده سازی می کند تا متد آزاد سازی را فراخوانی کند (مثلا TextReader.Dispose، TextReader.close را فراخوانی می کند).
- برای مثال به دستورات زیر توجه کنید.

```
abstract class TextReader : IDisposable
{
    :
    .
    public virtual void Dispose ()
    {
        // calls close
    }
    public virtual void close ()
    {
        :
        .
    }
}
```

این بهترین راهی است که مطمئن شوید همیشه close مربوط به TextReader را فراخوانی می کند :

```
using (TextReader reader = new StreamReader (filename))
{
    string line;
    while ((line = reader.ReadLine () ) != null)
    {
        console.WriteLine (line);
    }
}
```

این روش تمام مسائلی که در روش try و finally وجود داشت را حل می کند. نمی توانید بعد از اتمام دستور using، از متغیری که در دستور using تعریف کرده اید استفاده کنید، زیرا این متغیر دیگر در محدوده خود نیست. اگر از آن استفاده کنید یک خطای زمان کامپایل دریافت خواهید کرد.

فراخوانی یک متد آزاد سازی از داخل یک مخرب

یکی از معایب متد آزاد سازی این است که شما به عنوان برنامه نویس باید آنها را فراخوانی کنید و معمولاً برنامه نویسان فراموش می کنند که مواردی مثل این را انجام دهند. یک توازن بین استفاده از مخرب و فراخوانی متد آزاد سازی وجود دارد. مخرب همیشه فراخوانی می شود ولی شما نمی دانید چه زمانی، در حالی که دقیقاً می دانید که متد آزاد سازی چه زمانی اجرا می شود. با این حال، می توان مطمئن شد که متد آزاد سازی همیشه فراخوانی می شود. برای این کار باید متد آزاد سازی را از داخل یک مخرب فراخوانی کرد. ممکن است فراموش کنید که متد آزاد سازی را فراخوانی کنید، ولی حداقل می توانید مطمئن باشید که این متد فراخوانی می شود، حتی اگر این فراخوانی در هنگام بسته شدن برنامه باشد. مثال زیر می تواند مفاهیم بالا را کاملاً توضیح دهد.

قبل از دستورات بهتر است به نکات زیر توجه کنید.

- ۱- این کلاس، IDisposable را پیاده سازی می کند.
- ۲- متد public Dispose می تواند در هر زمانی فراخوانی شود.
- ۳- متد Dispose می تواند بدون ایجاد هیچ مشکلی، چندین بار فراخوانی شود.
- ۴- مخرب این کلاس، Dispose را فراخوانی می کند.
- ۵- متد Dispose، GC.SuppressFinalize را فراخوانی می کند تا موجب شود که gc بیهوده، مخرب را فراخوانی نکند. این در حقیقت، اختیاری است. ممکن است فراخوانی GC.SuppressFinalize از فراخوانی مخرب توسط gc بیشتر طول بکشد.
- ۶- تمام متدهای معمولی کلاس بررسی می کنند که آیا شیء قبلاً آزاد شده است یا خیر. اگر آزاد شده باشد، یک Exception ایجاد خواهد شد.

```
class Example : IDisposable
{
    ...
    ~Example ()
    {
        Dispose ();
    }
    public virtual void Dispose ()
    {
        if (!disposed)
        {
            try
            {
                //release scarce resource here
            }
            //release scarce resource here
        }
        finally
        {
            disposed = true;
            GC.SuppressFinalize (this)
        }
    }
    public void SomeBehavior ()
    {
        checkifDisposed ();
    }
    ...
    ...
    private void checkifDisposed ()
    {
        if (disposed)
        {
            throw new objectDisposedException ("Example");
        }
    }
    private Resource scarce;
    private bool disposed = false;
}
```

مفهوم وراثت (Inheritance)

در دنیای واقعی، وراثت به این معناست که هر فرزندی خواص عمومی را از والد خود به ارث می گیرد، مثل رنگ چشم یا رنگ مو یا چهره ظاهری یا رفتارهای غریزی و ...، در دنیای نرم افزار نیز وراثت به همین معناست. یعنی کلاس جدید را طوری تعریف می کنیم که اعضای کلاس دیگر را به کار بگیرد.

وارثت یکی از مباحث پیشرفته و نیز کاربردی برنامه نویسی شیء گرا محسوب می شود. در

چارچوب Net. از وراثت استفاده زیادی شده است و حتی خود شما تاکنون کلاس هایی ایجاد کرده اید که از کلاس های دیگر ارث برده اند. هر Form ویندوزی که در برنامه های خود ایجاد می کنید، در حقیقت یک کلاس جدید است که بسیاری از اطلاعات خود را از کلاس مربوط به یک Form خالی به ارث می برد.

وراثت برای ایجاد اشیایی به کار برده می شود که تمام اعضای یک شیء دیگر را داشته باشد و علاوه بر آنها، شامل چندین عضو جدید برای خودش باشد. هدف اصلی وراثت این است که بتوانید کارائی های یک کلاس را، بدون اینکه بدانید آن کلاس به صورت درونی چگونه کار می کند، افزایش دهید. به عبارت دیگر با استفاده از وراثت می توانید اشیایی را بر پایه اشیای دیگر که توسط برنامه نویسان دیگری نوشته شده است ایجاد کنید، بدون اینکه بدانید برنامه نویسان اصلی چگونه آن شیء را ایجاد کرده اند.

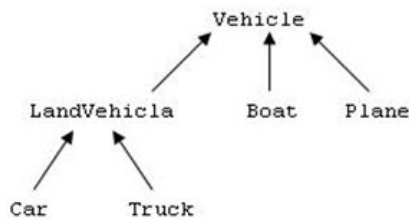
وراثت، امکان استفاده ی مجدد از کلاس ها را فراهم می کند که یک اصل مهم در موضوع مهندسی نرم افزار است. نکته مهم تر از استفاده مجدد، کاهش هزینه برنامه نویسی و تسریع در تولید نرم افزار است.

همانطور که گفته شد در وراثت، کلاس رفتار یا صفاتی را از کلاس دیگر به ارث می برد. کلاسی که موجود است، کلاس پایه، کلاس مافوق، یا کلاس پدر نامیده می شود. و کلاسی که از کلاس پایه ایجاد می شود، کلاس مشتق، کلاس فرعی، یا کلاس فرزند می گویند. هر کلاس مشتق می تواند فقط یک کلاس پایه داشته باشد ولی هر کلاس پایه می تواند چندین کلاس مشتق داشته باشد.

تمام کلاس های موجود در زبان C# صفات و رفتار های خود را از کلاس Object به ارث می برند. کلاس Object در واقع یک کلاس کلی است و رفتار ها و صفاتی را تعریف می کند که توسط سایر کلاس های موجود در کتابخانه زبان C# به ارث برده می شوند.

به طور مثال، فرض کنید یک کلاس پایه به نام Vehicle برای وسایل نقلیه داریم. وسیله نقلیه می تواند ماشین، تراکتور، قایق و هواپیما را شامل شود که ممکن است هر یک از آنها احتیاج به اطلاعات یا توابع اضافی داشته باشند. بنابراین می توان برای هر کدام کلاس های فرعی را

تعریف کرد که خواص کلاس Vehicle را به ارث می برند علاوه براین که دارای فیلدهای جدید دیگری هم هستند.



اگر کلاس ویژگی های تنها یک کلاس را به ارث ببرد وارث منفرد (single inheritance) و اگر از چند کلاس به ارث گرفته شود وارث چندگانه (multiple inheritance) نامیده می شود. توجه داشته باشید که در زبان C# وارث چندگانه وجود ندارد، یعنی کلاسی بیش از یک کلاس به طور مستقیم مشتق شود.

برای اعلان ارث بری یک کلاس از کلاس دیگر در هنگام تعریف کلاس به صورت زیر انجام می شود :

نام کلاس پدر : نام کلاس فرزند class سطح دسترسی

```
{
```

تعریف کلاس فرزند شامل اضافه کردن اعضای جدید یا احیاناً تغییر اعضای قدیمی

```
}
```

در تعریف زیر کلاس child از کلاس parent ارث برده است.

```
public class : parent
```

```
{
```

```
...
```

```
}
```

در وراثت اعضایی با سطح دستیابی private در کلاس پدر مستقیماً توسط کلاس های فرزند قابل دستیابی نیستند اما اعضای اختصاصی کلاس پدر به ارث برده می شوند. بقیه اعضای کلاس پدر وقتی به عنوان اعضای کلاس فرزند در می آیند همان سطح دستیابی اصلی خود را حفظ می کنند. به عنوان مثال اعضای public کلاس پدر وقتی به عنوان عضو کلاس فرزند می شوند همان سطح دستیابی public خود را حفظ خواهند کرد. اعضای کلاس protected کلاس پدر نیز وقتی به عنوان اعضای کلاس فرزند در می آیند سطح دستیابی

protected را حفظ می کنند. با استفاده از این اعضای ارثی کلاس پدر، کلاس فرزند می تواند اعضای private کلاس پدر را دستکاری کند اگر این اعضاء چنین قابلیت را در کلاس پدر فراهم کنند.

برای فراخوانی یکی از متد های کلاس پدر در یکی از متد های کلاس فرزند، از روش زیر استفاده می شود :

متد فرزند

```
{  
    ;(ارسال پارامتر ها در صورت وجود) نام متد پدر . base  
}
```

کلمه کلیدی base، اعلان می کند که متد مربوط از کلاس پدر فراخوانی شده است و در صورت عدم ذکر آن، متد مذکور از خود کلاس فرزند که متد را به ارث برده است فراخوانی می شود. ولی امکان دارد این متد در کلاس فرزند، تغییر داده شده باشد و دیگر همان متد اولیه کلاس پدر نباشد.

برنامه نویسان C# باید به این نکته دقت کنند به طور صریح نمی توانید مشخص کنید که آیا وراثت، public، private، یا protected است. وراثت در C# همیشه به طور ضمنی public است.

وراثت در برنامه نویسی به طبقه بندی مربوط می شود و یک رابطه بین کلاس ها است. اگر کتاب علوم تجربی دوران ابتدایی را به یاد داشته باشید، در این کتاب در مورد پستانداران مطالبی را آموختید که مثلا اسب یک پستاندار است. در زبان C# می توانید این مورد را با ایجاد دو کلاس مدل سازی کنید. یکی با عنوان mammal (پستانداران) و دیگری به نام horse (اسب). سپس تعریف کنید که horse از mammal ارث می برد. وراثت این رابطه را مدل سازی می کند و مشخص می کند که تمام horse ها، mammal هستند.

توجه! وراثت یک رابطه بین اشیاء نیست، بلکه یک رابطه بین کلاس هاست. رابطه همیشه در سطح کلاس وجود دارد، و تضمین می کند که این رابطه همیشه برای تمام اشیاء آن کلاس

برقرار است. در واقع این مطلب را می توان درک کرد که انعطاف پذیری کمی در وراثت وجود دارد.

در دستورات زیر کلاس derived از کلاس base ارث می برد. یک کلاس می تواند حداکثر از یک کلاس ارث ببرد و نمی تواند از دو یا چند کلاس ارث ببرد.

```
class derived : base
{
...
}
```

همانطور که گفتیم کلاس System.Object، کلاس ریشه برای تمام کلاس ها است. به عبارت دیگر، همه کلاس ها به صورت ضمنی از این کلاس ارث می برند. اگر یک کلاس به صورت زیر تعریف کنید :

```
class Token
{
public Token (string name)
{
...
}
...
}
```

کامپایلر این کد را به صورت زیر تبدیل می کند. اگر بخواهید خودتان هم می توانید این کد را به صورت زیر بنویسید، ولی بهتر است این کار را انجام ندهید.

```
class Token : System.Object
{
public Token (string name)
{
...
}
...
}
```

فراخوانی سازنده های کلاس پدر

سازنده کلاس مشتق شده یا فرزند باید سازنده کلاس پایه خود را فراخوانی کند. برای این کار از کلمه کلیدی base استفاده می شود. هیچ ابهامی در استفاده از این کلمه کلیدی وجود ندارد، زیرا هر کلاس، حداکثر یک کلاس پایه یا پدر دارد.

در صورتی که در سازنده پایه، یک سازنده به صورت پیش فرض و بدون پارامتر داشته باشیم در صورت عدم تعریف سازنده در کلاس مشتق، این سازنده به ارث برده می شود. در غیر این صورت برای داشتن یک سازنده پیش فرض باید یک سازنده بدون پارامتر در کلاس مشتق شده تعریف کنیم یعنی سازنده بدون پارامتر کلاس پایه، تنها در حالتی به ارث برده می شود که کلاس مشتق شده هیچ سازنده ای با پارامتر یا بدون پارامتر در کلاس خود تعریف نکرده باشد.

در صورت عدم تعریف سازنده در کلاس مشتق شده، تنها سازنده پیش فرض بدون پارامتر کلاس پایه به ارث برده می شود و سایر سازنده ها به ارث برده نمی شوند. در سازنده های کلاس مشتق شده، اگر سازنده پیش فرض بدون پارامتر در کلاس پایه وجود داشته باشد به طور اتوماتیک فراخوانی خواهد شد و سپس دستورات سازنده مشتق شده اجرا می شود. ولی برای فراخوانی سازنده های غیر پیش فرض از کلاس پایه در سازنده های کلاس مشتق شده، سازنده کلاس مشتق شده به صورت زیر تعریف می شود :

(تعریف پارامتر های سازنده مشتق شده یا فرزند) نام سازنده مشتق شده سطح دسترسی

(ارسال مقادیر به پارامتر های سازنده مورد نظر از کلاس پایه) : base

{

...

}

مقادیری که به سازنده پایه ارسال می شوند معمولا از پارامتر های سازنده فرزند هستند. یعنی پارامتری که مربوط به سازنده مشتق شده است به سازنده پایه نیز ارسال می شود. به مثال زیر توجه کنید.

```
class IdentifierToken : Token
{
public IdentifierToken (string name)
: base (name) //calls Token (name)
{
...
}
...
}
```

اگر صریحا سازنده کلاس پایه را در سازنده کلاس مشتق شده فراخوانی نکنید، خود کامپایلر

سعی می کند تا سازنده پیش فرض کلاس پایه را در کلاس مشتق شده فراخوانی کند. کامپایلر کد زیر را :

```
class Token
{
public Token (string name)
{
...
}
...
}
```

به صورت زیر ترجمه خواهد کرد.

```
class Token : System.Object
{
public Token (string name)
: base ()
{
...
}
...
}
```

این کد کار می کند زیرا System.Object یک سازنده پیش فرض و public دارد. البته همه کلاس ها یک سازنده پیش فرض و public ندارند که در این صورت اگر سازنده کلاس پایه را فراخوانی نکنید، یک خطای زمان کامپایل دریافت خواهید کرد. برای مثال :

```
class IdentifierToken : Token
{
public IdentifierToken (string name)
//error, base class Token does not have
// a public default constructor
{
...
}
...
}
```

مخرب ها در کلاس های مشتق شده

وقتی زباله روب، شیء کلاس مشتق را از حافظه حذف می کند مخرب آن شیء را فراخوانی می کند. به این ترتیب زنجیره ای از فراخوانی های مخرب ایجاد می شود که در آن، مخرب کلاس مشتق و مخرب های مستقیم و غیر مستقیم کلاس پایه به ترتیب عکس اجرای سازنده

ها، اجرا می شود. اجرای مخرب باید تمام منابعی را که شیء در اختیار دارد آزاد کند تا زباله روب حافظه آن را پس بگیرد. وقتی زباله روب مخرب شیء کلاس مشتق را فراخوانی می کند، مخرب کارش را انجام می دهد سپس مخرب کلاس پایه را فراخوانی می کند. این فرایند ادامه می یابد تا مخرب کلاس Object فراخوانی شود.

زبان C#، مخرب ها را با استفاده از متد Finalize کلاس object پیاده سازی می کند. هنگام ترجمه، تعریف کلاسی که حاوی مخرب است، کامپایلر، تعریف مخرب را به متد Finalize ترجمه می کند که وظایف مخرب را انجام می دهد و سپس متد Finalize کلاس پایه را فراخوانی می کند که آخرین دستور در متد Finalize کلاس مشتق می باشد. دقیقاً نمی توان مشخص کرد که فراخوانی مخرب چه زمانی صورت می گیرد. زیرا نمی توان تعیین کرد که زباله روبی چه زمانی صورت می گیرد. به هر حال، با تعریف مخرب می توان کدی را تعریف کرد که قبل از اجرای زباله روب، اجرا شود.

متد های new

اگر یک کلاس پایه و یک کلاس مشتق شده از آن، دو متد با مشخصه تعریف یکسانی داشته باشند (مشخصه تعریف یک متد، اسم متد و تعداد و نوع پارامتر های آن است)، این دو متد به هم مرتبط نیستند. برای مثال، اگر مثال زیر را کامپایل کنید، کامپایلر یک پیام اخطار به شما می دهد و می گوید که `Token.Name` , `IdentifierToken.Name()` را مخفی کرده است :

```
class Token
{
...
public string Name ()
{
...
}
}
class IdentifierToken : Token
{
...
public string ()
{
...
}
}
```

این اخطار به شما یاد آوری می کند که از آنجایی که دو متد، مشخصه یکسانی دارند و به هم مرتبط نیستند، تعریف Name در IdentifierToken، متد Name در Token را مخفی می کند (می پوشاند). اکثر مواقع، این تشابه اسمی، باعث سردرگمی می شود و باید اسم آنها را تغییر دهید. ولی اگر بخواهید که دو متد، مشخصه تعریف یکسانی داشته باشند، می توانید با استفاده از کلمه کلیدی new، کاری کنید که کامپایلر دیگر پیغام اخطار ندهد :

```
class Token
{
...
public string Name ()
{
...
}
}
class IdentifierToken : Token
{
...
new public string ()
{
...
}
}
```

استفاده از کلمه کلیدی new به این صورت، این واقعیت که دو متد هنوز کاملاً به هم ربطی ندارند و اینکه هنوز مخفی سازی انجام می شود را تغییر نمی دهد و فقط باعث می شود که دیگر پیغام اخطار، ظاهر نشود. در واقع، منظور از کلمه کلیدی new این است : (من یک برنامه نویس حرفه ای هستم و می دانم چه کاری انجام می دهم).

برای اینکه کاملاً درک کنید مخفی سازی چه مفهومی دارد، مثال زیر را در نظر بگیرید و فکر کنید که کدام یک از دو متد فراخوانی خواهد شد :

```
static void Method (Token t)
{
Console.WriteLine (t.Name ());
}
static void Main ()
{
IdentifierToken variable = new IdentifierToken ("variable");
Method (variable);
}
```

در این مثال، Main یک متغیر از نوع IdentifierToken را تعریف می کند و آن را با یک شیء از نوع IdentifierToken مقدار دهی اولیه می کند. Method پارامتر خود را از نوع Token تعریف می کند، نه از نوع IdentifierToken. این کار مجاز است، زیرا هر IdentifierToken یک Token است و بنابراین می تواند جایگزین Token شود. سپس بدنه Method، متد Name مربوط به پارامتر را فراخوانی می کند. وقتی به این کد نگاه کنید متوجه خواهید شد که در این مورد خاص، پارامتر، اگر چه از نوع Token تعریف شده است، در واقع به یک شیء از نوع IdentifierToken اشاره دارد. پس ممکن است انتظار داشته باشید که فراخوانی Name به IdentifierToken.Name تبدیل شود. ولی این فراخوانی به Token.Name تبدیل می شود. زیرا دو متد Name، مربوط به هم ربطی ندارند و همدیگر را مخفی می کنند (می پوشانند). شاید این رفتاری نباشد که شما می خواهید. خوشبختانه، یک راه وجود دارد که فراخوانی، آن طوری که شما انتظار دارید عمل کند.

مثال ۳۰ - ۵: برنامه زیر نحوه برقراری ارتباط با کلاس های پدر و فرزند را نشان می دهد.

```
class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }
    public void print()
    {
        Console.WriteLine("I am a Parent Class.");
    }
}
public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Construcotr.");
    }
    public void Main()
    {
        ChildClass child = new ChildClass();
        child.print();
    }
}
```

اکنون در متد اصلی کلاس program.cs دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    ChildClass child = new ChildClass();
    child.print();
}
```

خروجی برنامه بالا به صورت زیر خواهد شد.

Parent Constructor.

Child Constructor.

I'm a Parent Class.

توضیح : در دستورات بالا دو کلاس وجود دارد. کلاس اولی ParentClass و کلاس دومی ChildClass می باشد. کاری که می خواهیم در اینجا انجام دهیم این است که زیر کلاسی ایجاد کنیم که با استفاده از دستورات موجود در ParentClass عمل نماید. برای این منظور ابتدا باید در اعلان ChildClass مشخص کنیم که این کلاس می خواهد از کلاس ParentClass ارث بری داشته باشد. این عمل با اعلان : `public class ChildClass` روی می دهد. کلاس پایه با قرار دادن : بعد از نام کلاس مشتق شده معین می شود.

زبان C# فقط از ارث بری یگانه پشتیبانی می کند. البته این زبان امکان مشتق شدن از رابط های چندگانه را نوعاً فراهم می آورد. این بدان معناست که یک کلاس C# می تواند از یک کلاس دیگر و هر تعداد رابط مشتق شود. به خاطر وجود System.Object به عنوان یک نوع مبنای رایج، هر کلاس C# (به جز Object) دقیقاً دارای یک کلاس مبناست و ممکن است در کل دارای هر تعداد رابط مبنا باشد.

ChildClass دقیقاً توانایی های ParentClass را دارد. از این رو می توان گفت ChildClass یک ParentClass است. (ChildClass IS a ParentClass) ChildClass دارای متد `print()` مربوط به خود نیست و از متد کلاس ParentClass استفاده می کند.

کلاس های پایه به طور خودکار، قبل از کلاس های مشتق شده نمونه ای از روی آنها ایجاد می کند. به خروجی برنامه توجه کنید. سازنده ParentClass قبل از سازنده ChildClass اجرا می گردد.

نکته!

بعضی از زبان ها مانند C++ از وراثتی موسوم به وراثت چندگانه پشتیبانی می کنند که در آن یک کلاس از بیش از یک کلاس دیگر مشتق می شود. مزایای استفاده از وراثت چندگانه قابل بحث می باشد. از طرفی، شکی نیست که امکان استفاده از وراثت چندگانه برای نوشتن کد های بسیار پیچیده مانند کتابخانه ATL مربوط به زبان C++ وجود دارد. از طرف دیگر، اغلب کدی که از وراثت چندگانه پیاده سازی استفاده می کند، به سختی قابل فهم و اشکال زدایی است.

نکته!

در حقیقت ATL همان (Active Template Library) می باشد که شامل یکسری الگو بر پایه کلاس های پایه ای میکروسافت است که شما به آسانی و با سرعت می توانید اشیای COM (Component Object Model) را خلق نمایید پشتیبانی COM در زبان C++ این اجازه را به برنامه نویسان می دهد که اشیاء متنوع COM و کنترل های Activex را ایجاد نمایند.

مثال ۳۱ – ۵: برنامه زیر نحوه برقراری ارتباط کلاس های مشتق شده با کلاس پایه را نشان می دهد.

حل : ابتدا یک پروژه از نوع Console ساخته و یک کلاس داخلی با عنوان Parent تعریف کرده و دستورات زیر را در آن تایپ کنید.

```
public class Parent
{
    string parentString;
    public Parent()
    {
        Console.WriteLine("Parent Construcot");
    }
    public Parent(string myString)
    {
        parentString = myString;
        Console.WriteLine(parentString);
    }
    public void print()
    {
        Console.WriteLine("I am a Parent Class.");
    }
}
```

```

public class Child : Parent
{
    public Child() : base ("From Derived")
    {
        Console.WriteLine("Child Constructor.");
    }
    public new void print()
    {
        base.print();
        Console.WriteLine("I am a Child Class");
    }
}

```

در متد اصلی کلاس program.cd دستورات زیر را نوشته و برنامه را اجرا کنید.

```

static void Main(string[] args)
{
    Child child = new Child();
    child.print();
    ((Parent)child).print();
}

```

اکنون برنامه را اجرا کنید تا خروجی را مشاهده کنید.

```

From Derived
Child Constructor.
I'm a Parent Class.
I'm a Child Class.
I'm a Parent Class.

```

توضیح : کلاس های مشتق شده در طول ایجاد نمونه می توانند با کلاس پایه خود ارتباط برقرار کنند. این برنامه چگونگی انجام این عمل را در سازنده ChildClass نشان می دهد. استفاده از : و کلمه کلیدی base باعث فراخوانی سازنده کلاس پایه به همراه لیست پارامتر هایش می شود. اولین سطر خروجی، فراخوانی سازنده کلاس پایه را به همراه رشته From Derived نشان می دهد.

ممکن است حالتی رخ دهد که نیاز داشته باشید تا متد موجود در کلاس پایه را خود پیاده سازی نمایید. کلاس Child این عمل را با اعلان متد print() مربوط به خود انجام می دهد. متد print() مربوط به کلاس Child، متد print() کلاس Parent را پنهان می کند. نتیجه کار

آن است که متد `print()` کلاس `Parent` تا زمانی که عمل خاصی را انجام ندهیم قابل فراخوانی نمی باشد.

درون متد `print()` کلاس `Child`، صریحا متد `print()` کلاس `Parent` را فراخوانی کرده ایم. این عمل با استفاده از کلمه کلیدی `base` قبل از نام متد انجام گرفته است. با استفاده از کلمه کلیدی `base` می توان به هر یک از اعضای `public` و `protected` کلاس پایه دسترسی داشت. خروجی مربوط به متد `print()` کلاس `Child` در سطرهای سوم و چهارم خروجی دیده می شوند.

روش دیگر دسترسی به اعضای کلاس پایه، استفاده از `Casting` صریح است. توجه داشته باشید که کلاس مشتق شده نوع خاصی از کلاس پایه اش می باشد. این مسئله باعث می شود تا بتوان کلاس مشتق شده را مورد عمل `Casting` قرار داد و آن را نمونه ای از کلاس پایه اش قرار داد. آخرین خط خروجی نشان می دهد که متد `print()` کلاس `Parent` اجرا شده است.

به وجود کلمه کلیدی `new` در متد `print()` کلاس `Child` توجه کنید. این عمل باعث می شود تا متد `print()` کلاس `Child` متد `print()` کلاس پایه اش را پنهان نماید. در صورتی که از کلمه کلیدی `new` استفاده نشود، کامپایلر پیغام خطای را تولید می کند تا توجه شما را به این مسئله جلب کند.

مثال ۳۲ - ۵: برنامه زیر نحوه کار و استفاده از کلاس پایه و مشتق شده را نشان می دهد. توضیحات به صورت `Comment` در داخل برنامه ذکر شده است. کاربر با کلیک روی دکمه `button` این پیغام را می تواند مشاهده کند.

حل: ابتدا یک پروژه جدید از نوع ویندوزی ساخته و سپس `Form` آن را مطابق پنجره تصویر ۳۸ - ۵ طراحی کنید.

یک کلاس داخلی با نام `Person` ایجاد کرده، سپس دستورات زیر را بنویسید.



تصویر ۳۸ - ۵

```
public class Person
{
    //classic way of defining a property
    private string _FirstName;
    public string FirstName
    {
        get
        {
            return _FirstName;
        }
        set
        {
            _FirstName = value;
        }
    }
    //simple/compact way of defining a property with get/set operations
    //internally/implicitly maintains a private variable
    public string LastName
    {
        get;
        set;
    }
}
```

دوباره یک کلاس داخلی با نام Employee ایجاد کرده و دستورات زیر را در آن بنویسید.

```
public class Employee : Person
{
    public int EmployeeID
    {
        get;
        set;
    }
    //the "FirstName" and "LastName" properties in "Person" class
    //are automatically carried into this class. We need not redefine them.
}
```

به Form^۱ برگشته و دستورات زیر را در رویداد Click دکمه Click بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    Person p = new Person();
    p.LastName = "David";
    p.FirstName = "Chat";
    Employee oEmp = new Employee();
    Employee emp = new Employee();
    emp.EmployeeID = 1001;
    oEmp.FirstName = "Win";
    oEmp.LastName = "Others";
    MessageBox.Show(string.Format("Name = '{0} {1}'",
        oEmp.FirstName, oEmp.LastName));
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

آشنایی با عملگر is

عملگر is به شما اجازه می دهد تا سازگار بودن یک شیء را با یک نوع بررسی کنید. به عنوان مثال برای بررسی سازگار بودن یک متغیر با نوع object. عبارت سازگار بودن به معنای یکسان بودن یک شیء با شیء مذکور یا مشتق شدن از آن شیء است.

```
int i = 10;
if (i is object)
{
    Console.WriteLine("i is an object");
}
```

نوع int مانند انواع داده ای در زبان C# از object مشتق می شود، بنابراین عبارت i is object دارای مقدار true بوده و پیغامی را نمایش می دهد.

آشنایی با عملگر as

عملگر as برای انجام تبدیلات صریح از نوع مرجع است. در صورتی که نوع تبدیل شونده با نوع مشخص شده سازگار باشد، تبدیل با موفقیت انجام خواهد شد. در صورت عدم سازگاری بین این دو، عملگر مقدار null را باز خواهد گرداند. همانطور که در کد زیر نشان داده شده است، در صورتی که مرجع object واقعا به یک نمونه رشته ای ارجاع نکند، تلاش برای تبدیل مرجع یک object به یک رشته، مقدار null را بر می گرداند:

```
object o1 = "Some String";
object o2 = "5";
string s1 = o1 as string; // S1 = "Some string"
string s2 = o2 as string; // S2 = null
```

عملگر as به شما اجازه می دهد تا یک تبدیل نوع امن را در یک مرحله بدون نیاز به تست نوع با استفاده از is و سپس انجام عمل تبدیل انجام دهید.

فرآیند کپسوله سازی (Encapsulation) داده ها

برای حصول درک برنامه نویسی شیء گرا در ابتدا باید پایه ای محکم و استوار بنا کنید تا با تکیه بر این دانش، مهارت خود را گسترش دهید. نخست لازم است که مفاهیم اولیه برنامه نویسی شیء گرا را کامل بشناسید و برای درک کامل آنها تلاش کنید. تنها وقتی خواهید توانست تکنیک شیء گرایی را بدرستی در پروژه های خود اعمال کنید که درک کاملی از اصول و مبانی آن داشته باشید.

این مقدمه ما را به سه مفهوم اساسی می رساند که بایستی در مورد زبان های شیء گرا صادق باشد. این سه مفهوم غالباً تحت عنوان سه رکن برنامه نویسی شیء گرا محسوب می شوند این سه رکن عبارتند از :

۱- ارث بری

۲- کپسوله سازی

۳- چند ریختی

از آنجایی که برنامه نویسی شیء گرا براساس این سه مفهوم بنا شده است، این سه رکن شبیه، ستونی است که متشکل از چند بلوک است یعنی کافی است که بلوک پائینی را پائین بکشید تا کل ستون فرو بریزد.

کپسوله سازی به ما این امکان را می دهد که به جای برخورد با برنامه به صورت موجودیتی منفرد و یک تکه آن را به اجزای مستقل کوچکتری تقسیم کنیم، و هر جزء کار خود را مستقل از سایر اجزاء انجام می دهد.

کپسوله سازی یا مخفی کردن جزئیات بخشی از عملیات جهان خارج است و این اجازه را می دهد که اجزای نرم افزاری مستقلی را ایجاد کنیم.

وقتی که کپسوله سازی اعمال شده باشد می توان هر موجودیت نرم افزاری را به صورت یک جعبه سیاه در نظر گرفت. هر گاه رابط خارجی جعبه را بشناسیم و بدانیم که چگونه کار می

کند می توان متوجه شد که جعبه به چه صورتی کار می کند. برنامه نویس تنها با جعبه سیاه سر و کار دارد و با آن تبادل اطلاعات می کند و آنچه درون جعبه رخ می دهد اهمیتی ندارد. اگر بتوانید به دقت از کپسوله سازی استفاده کنید. یک شیء، تبدیل به موجودیتی قابل اتصال می شود. برای اینکه شیء ای بتواند از شیء ای استفاده کند فقط لازم است که بداند چگونه رابط عمومی آن را به کار بگیرد. چنین استقلال سه مزیت ارزشمند دارد :

اول اینکه مفهوم استقلال، یعنی از شیء می توان در هر جایی و به صورت مکرر استفاده کرد. وقتی بتوانید اشیای خود را کپسوله کنید اشیاء به هیچ برنامه خاص مرتبط یا متصل نیستند، ولی در عوض می توانید آنها را هر جا که استفاده از آنها قابل توجیه باشد استفاده نمایید. برای استفاده از شیء در هر جای دیگری فقط با رابط آن سر و کار دارید.

دوم اینکه کپسوله سازی این امکان را فراهم می آورد که شیء خود را هر چه قدر که لازم است تغییر دهید تا زمانی که رابط را تغییر ندهید تمام تغییرات در شیء برای اشیاء دیگری که از شیء استفاده می کنند غیر قابل دید است.

سوم اینکه کپسوله سازی به شما اجازه می دهد که شیء خود را بروز کنید. معایب و اشکالات آن را بر طرف نمایید بدون آنکه نیازی داشته باشید تا سایر اشیاء موجود در برنامه را تغییر دهید. استفاده از یک شیء کپسوله شده مانع تاثیرات متقابل ناخواسته بین شیء و سایر قسمت های برنامه می شود.

به طور مثال اگر شیء ماشین با شیء راننده تعامل کند، در صورتی که رابط ماشین شامل متد های (حرکت به جلو)، (حرکت به عقب)، و توقف باشد همه این موارد اطلاعاتی است که راننده برای تعامل با ماشین نیاز دارد. ماشین شامل شیء موتور نیز می باشد اما راننده نیازی به شناخت شیء موتور ندارد. تمام اطلاعاتی که راننده در مورد این متد ها دارد این است که می توانند به راحتی فراخوانی شوند و مقادیر ویژه ای را نیز برگردانند. پس اگر شیء موتور تغییر کند تا زمانی که رابط آن به درستی کار کند این امر تفاوتی برای راننده ایجاد نمی کند. مشکلی که ممکن است برای شما ایجاد شود و شما را دچار اشتباه کند، دید اشتباه شما نسبت به کپسوله سازی است یعنی شما قصد داشته باشید کلاسی بسازید که از هر لحاظی کامل

باشد و در هر موقعیتی قابل استفاده باشد. این امر غیر ممکن خواهد بود یعنی شما نمی توانید کلاسی پیاده سازی کنید که تمام متد ها و کار ها و اعمال یک شیء را پیش بینی کرده باشد. ایده ها و نظرات بعضی از برنامه نویسان کلا در مورد برنامه نویسی شیء گرا و استفاده از سه رکن اصلی آن این است که همه چیز و همه کار به وسیله مبحث کپسوله سازی انجام می شود. ولی در واقعیت اینگونه نیست حتی در بعضی مواقع استفاده از کپسوله سازی و کلا مباحث OOP در مواردی می توانند برای ما دردسر ساز هم شوند.

شما تا به اینجا و در بخش های قبلی نیز با مفهوم کپسوله سازی کار کرده و آشنا شده اید ولی نمی دانستید کاری که انجام می دهید کپسوله سازی است.

سوالی که اکثر افراد دارند این است که می گویند مباحث برنامه نویسی شیء گرا کجا خاتمه می یابد و یا اصلا به چه دردی می خورند؟ در دنیای برنامه نویسی امروز، برنامه ها بایستی دارای ویژگی خوانایی، نقص و خطای کم، قابلیت استفاده و تغییر مجدد توسط تیم برنامه نویسی را دارا باشند. و این امور تنها از طریق برنامه نویسی شیء گرا قابل انجام است.

ممکن است شما یک برنامه برای موسسه X تعریف کنید، موسسه Y با اندکی تغییر نیاز به همان برنامه داشته باشد، اینجاست که قواعد شیء گرایی دست شما را در تغییر برنامه باز می گذارد.

هر چند که تا به حال زیاد با مفهوم کپسوله سازی کار کرده و آشنا شده اید ولی می خواهیم در این بخش با یک مثال ساده این مفهوم را جامع تر توضیح دهیم :



تصویر ۳۹ - ۵

حال یک کلاس داخلی به نام MyTime به پروژه خودمان اضافه می کنیم. از اینجاست که می توان ادعا کرد که کپسوله سازی داده ها رعایت شده است. زیر خود تعریف کلاس می تواند

یک نوع کپسوله سازی باشد چون ما اطلاعات را درون یک کلاس و به صورت دسته بندی شده قرار می دهیم.

خوب در کلاس MyTime سه فیلد به نام های Hour، Minute، Second از نوع string و با سطح دسترسی private تعریف می کنیم. پس از این فیلد ها سایر قسمت های برنامه و کلاس های دیگر نمی توانند از این فیلد ها استفاده کنند. امنیت داده ها و مخفی سازی داده ها از کلاس های دیگر در مبحث کپسوله سازی دقیقاً همین است. یک متد با سطح دسترسی public و با نام SetTime از نوع void تعریف می کنیم، با این متد می خواهیم مقدار ساعت روی مقدار فعلی سیستم تنظیم شود. متد دیگری با سطح دسترسی public و از نوع رشته ای و با نام GetTime ایجاد می کنیم. سپس دستورات مورد نظر را مطابق زیر می نویسیم :

```
class MyTime
{
    string hour;
    string minute;
    string second;
    public void SetTime()
    {
        hour = DateTime.Now.Hour.ToString();
        minute = DateTime.Now.Minute.ToString();
        second = DateTime.Now.Second.ToString();
    }
    public string GetTime()
    {
        return hour + ":" + minute + ":" + second;
    }
}
```

به Form۱ برنامه برگشته و در قسمت public partial از کلاس MyTime یک نوع جدید به نام tm ایجاد کنید.

```
public partial class Form1 : Form
{
    MyTime tm = new MyTime();
```

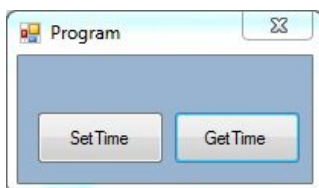
در Form۱ بر روی دکمه SetTime دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    tm.SetTime();
}
```

دوباره در Form۱ بر روی دکمه GetTime دابل کلیک کرده و دستورات زیر را بنویسید.

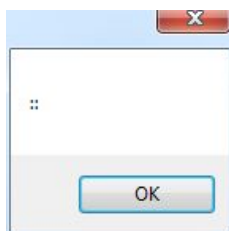
```
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show(tm.GetTime());
}
```

بنابراین اگر برنامه را اجرا کنید پنجره ای مطابقی پنجره تصویر ۴۰ - ۵ نمایش داده می شود.



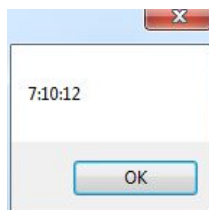
تصویر ۴۰ - ۵

اگر در زمان اجرا ابتدا بر روی دکمه GetTime کلیک کنید یک علامت دو نقطه ساده مانند پنجره تصویر ۴۱ - ۵ نمایش داده می شود زیرا از متد SetTime استفاده نکرده اید.



تصویر ۴۱ - ۵

ولی در صورتی که ابتدا بر روی دکمه SetTime کلیک کنیم و متد SetTime را فراخوانی کنیم و سپس بر روی دکمه GetTime کلیک کنیم زمان جاری سیستم نمایش داده می شود (پنجره تصویر ۴۲ - ۵).



تصویر ۴۲ - ۵

سربار گذاری متد ها در حالت ارث بری

اگر در کلاس مشتق شده، متدی از کلاس پایه به ارث برده شود که همانم یک یا چند متد از کلاس مشتق شده باشد ولی از لحاظ ساختاری (نوع، تعداد و ترتیب پارامتر ها) متفاوت باشد، متد پایه نیز متد سربار شده آن متد ها می شود. به عبارت دیگر مواردی که در مورد سربار گذاری متد ها پیش تر گفتیم در مورد سربار گذاری متد کلاس با متد های ارث برده شده از کلاس پایه نیز شامل می شود.

توجه داشته باشید که سربار گذاری متد های کلاس پایه از این جهت از اتصال متد ها متفاوت است که در سربار گذاری شما متدی همانم ولی با ساختار متفاوت از کلاس پایه تعریف می کنید. ولی در اتصال متد ها شما متدی همانم و مشابه متد پایه تعریف می کنید. به طور مثال در نظر بگیرید که در کلاس پایه متدی با نام sum و با دو پارامتر از نوع صحیح و اعشاری به صورت زیر تعریف کرده اید :

```
public float sum (int a , float b)
{
    return a + b;
}
```

حال در کلاس مشتق شده دو متد Sum دیگر تعریف می کنیم اولی با دو پارامتر از نوع int و دومی با دو پارامتر از نوع اعشاری.

```
public int sum (int a , int b)
{
    return a + b;
}
```

```
public float sum (float c , float d)
{
    return c + d;
}
```

در دستورات بالا کلاس مشتق شده دو متد sum تعریف کرده است که علاوه بر اینکه یکدیگر را سربار گذاری کرده اند، متد sum ارث برده شده از کلاس پایه را نیز سربار گذاری کرده اند.

با این توضیحات، شیء ای که از کلاس مشتق شده ایجاد می شود، سه متد sum متفاوت را در اختیار خواهد داشت که بر حسب نوع پارامتر ها می تواند متد مورد نظر را انتخاب کند.

سازنده ها و مخرب ها در ارث بری

در زبان های شیء گرا هنگامی که شیء ای از کلاس مشتق شده ایجاد می شود. ابتدا سازنده کلاس پایه اجرا می شود و سپس سازنده کلاس مشتق شده. چنانچه سازنده کلاس پایه پارامتر های ورودی داشته باشد می بایست این پارامتر ها را به نحوی برای آن مرتبط کنیم. به این صورت که در متد سازنده کلاس مشتق شده این پارامتر ها را دریافت کنیم و آنها به متد سازنده کلاس پایه ارسال کنیم.

روند اجرای مخرب ها به صورت عکس سازنده هاست یعنی ابتدا مخرب کلاس مشتق شده و سپس مخرب کلاس پایه اجرا می شود.

این مفاهیم را بهتر است که در محیط ویژوال استودیو و تحت یک پروژه دنبال کنیم یک پروژه از نوع ویندوزی ایجاد کرده و سپس یک کلاس جدید به نام fatherclass در آن ایجاد می کنیم.

چون می خواهیم از پیام استفاده کنیم بهتر است در بخش using ها دستور using System.Windows.Forms; را به پروژه خودمان اضافه کنیم. حال در کلاس fatherclass دستورات زیر را بنویسید.

```
class fatherclass
{
    public fatherclass()
    {
        MessageBox.Show("Constructor of base is run.");
    }
}
class child : fatherclass
{
    public child()
    {
        MessageBox.Show("Constructor of derived is run.");
    }
}
```

با پیغام Constructor of base is run. متوجه اجرای سازنده کلاس پایه خواهیم شد. و با پیام Constructor of derived is run. متوجه زمان اجرا شدن کلاس child خواهیم شد.

به Form۱ برنامه برگشته و آن را مطابق پنجره تصویر ۴۳ - ۵ طراحی کنید.



تصویر ۴۳ - ۵

بر روی دکمه Click دابل کلیک کرده و در رویداد Click آن دستورات زیر را نوشته و برنامه را اجرا کنید.

```
private void button1_Click(object sender, EventArgs e)
{
    child ch = new child();
}
```

در صورتی که برنامه را اجرا کنید ابتدا سازنده کلاس پایه اجرا می شود یعنی ابتدا پیام Constructor of base is run. و سپس پیام Constructor of derived is run. اجرا خواهد شد.

مساله ای که وجود دارد روند اجرای مخرب و سازنده های پارامتر دار کلاس پایه است. اجازه دهید تا روند اجرای مخرب های را با هم دنبال کنیم و بعد از آن به سراغ سازنده های پارامتر دار خواهیم رفت. در کلاس fatherclass و child دستورات زیر را برای مخرب می نویسیم.

```
class fatherclass
{
    public fatherclass()
    {
        MessageBox.Show("Constructor of base is run.");
    }
    ~fatherclass()
    {
        MessageBox.Show("Destructor of base is run.");
    }
}
class child : fatherclass
{
    public child()
    {
        MessageBox.Show("Constructor of derived is run.");
    }
    ~child()
    {
        MessageBox.Show("Destructor of derived is run.");
    }
}
```

حال اگر برنامه را اجرا کنید با کلیک روی دکمه اول سازنده کلاس پایه، سپس سازنده کلاس مشتق شده اجرا خواهد شد. به محض اینکه Form برنامه را با کلیک بر روی دکمه Close در نوار عنوان ببندید مخرب ها اجرا خواهند شد. ابتدا مخرب کلاس مشتق شده و سپس مخرب کلاس پایه اجرا خواهد شد.

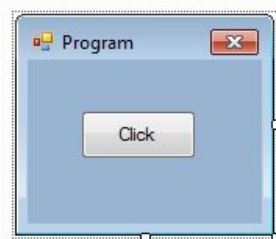
اگر متد سازنده ما دارای پارامتر های ورودی باشد و کلاس ما نیز کلاس پایه باشد چه اتفاقی رخ می دهد؟ بایستی به نحوی مساله پارامتر های ورودی را در هنگام ارث بری تعمیم کنیم و به نحوی از پارامتر های ورودی استفاده کنیم. برای این منظور یک پروژه جدید ویندوزی ساخته و یک کلاس داخلی به نام Person در آن ایجاد می کنیم. این کلاس دارای تعدادی فیلد و یک متد سازنده به صورت زیر است :

```
class Person
{
    protected string name;
    protected string family;
    public Person(string name_t, string family_t)
    {
        name = name_t;
        family = family_t;
    }
}
```

کلاس جدیدی به نام student ایجاد می کنیم که از کلاس Person ارث بری می کند. حال بایستی به نحوی مساله پارامتر های سازنده کلاس پایه را حل کنیم. چون موقع اجرا سازنده کلاس پایه اجرا می شود و بایستی این دو پارامتر را مقدار دهی کنیم تا با پیام خطا مواجه نشویم. بایستی در سازنده کلاس مشتق شده خودتان به تعداد پارامتر های ورودی سازنده کلاس پایه پارامتر ورودی یا متغیر تعریف کنید. به طور مثال ما پارامتر های name_tt و family_tt را در سازنده کلاس مشتق شده به کلاس پایه منتقل می کنیم و هیچ استفاده ای در سازنده خود نمی کنیم. چون کلاس پایه ما دارای سازنده پارامتر دار است مجبور هستیم که این دو پارامتر را تعریف کنیم و با عبارت base به کلاس پایه منتقل می کنیم. و در نهایت با متد getitems مقادیر را نمایش می دهیم.

```
class student : Person
{
    protected string stdno;
    public student(string stdno_t, string name_tt,
        string family_tt) :base (name_tt,family_tt)
    {
        stdno = stdno_t;
    }
    public void getitems()
    {
        MessageBox.Show(name);
        MessageBox.Show(family);
        MessageBox.Show(stdno);
    }
}
```

پس به Form۱ برگشته و بر روی کنترل Click دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.



تصویر ۴۴ - ۵

```
private void button1_Click(object sender, EventArgs e)
{
    student st = new student("88104210", "رضا", "بهرامی راد");
    st.getititems();
}
```

اگر برنامه را اجرا کنید پیغام ها به صورت پشت سر هم نمایش داده می شوند. حال این سوال ممکن است به ذهن شما برسد که آیا ما همیشه دو کلاس داریم که از هم ارث بری می کنند، یعنی همیشه یک کلاس از کلاس دیگر ارث بری می کند و همه چیز به پایان می رسد؟ نه! ممکن است که اتفاقات دیگری هم رخ دهد مثلاً ما کلاس دیگری به نام GADUATE تعریف می کنیم که از کلاس Student ارث بری می کند. کلاس student از کلاس Person ارث بری می کند. یعنی کلاس GADUATE سومین کلاسی است که عمل ارث بری را انجام می دهد.

کلاس student مشکلات کلاس Person را از لحاظ پارامتر های ورودی حل می کند. حالا باید کلاس GADUATE به نحوی مشکلات پارامتر ورودی کلاس student را حل کند تا کلاس student هم بتواند به نحوی پارامتر های ورودی کلاس Person را حل نماید، نگران نباشید هیچ مشکلی پیش نمی آید.

توجه کنید که سطح دسترسی فیلد stdno را در سطح دسترسی protected تعریف کنید. حال دستورات زیر را در کلاس GADUATE تعریف کنید.

```
class GADUATE : student
{
    public GADUATE(string stdno, string name, string family)
        : base(stdno, name, family)
    {
    }
    public void getititems2()
    {
        MessageBox.Show(name);
        MessageBox.Show(family);
        MessageBox.Show(stdno);
    }
}
```

همانطور که در دستورات بالا مشاهده می کنید سه پارامتر برای متد سازنده کلاس GADUATE تعریف کرده ایم سپس با استفاده از دستور base متغیر ها را به کلاس student

منتقل کرده ایم. موقع اجرا پارامترها در متغیرهایی که در جلوی دستور base نوشته ایم قرار می گیرند و از آنجا به کلاس student منتقل می شوند. کلاس student آنها را در پارامترهای ورودی سازنده خود قرار می دهد و از آنجا در بخش متغیرهای base قرار می گیرند و از این قسمت هم در قسمت پارامترهای ورودی متد سازنده کلاس Person قرار می گیرند. سپس در متد ۲ getitems کلاً GADUATE مقادیر را نشان می دهیم.

بر روی Form یک کنترل button دیگر قرار داده و خاصیت Text آن را برابر Click۲ قرار می دهیم و در رویداد Click آن دستورات زیر را می نویسیم.

```
private void button2_Click(object sender, EventArgs e)
{
    GADUATE st = new GADUATE("8910253", "مرتضی", "سلیمانپور");
    st.getitems2();
}
```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

نکته!

هر چه قدر که در پروژه خود کلاس تعریف کنید بایستی مشکل پارامترهای ورودی را حل کنید.

چند ریختی (Polymorphism)

تا به اینجا دو رکن اساسی در برنامه نویسی شیء گرا یعنی وراثت و کپسوله سازی را بررسی کردیم، در این بخش به آخرین رکن برنامه نویسی شیء گرا می پردازیم. اگر کپسوله سازی و وراثت را به عنوان ضربات مهم در فنون کاراته تصور کنیم چند ریختی ضربه نهایی برای از کار انداختن حریف خواهد بود. وقتی نوبت به چند ریختی می رسد، الگوی واقعی شیء گرایی خود را نشان می دهد.

چند ریختی، ترجمه واژه Polymorphism است که از دو واژه Poly به معنای (چند) و Morph به معنای (ریخت یا شکل) تشکیل شده است. بنابراین، چند ریختی یعنی توانایی استفاده از شکل های مختلف یک نوع، بدون پرداختن به جزئیات آن است.

چند ریختی دلیلی است که ثابت می کند وراثت فقط به کار استفاده مجدد از دستورات نمی آید بلکه مهمترین کاربرد آن، ایجاد توانایی چند شکلی بودن از طریق روابط جانشین پذیری است.

برخی از مزایا و ویژگی های چند ریختی عبارتند از :

۱- امکان نوشتن برنامه هایی را فراهم می آورد که انواع مختلفی از کلاس های مرتبط را به شکل کلی اداره کنند.

۲- افزودن کلاس های جدید به سیستم را تسهیل می کند.

۳- با استفاده از چند ریختی امکان طراحی سیستم هایی فراهم می شود که به آسانی بسط پذیر باشند. برنامه ها می توانند اشیایی از تمام کلاس موجود در سلسله مراتب را به عنوان اشیایی از کلاس پایه مشترک استفاده کنند. و علاوه بر این، اگر کلاسی در سلسله مراتب کلاس هایی باشند که در برنامه از آنها استفاده شده است، افزودن آن برنامه با کمترین اصلاحات امکان پذیر است.

چند ریختی در زبان C# در حالت های مختلفی وجود دارد :

۱- متد های مجازی

۲- کلاس ها و متد های abstract

۳- delegate ها

ولی چند ریختی در حالت کلی به دو صورت پیاده سازی می شود :

۱- چند ریختی وراثتی (Inheritance Polymorphism)

۲- چند ریختی رابطه ای (Interface Inheritance)

در ادامه به معرفی هر یک از موارد گفته شده خواهیم پرداخت.

متد های Virtual و Override

متد های مجازی به ما این امکان را فراهم می آورند، که متد های کلاس مشتق شده بتوانند پیاده سازی ها متفاوتی از متد های کلاس پایه داشته باشند. به عبارت دیگر، یعنی اینکه بتوانیم متدی که در کلاس پایه تعریف شده است را بازنویسی کنیم.

برای مشخص کردن متد مجازی در کلاس پایه، باید واژه کلیدی virtual را قبل از آن به کار برد. علاوه بر این، در کلاس مشتق، هنگام تعریف مجدد متد مجازی، باید آن را با واژه override مشخص کرد.

(نام متد نوع داده برگشتی virtual سطح دسترسی

```
{  
;دستورات  
}
```

طرز تعریف دوباره متد virtual در کلاس مشتق شده

(نام متد نوع داده برگشتی override سطح دسترسی

```
{  
;دستورات جدید  
}
```

برای مثال دستورات زیر را ببینید.

```
public class Shape  
{  
...  
public virtual int area()  
{  
...  
}  
}
```

در کلاس Shape متد area به طور مجازی تعریف شده است. اکنون دستورات زیر را ببینید :

```
public class Circle : Shape  
{  
...  
public override int area()  
{  
...  
}  
}
```

متد area در کلاس Circle که از کلاس Shape مشتق می شود، با واژه override مشخص شده است.

چند قاعده مهم وجود دارد که بایستی در زمان تعریف متد های چند شکلی با استفاده از کلمات کلیدی virtual و override از آنها پیروی کرد :

- ۱- نمی توانید یک متد private را به همراه کلمات کلیدی virtual و override تعریف کنید. اگر این کار را انجام دهید، یک خطای زمان کامپایل دریافت خواهید کرد.
 - ۲- دو متد باید مثل هم باشند. یعنی با اسم، نوع پارامترها و نوع برگشتی آنها یکی باشد.
 - ۳- دو متد باید اجازه دستیابی یکسانی را ارائه دهند. برای مثال، اگر یکی از متد ها public باشد، متد دیگر نیز باید public باشد.
 - ۴- فقط می توانید متد virtual را override کنید. اگر متد کلاس پایه، virtual نباشد و سعی کنید که آن را override کنید، یک خطای زمان کامپایل دریافت خواهید کرد. این منطقی است، زیرا بر عهده کلاس پایه است که تصمیم بگیرد که آیا متد های آن می توانند override شوند یا خیر.
 - ۵- اگر کلاس مشتق شده، متد را با استفاده از کلمه کلیدی override تعریف نکند، متد کلاس پایه override نمی شود. بلکه متد مورد نظر در کلاس مشتق شده، متد کاملاً متفاوتی می شود که فقط اسم آن با اسم متد در کلاس پایه، یکی است. البته این باعث می شود که کامپایلر به شما اخطار دهد که می توانید با استفاده از کلمه کلیدی new کاری کنید که این پیغام اخطار نمایش داده نشود.
 - ۶- یک متد override، به صورت ضمنی virtual است و می تواند در کلاس مشتق شده از این کلاس، override شود. ولی، می توانید صریحاً و با استفاده از کلمه کلیدی virtual مشخص کنید که یک متد override، virtual است. زیرا هر متدی که هیچ کلمه کلیدی خاصی نداشته باشد، یک متد غیر چند ریختی است که متد دیگری را مخفی نمی کند (این قاعده تقریباً همیشه صدق می کند). بهترین روش برای درک این مفاهیم بررسی چندین مثال در این زمینه است.
- قصد داریم کلاسی تعریف کنیم که حقوق کارمندان را برای ما محاسبه کرده و اعلام کند. ابتدا یک پروژه جدید از نوع ویندوزی ساخته و یک کلاس داخلی به نام Salary برای آن تعریف می کنیم. سپس دستورات زیر را در کلاس Salary می نویسیم :

```

class Salary
{
    int t_work;
    int salary;
    public Salary(int worktime)
    {
        t_work = worktime;
    }
    protected virtual int calculatetax()
    {
        return (((t_work * 50)*20/100));
    }
    public int getsalary()
    {
        salary = (t_work * 50) - calculatetax();
        return salary;
    }
}

```

در کلاس Salary اولین فیلد مربوط به مقدار زمان و ساعت کاری است که با متغیر t_work بیان شده است. برای اینکه برنامه شلوغ نشود و فیلد های زیادی تعریف نکنیم ما هر ساعت کاری را ۵۰ دلار حساب می کنیم البته ۵۰ دلار برای یک کارمند حقوق بالایی است. یکی از فیلد های دیگر میزان حقوق هست که با متغیر salary تعیین می کنیم و از نوع صحیح در نظر می گیریم. بعد از آن از یک متد سازنده استفاده کردیم و در این متد از کاربر می خواهیم که ساعت کاری کارمند را وارد کند. سپس یک متد به نام calculatetax می نویسیم و در آن میزان حقوق کارمند را محاسبه می کنیم. مساله ای که برای ما در درس ساز هست محاسبه مالیات شخص می باشد تا از میزان حقوق آن کم کنیم تا به یک نتیجه مطلوب برسیم، مقدار مالیات را ۲۰ درصد در نظر می گیریم. پس یک متد از نوع مجازی به نام getsalary با سطح دسترسی protected تعریف می کنیم.

حال باید از این کلاس در برنامه خودمان استفاده کنیم. در Form۱ بر روی دکمه Click دابل کلیک کرده و در رویداد Click آن دستورات زیر را می نویسیم :



تصویر ۴۵ - ۵

```
private void button1_Click(object sender, EventArgs e)
{
    Salary sa = new Salary(Convert.ToInt32(textBox1.Text.Trim()));
    MessageBox.Show(sa.getsalary().ToString());
}
```

اگر برنامه را اجرا کرده و مقدار ۱۰ را وارد کنید میزان دریافتی کارمند برابر ۴۰۰ خواهد بود که برای یک کارمند حقوق بالایی است.

تا به حال هیچ استفاده ای از متد مجازی calculatetax نکرده ایم. خوب در بین کارمندان، ممکن کارمندان خاصی باشند که به دلایل مختلفی مالیات کسر شده از حقوق آنها ۱۰ درصد باشد، خوب به عنوان یک برنامه نویس چه فکر می کنید برای این مشکل! شاید اولین فکری که به نظرتان برسد تعریف یک کلاس جدید باشد و انجام تمام مراحل تا رسیدن به یک نتیجه مشخص.

ولی ما یک کار بهتر انجام می دهیم و آن تعریف یک کلاس جدید با نام Salary۲ و ارث بری از کلاس Salary است. درون کلاس Salary۲ متد مجازی calculatetax را به صورت override استفاده می کنیم.

کافی است که فیلد t_work را درون کلاس Salary به سطح دسترسی protected تغییر می دهیم تا درون کلاس Salary۲ نیز قابل استفاده باشد.

```
class Salary
{
    protected int t_work;
    int salary;
    public Salary(int worktime)
    {
        t_work = worktime;
    }
    protected virtual int calculatetax()
    {
        return (((t_work * 50)*20/100));
    }
    public int getsalary()
    {
        salary = (t_work * 50) - calculatetax();
        return salary;
    }
}
```

```
class Salary2 : Salary
{
    public Salary2(int worktime)
        : base(worktime)
    {
    }
    protected override int calculatetax()
    {
        return (((t_work * 50) * 10) / 100);
    }
}
```

به سازنده Salary2 در کلاس Salary2 دقت کنید در نکته قبل هم گفتیم که بایستی سازنده کلاس مشتق شده از لحاظ پارامتری با سازنده کلاس پایه یکسان و هم نوع باشد. حال به Form1 برگشته و آن را مطابق پنجره تصویر ۴۶ - ۵ طراحی می کنیم.



تصویر ۴۶ - ۵

بر روی دکمه Click2 دابل کلیک کرده و دستورات زیر را بنویسید.

```
private void button2_Click(object sender, EventArgs e)
{
    Salary2 sa = new Salary2(Convert.ToInt32(textBox2.Text.Trim()));
    MessageBox.Show(sa.getsalary().ToString());
}
```

در صورتی که برنامه را اجرا کرده و برای کارمندان خاص، میزان ساعت کاری را ۱۰ وارد کنید میزان دریافتی را برابر ۴۵۰ نشان خواهد داد.

مثال جدید ما شبیه مثال قبلی است ولی با اندکی تفاوت. به طور مثال اگر مرد کارمند بود ۱۰ درصد به حقوقش اضافه شود یعنی به دلایلی کارمندان مرد پس از کسر مالیات باز ده درصد افزایش حقوق داشته باشند. ولی اگر کارمند، خانم بود افزایش حقوقی نداشته باشد.

یک پروژه ویندوزی جدید ساخته و درون آن یک کلاس داخلی به نام Salary تعریف می کنیم. در این کلاس فیلد جدید به نام sex و با سطح دسترسی protected از نوع bool برای تعیین جنسیت مشخص کرده ایم. لذا در کلاس Salary دستورات زیر را وارد کنید.

```

class Salary
{
    protected int t_work;
    protected bool sex;
    int salary;
    public Salary(int worktime, bool sex_t)
    {
        sex = sex_t;
        t_work = worktime;
    }
    protected virtual int calculatetax()
    {
        return(((t_work*50)*20)/100);
    }
    public int getsalary()
    {
        salary = ((t_work * 50) - calculatetax());
        if (sex == true)
        {
            int s_t = (((t_work * 50) * 10) / 100);
            salary = salary + s_t;
        }
        return salary;
    }
}

```

در Form۱ بر روی دکمه Click دابل کلیک کنید، سپس در رویداد Click آن دستورات زیر را بنویسید.



تصویر ۴۷ - ۵

دو کنترل CheckBox برای تعیین جنیست به کار می رود. در بخش public partial یک متغیر به نام a تعریف می کنیم. متغیر هایی که در این بخش تعریف می شوند در کل کلاس Form قابل استفاده هستند. از دو کنترل checkbox و متغیر a برای تعیین جنیست استفاده می کنیم.

```

public partial class Form1 : Form
{
    bool a;

```

```
private void button1_Click(object sender, EventArgs e)
{
    if (checkBox1.Checked)
        a = true;
    if (checkBox2.Checked)
        a = false;
    Salary sa = new Salary(Convert.ToInt32(textBox1.Text.Trim()), a);
    MessageBox.Show(sa.getsalary().ToString());
}
```

اگر برنامه را اجرا کرده و گزینه مرد را انتخاب نموده و مقدار ۱۰ را وارد کنید مقدار ۴۵۰ برای کاربر نمایش داده می شود. ولی اگر گزینه زن را انتخاب کنید و مقدار ۱۰ را وارد کنید مقدار ۴۰۰ را نمایش خواهد داد.

مثال ۳۴ - ۵: برنامه زیر نحوه چاپ یک پیغام را با استفاده از متد virtual و override در یک برنامه نشان می دهد؟

حل: برای حل مساله کافی است که یک پروژه از نوع Console ساخته و یک کلاس داخلی با نام A ایجاد کرده و به نحوی که کلاس B از کلاس A ارث می برد.

```
class A
{
    public virtual void Test()
    {
        Console.WriteLine("Are ypu a Programmer?");
    }
}
class B : A
{
    public override void Test()
    {
        Console.WriteLine("Yes, I am a Programmer");
    }
}
```

در متد اصلی کلاس program.cs دستورات زیر را وارد کنید.

```
static void Main(string[] args)
{
    // Compile-time type is A.
    // Runtime type is A as well.
    A ref1 = new A();
    ref1.Test();
    // Compile-time type is A.
    // Runtime type is B.
    A ref2 = new B();
    ref2.Test();
}
```

در صورتی که برنامه را اجرا کنید، خروجی برنامه به صورت زیر خواهد بود.

```
Are ypu a Programer?  
Yes, I am a Programer
```

کلاس ها و متد های abstract

کلاس های abstract کلاس هایی هستند که صرفا جهت ساخت دهی به مدل شیء گرایی برنامه مورد استفاده قرار می گیرند. امکان ایجاد اشیاء از این کلاس ها وجود ندارد و اساسا این کلاس ها به این خاطر ایجاد می شوند که بتوانیم از آنها کلاس مشتق کنیم و از کلاس های مشتق شده استفاده کنیم.

متد های abstract نیز متد هایی هستند که هیچ گونه پیاده سازی ندارند و تنها به این جهت تعریف می شوند که کلاس مشتق شده آن متد را باز نویسی کند. این متد ها در حقیقت ساختار یک متد را معرفی می کنند که ملزم هستیم در کلاس مشتق شده، ساختار متد مورد نظر مانند (پارامتر ها، نوع بازگشتی، نام متد) را پیاده سازی کنیم.

به کلاس ها و متد های abstract متد های انتزاعی نیز می گویند. تفاوت کلاس های انتزاعی و غیر انتزاعی را می توان به شرح زیر بیان کرد :

۱- کلاس های انتزاعی قابل نمونه سازی نیستند.

۲- کلاس های انتزاعی، کلاس های پایه کلاس های مشتق محسوب می شوند.

۳- کلاس های انتزاعی شامل متد ها و خواص انتزاعی اند. متد ها و خواص انتزاعی، در کلاس انتزاعی پیاده سازی نمی شوند، بلکه فقط به عنوان جا نگهدار محسوب خواهند شد. بنابراین، خواص و متد های انتزاعی باید در کلاس مشتق پیاده سازی شوند.

کلاس ها و متد های انتزاعی با واژه abstract مشخص می شوند. علاوه بر این، متد ها و خواص انتزاعی نیز با واژه abstract تعیین می گردند. بدیهی است که متد های انتزاعی، به طور خودکار، مجازی نیز هستند و لازم نیست که آنها را با واژه virtual مشخص کرد.

```
public abstract class test  
{  
    public abstract متد نام (parameters)  
    {  
        ...  
    }  
}
```

همانطور که مشاهده می کنید کلمه کلیدی abstract در تعریف کلاس یا متد، پس از نوع دستیابی قرار می گیرند. توجه داشته باشید که متد static نمی توانند abstract باشند. تعریف متد یا خاصیت abstract در کلاس غیر abstract شما را با خطا مواجه خواهد کرد. تلاش برای نمونه سازی کلاس انتزاعی، با خطای ترجمه مواجه می شود. اگر در کلاس مشتق، متد با خاصیت انتزاعی را پیاده سازی نکنید، با خطا مواجه می شوید، مگر این که کلاسی مشتق نیز انتزاعی باشد.

همانطور که می دانید تمام کنترل هایی که در برنامه های خود از آنها استفاده می کنید در حقیقت یک شیء از کلاسی مربوط به آن کنترل هستند. برای مثال هنگامی که یک کنترل Button روی Form برنامه قرار می دهید، در حقیقت یک شیء از کلاسی به نام Button ایجاد کرده اید و در طی برنامه نیز با آن شیء کار می کنید.

فرض کنید بخواهید کلاس مربوط به چنین کنترل هایی را در برنامه خود بنویسید. برای این کار ابتدا باید یک کلاس پایه، برای مثال به نام Window ایجاد کرده و تمام خاصیت ها و متد های مشترک بین کنترل ها را در این کلاس قرار دهید. سپس تمام کنترل هایی که می خواهید ایجاد کنید را، مانند کنترل Button و یا کنترل TextBox، از این کلاس به ارث ببرید. در این حالت مسلماً نمی خواهید بعد ها کسی بتواند شیء را از کلاس Window نمونه سازی کند و از آن شیء در برنامه خود استفاده کند، چون این امر بی معنی است. کلاس Window نشان دهنده هیچ کنترل خاصی نیست، بلکه فقط به عنوان یک کلاس پایه برای تمام کنترل ها به کار می رود. بنابراین این کلاس را به عنوان یک کلاس abstract مشخص می کنید. کلاس های abstract به بیان ساده تر کلاس هایی هستند که نمی توان هیچ شیء ای را از آنها نمونه سازی کرد و حتماً باید به عنوان کلاس های پایه برای دیگر کلاس ها مورد استفاده قرار گیرند.

مطالب تئوری در مورد کلاس ها و متد های abstract به اندازه کافی گفته شد حالا بهتر است با یک مثال عملی به تشریح و کارکرد این متد ها بپردازیم. ابتدا یک پروژه جدید از نوع Console ایجاد کرده و یک کلاس داخلی به نام Window برای آن تعریف کنید و دستورات زیر را به آن اضافه کنید.

```

abstract public class Window
{
    public int Left;
    public int Top;
    //Constructor Method to set
    // the initial value of field
    public Window()
    {
        this.Top = 0;
        this.Left = 0;
    }
    //simulates drawing the window
    // notice : no implementation
    abstract public void DrawingWindow();
}

```

در داخل همین فضای نام و بعد از کلاس Window، کلاس جدیدی به نام ListBox به صورت زیر تعریف کنید. این کلاس از کلاس Window مشتق می شود و به صورت فرضی یک کنترل ListBox را در صفحه نمایش می دهد. در مورد تداخل نام این کلاس با کلاس ListBox در ویژوال استودیو نگران نباشید، زیرا کلاس ListBox ویژوال استودیو در فضای نام System.Windows.Forms قرار دارد.

```

public class ListBox : Window
{
    private string listBoxContents;
    //constructor adds a parameters
    public ListBox(int top, int left, string contents)
    {
        Top = top;
        Left = left;
        listBoxContents = contents;
    }
    // an overridden version implementation the
    // abstract method
    public override void DrawingWindow()
    {
        Console.WriteLine("Writing string to the" +
            "listbox : " + listBoxContents);
    }
}

```

می خواهیم کنترل دیگری نیز مانند یک کنترل Button، با استفاده از کلاس Window ایجاد کنیم. برای این کار کلاس دیگری را بعد از کلاس ListBox در همین فضای نام ایجاد می کنیم.

```
public class Button : Window
{
    //The new class's constructor
    public Button(int top, int left)
    {
        Top = top;
        Left = left;
    }
    //implement the abstract method
    public override void DrawingWindow()
    {
        Console.WriteLine("Drawing a button at" + Top
            + ", " + Left);
    }
}
```

ایجاد کلاس های مورد نیاز برنامه به پایان رسید، حالا باید نحوه عملکرد آنها را بررسی کنیم. در کلاس program.cs و در متد اصلی برنامه دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    ListBox lstBox1 = new ListBox(1, 2, "First List Box");
    ListBox lstBox2 = new ListBox(3, 4, "Second List Box");
    Button newButtn = new Button(5, 6);
    lstBox1.DrawingWindow();
    lstBox2.DrawingWindow();
    newButtn.DrawingWindow();
}
```

در صورتی که برنامه را اجرا کنید، خروجی برنامه به صورت زیر خواهد بود.

```
Writing string to the listbox : First List Box
Writing string to the listbox : Second List Box
Drawing a button at 5,6
```

بنابراین نگاهی کلی به این برنامه می اندازیم. برنامه را با یک کلاس پایه به نام Window شروع کردیم. همانطور که گفتیم این کلاس باید به گونه ای باشد که تمام خاصیت ها و متد هایی را که بین تمام کنترل ها مشترک است شامل شود. همچنین برای اینکه نتوان شیء ای را از این کلاس ایجاد کرد آن را به صورت abstract تعریف می کنیم. تمام کنترل ها در یک Form دارای موقعیت مکانی معینی هستند که به وسیله دو خاصیت Top (فاصله کنترل از بالای Form) و Left (فاصله کنترل از سمت چپ Form) مشخص می شوند. پس همه آنها باید شامل دو فیلد به نام های Top و Left باشند. به همین دلیل این دو فیلد را در کلاس Window قرار می دهیم.

همچنین می خواهیم تمام کنترل هایی که از این کلاس مشتق می شوند، حتما متدی به نام DrawWindow داشته باشند تا آنها را در Form رسم کند. اما همانطور که گفتیم این متد برای هر کنترل متفاوت است و نمی توانیم پیاده سازی آن را در کلاس Window قرار دهیم، بنابراین آن را در کلاس پایه به صورت abstract معرفی می کنیم تا تمام کلاس هایی که از کلاس Window مشتق می شوند چنین متدی را در خود ایجاد کنند. به این صورت می توانیم تضمین کنیم که تمام کنترل هایی که از کلاس Window مشتق می شوند دارای متدی به نام DrawWindow هستند که آنها را در صفحه رسم می کند. دقت کنید که اگر متدی در یک کلاس به عنوان abstract معرفی شود، آن متد نباید شامل هیچ دستوری باشد. به عبارت دیگر آن متد نباید دارای پیاده سازی باشد. در انتها نیز یک متد سازنده برای این کلاس ایجاد می کنیم تا مقدار اولیه فیلد های Top و Left را مشخص کند.

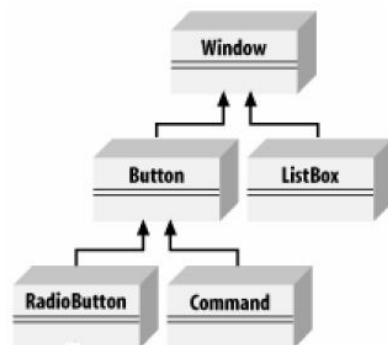
باید با استفاده از کلاس Window، کنترل های مورد نظرم را ایجاد کنیم. ابتدا از کنترلی همانند ListBox شروع می کنیم. برای این کار کلاسی به نام ListBox ایجاد کرده و کلاس Window را به عنوان کلاس پایه آن مشخص می کنیم. در این کلاس فیلد جدیدی به نام listBoxContents از نوع رشته ای ایجاد می کنیم تا علاوه بر موقعیت هر کنترل List Box، محتویات آن را نیز بتوانیم نگهداری کنیم و در مواقع مورد نیاز نمایش دهیم. همچنین یک متد سازنده نیز در این کلاس قرار می دهیم تا مقدار فیلد ها را تنظیم کند، همانطور که مشاهده می کنید این متد سازنده همانند متد های معمولی سه پارامتر می گیرد: پارامتر اول برای تعیین فاصله ی کنترل از سمت چپ (مقدار فیلد Left)، پارامتر دوم برای تعیین فاصله ی کنترل از بالای Form (مقدار فیلد Top) و پارامتر سوم برای تعیین متنی که باید در ListBox قرار گیرد (مقدار خاصیت listBoxContents). به این ترتیب می توانیم در همان ابتدای ایجاد شیء از کاربر بخواهیم که مقدار این موارد را تعیین کند. هنوز کلاس ListBox کامل نشده است و اگر در این مرحله آن را کامپایل کنیم، با خطا مواجه خواهیم شد، زیرا هنوز متد DrawWindow را در این کلاس override نکرده ایم و در کلاس Window هم این متد به صورت abstract مشخص شده است، پس باید در کلاس مشتق شده override شود. بنابراین

همانند override کردن متد های عادی که در قسمت های قبلی مشاهده کرده اید، متد DrawWindow از کلاس پایه را در این کلاس override می کنیم.

پس از اینکه که کنترل اول را ایجاد کردیم، به سراغ کنترل بعدی که یک Button است می رویم. برای ایجاد این کنترل نیز همانند کنترل ListBox، یک کلاس به نام Button ایجاد کرده و کلاس Window را به عنوان کلاس پایه ی آن مشخص می کنیم. بقیه موارد این کلاس نیز مشابه کلاس ListBox است و هیچ ابهامی در آن وجود ندارد. البته دقت کنید در این کلاس نیز همانند کلاس ListBox، اگر متد DrawWindow را override نکنیم، با خطای زمان کامپایل مواجه خواهیم شد.

حال که کنترل های مورد نظرم را ایجاد کردیم، باید آنها را در برنامه تست کنیم. برای این کار، ابتدا دو شیء از نوع ListBox و یک شیء نیز از نوع Button ایجاد می کنیم. سپس با فراخوانی متد DrawWindow در آنها، این کنترل ها را در صفحه نمایش می دهیم.

توجه! از یک کلاس abstract می توان یک کلاس abstract دیگر مشتق کرد. برای مثال تصور کنید می خواهید کلاسی به نام Button ایجاد کنید، اما اجازه ندهید کسی از این کلاس شیء ای را ایجاد کند. بلکه کلاس هایی مانند Command، RadioButton و ... را از این کلاس مشتق کنید. بنابراین کلاس Button خود باید یک کلاس abstract باشد که از کلاس abstract دیگری به نام window مشتق می شود (پنجره تصویر ۴۸ - ۵). در این حالت نیازی نیست که متد های abstract کلاس پایه را در کلاس مشتق شده override کنید و می توانید از پیاده سازی متد DrawWindow در کلاس Button صرف نظر کنید.



تصویر ۴۸ - ۵

مثال ۳۳ - ۵: برنامه زیر نحوه چاپ دو پیغام را با استفاده از کلاس و متد abstract بیان می کند. برنامه ساده و درک آن آسان است و نیاز به توضیحات اضافی ندارد.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و یک کلاس داخلی جدید با نام Test ایجاد کنید. سپس کلاس های Example۱ و Example۲ را طوری طراحی کنید که از کلاس Test ارث بری کنند. کلاس های Example۱ و Example۲ را در فضای نام کلاس Test بنویسید.

```
abstract class Test
{
    public int _a;
    public abstract void A();
}
class Example1 : Test
{
    public override void A()
    {
        Console.WriteLine("Are you a Programmer? ");
        base._a++;
    }
}
class Example2 : Test
{
    public override void A()
    {
        Console.WriteLine("Yes, I am a Programmer");
        base._a--;
    }
}
```

در کلاس اصلی برنامه یعنی program.cs و در بخش متد اصلی برنامه دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    // Reference Example1 through Test type.
    Test test1 = new Example1();
    test1.A();
    // Reference Example2 through Test type.
    Test test2 = new Example2();
    test2.A();
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

کلاس های sealed

در قسمت قبل با کلاس های abstract آشنا شدید و مشاهده کردید که چگونه می توان کلاس هایی داشت که فقط به عنوان کلاس پایه استفاده شوند و نتوان شیء ای از آنها نمونه سازی کرد.

کلاس های sealed دقیقاً مفهومی عکس کلاس های abstract دارند. به عبارت دیگر کلاس های sealed کلاس هایی هستند که نمی توان آنها را به عنوان کلاس پایه مورد استفاده قرار داد و از آنها کلاس جدیدی را مشتق کرد، بلکه فقط می توان از آنها برای نمونه سازی اشیاء استفاده کرد. برای تعریف یک کلاس sealed باید از کلمه کلیدی sealed قبل از تعریف کلاس همانند کد زیر استفاده کنید.

```
public sealed class SealedClass
{
    //Implementation here ...
}
```

واژه sealed چنین مفهومی را برای کلاس ها و متد ها فراهم می کند، به طوری که وقتی جلوی کلاس یا متدی قرار می گیرد، از وراثت آنها جلوگیری خواهد شد. متدی که با واژه sealed مشخص می شود، نمی تواند در کلاس مشتق تعریف شود. متد هایی که به صورت static و private تعریف می شوند به طور خودکار sealed هستند و قابل تعریف مجدد در کلاس مشتق نیستند.

کلاسی که sealed تعریف شده باشد، نمی تواند کلاس پایه باشد. تمام متد ها در کلاس sealed به طور ضمنی sealed هستند.

مثال ۳۴ - ۵: برنامه زیر نحوه استفاده از کلاس های sealed را نشان می دهد.

حل : در ابتدا یک پروژه جدید از نوع Console ایجاد کنید، سپس یک کلاس داخلی به نام SealedClass ایجاد کرده و دستورات زیر را در آن بنویسید.

```
// Sealed class
sealed class SealedClass
{
    public int Add(int x, int y)
    {
        return x + y;
    }
}
```

این متد جمع دو عدد را بر می گرداند. از این متد در داخل متد اصلی کلاس program.cs استفاده کنید.

```
static void Main(string[] args)
{
    SealedClass sealedCls = new SealedClass();
    int total = sealedCls.Add(4, 5);
    Console.WriteLine("Total = " + total.ToString());
}
```

چند ریختی وراثتی

با استفاده از این ویژگی می توان به متد کلاس مشتق شده پیاده سازی متفاوتی از متد کلاس پایه ایجاد نمود. این ویژگی در جایی مناسب است که می خواهید گروهی از اشیاء را به یک آرایه اختصاص دهید و سپس از متد هر یک از آنها استفاده کنید. این اشیاء الزاماً نباید از یک نوع شیء باشند هر چند اگر این اشیاء به واسطه ارث بری به یکدیگر مرتبط باشند می توان آنها را به عنوان انواع ارث بری شده به آرایه اضافه نمود. اگر هر یک از این اشیاء دارای متدی با نام مشترک باشند آنگاه می توان هر یک از آنها را جداگانه پیاده سازی و استفاده نمود. همیشه به یاد داشته باشید که override کردن متد ها همان چند ریختی است، در واقع چند ریختی تغییر انجام عمل یک متد است.

خوب بهتر است که با یک مثال عملی مباحث خود را پیش ببریم. ابتدا یک پروژه از نوع ویندوزی ساخته و درون این پروژه یک کلاس داخلی با عنوان DrawingObject ایجاد می کنیم و سپس دستورات زیر را می نویسیم:

```
class DrawingObject
{
    public virtual void Draw()
    {
    }
}
class Line : DrawingObject
{
    public override void Draw()
    {
        MessageBox.Show("I am a Line");
    }
}
```

```

class Circle : DrawingObject
{
    public override void Draw()
    {
        MessageBox.Show("I am a Circle");
    }
}
class Rectangle : DrawingObject
{
    public override void Draw()
    {
        MessageBox.Show("I am a Rectangle");
    }
}

```

چون می خواهیم پدیده چند ریختی را یک مقدار نمادین تر نشان دهیم. این کار را با استفاده از گرفتن یک تعدادی آرایه از نوع DrawingObject و تغییر آنها به حالاتی مانند Rectangle , Circle , Line انجام می دهیم.

ابتدا Form خود را مانند پنجره تصویر ۴۹ - ۵ طراحی می کنیم.



تصویر ۴۹ - ۵

در پنجره تصویر ۴۹ - ۵ بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را اضافه کنید.

```

private void button1_Click(object sender, EventArgs e)
{
    DrawingObject[] obj = new DrawingObject[3];
    obj[0] = new Line();
    obj[1] = new Circle();
    obj[2]=new Rectangle();
    foreach (DrawingObject objt in obj)
    {
        objt.Draw();
    }
}

```

در این برنامه از کلاس DrawingObject یک آرایه با تعدادی خانه تعریف کردیم و هر کدام

از آنها را به یکی از کلاس های مشتق شده از DrawingObject نسبت دادیم. دقت کنید مساله چند ریختی اینجا به خوبی نمایان می شود. ما آرایه ای که تعریف کردیم از نوع DrawingObject است اما به ما اجازه می دهد از کلاس های مشتق شده آن در حالت و شکل خود استفاده کنیم. ما نیز سه آرایه Line, Circle, Rectangle تعریف کردیم که از کلاس DrawingObject ارث بری می کنند. سپس درون دستور foreach متد Draw را فراخوانی کرده و از آن استفاده نمودیم.

آشنایی با Indexer

همانطور که خاصیت یک فیلد هوشمند است، Indexer هم یک آرایه هوشمند است. به عبارت دیگر، دستور زبان استفاده از یک Indexer مانند یک آرایه است. Indexer ها مفهومی بسیار ساده در زبان C# هستند با استفاده از آنها می توانید از کلاس خود همانند یک آرایه استفاده کنید. در داخل کلاس مجموعه ای از مقادیر را به هر طریقی که مورد نظرتان هست مدیریت کنید. این اشیاء می توانند حاوی مجموعه ای از اعضای کلاس یک آرایه دیگر و یا مجموعه ای از ساختار های پیچیده داده ای باشند، جدا از پیاده سازی داخلی کلاس، داده های این ساختار ها از طریق استفاده از Indexer قابل دسترسی هستند.

گرچه عملگر اندیس مربوط به Indexer همانند عملگر اندیس آرایه استفاده می شود، Indexer در کلاس به صورت خاصیت تعریف می شوند. بر خلاف خاصیت های معمولی که می توانستید نام دلخواهی را برای آنها انتخاب کنید، Indexer باید با واژه کلیدی this تعریف شود. ساختار کلی Indexer ها در صفحه بعد آمده است :

```

this [IndexType\ , name\ , IndexType2, name2, ...]
{
get
{
//use name\ , name2 , ... to get data
}
set
{
//use name\ , name2 , ... to set data
}
}

```

سطح دستیابی n، نوع دستیابی به خاصیت را مشخص می کند که قبلا راجع به آن صحبت کردیم. نوع داده برگشتی، یکی از انواع معتبر در زبان C# است. پارامتر های IndexType که در [] مشخص شده اند، توسط دستیاب های get و set قابل دستیابی هستند. این دستیاب ها چگونگی تعریف اندیس یا اندیس ها را برای انتخاب یا اصلاح عناصر داده مشخص می کنند. همانند خاصیت های عادی، get باید مقداری از نوع داده برگشتی را برگرداند و set می تواند با استفاده از واژه کلیدی value به مقداری که باید به عنصر داده نسبت داده شود، دستیابی داشته باشد.

توجه داشته باشید که اگر

۱- اگر Indexer را با واژه static تعریف کنید با خطا مواجه می شوید.

۲- Indexer یک متد نیست، پس نیازی به پارانتز ندارد.

۳- نوع Indexer، نوع داده bool است (یعنی true یا false).

۴- پارامتری که بین کروشه باز و بسته قرار می گیرد از نوع int است.

۵- تمام Indexer ها به جای اسم متد از کلمه کلیدی this استفاده می کنند.

۶- Indexer مثل خاصیت ها دارای متد get و set هستند.

مقایسه بین Indexer ها و متد ها

۱- متد می تواند بدون پارامتر باشد، در حالی که Indexer باید حداقل یک پارامتر داشته باشد.

```
public int this [ ] {...} // compile-time error
```

۲- Indexer باید متد get و یا set داشته باشد. حتی اگر متد های get و set کد مشترکی داشته باشند (مثل بررسی محدوده اندیس)، نمی توانید این کد `public int this [string name]{...} //okay` مشترک را قبل یا بعد از این متد ها قرار دهید.

```
public bool this [ int index ]  
{  
    boundsCheck (index); // compile-time error  
    get {...}  
    set {...}  
}
```

۳- متد می تواند نوع برگشتی void داشته باشد در حالی که Indexer نمی تواند.

مقایسه بین Indexer ها و آرایه ها

دستور زبان (Syntax) استفاده از Indexer، بسیار شبیه آرایه است. با این حال، چند تفاوت مهم بین آنها وجود دارد :

۱- Indexer ها می توانند از اندیس غیر عددی نیز استفاده کنند. در حالی که آرایه ها فقط می توانند از اندیس های عددی استفاده کنند.

۲- می توان Indexer ها را مثل متد ها overload کرد، در صورتی که آرایه ها نمی توانند overload شوند.

۳- نمی توان از Indexer به صورت پارامتر های ref یا out استفاده کرد، در حالی که می توان از عناصر آرایه برای این منظور استفاده کرد.

```
IntBits bits; //bits contains indexer  
Method (ref bits[1]); //compile-time error
```

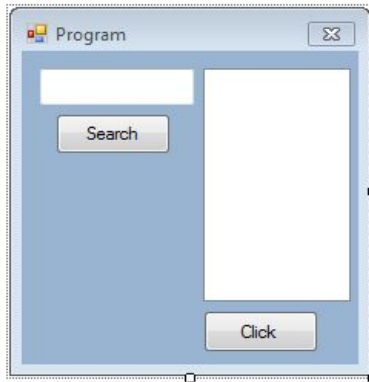
مقایسه بین Indexer ها و خاصیت ها

Indexer و خاصیت از این نظر شبیه به هم هستند که هر دو از متد get و set استفاده می کنند. Indexer همانند یک خاصیت با چندین مقدار است. با این حال، چند تفاوت مهم بین آنها وجود دارد.

مهمترین تفاوت این است که می توان خاصیت های static تعریف کرد، ولی Indexer های static غیر مجاز هستند.

حال بهتر است کمی از مطالب تئوری فاصله بگیریم و شروع به استفاده از Indexer در محیط برنامه نویسی بپردازیم:

ابتدا یک پروژه جدید ویندوزی ساخته و سپس Form آن را مطابق پنجره تصویر ۵۰ - ۵ طراحی کنید.



تصویر ۵۰ - ۵

در این برنامه قصد داریم یک Indexer تعریف کنیم و با استفاده از آن خانه های آرایه را پر کنیم و مقادیر وارد شده را درون یک کنترل List View نمایش دهیم. سپس مقداری را داخل کنترل textBox وارد کرده و با کلیک دکمه Search اجزای داخل کنترل List View را جستجو کند و در صورت پیدا شدن رنگ پس زمینه و پیش زمینه عنصر مورد نظر تغییر کند.

در داخل فضای نام Form جاری یک کلاس به نام intindexer تعریف کرده و دستورات زیر را در آن می نویسیم. دقت کنید که کلاس Form در ابتدای تعاریف کلاس در این فضای نام قرار گیرد و گرنه با خطا روبرو خواهید شد.

توجه! خاصیت Enable دکمه Search را برابر False قرار دهید.

```
class intindexer
{
    private string[] array;
    public intindexer(int size)
    {
        array = new string[size];
        for (int i = 0; i < size; i++)
        {
            array[i] = "EMPTY";
        }
    }
    public string this[int pos]
    {
        get
        {
            return array[pos];
        }
        set
        {
            array[pos] = value;
        }
    }
}
```

در قسمت public partial دستورات زیر را بنویسید.

```
public partial class Form1 : Form
{
    intindexer arr = new intindexer(10);
```

در دستورات بالا یک متغیر واسط از کلاس intindexer ایجاد می کنیم. پس بر روی دکمه Click دابل کلیک کرده و در رویداد Click آن دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    arr[5] = "Reza";
    arr[2] = "Morteza";
    arr[3] = "Sajad";
    for (int i=0 ; i < 10; i++)
        listView1.Items.Add(arr[i]);
    button2.Enabled = true;
}
```

بر روی دکمه Search دابل کلیک کرده و دستورات زیر را وارد کنید.

```
private void button2_Click(object sender, EventArgs e)
{
    for (int i = 0; i < 10; i++)
    {
        if (arr[i] == textBox1.Text.Trim())
        {
            listView1.Items[i].BackColor = Color.Yellow;
            listView1.Items[i].ForeColor = Color.Red;
        }
    }
}
```

برنامه را اجرا کنید و در ابتدا بر روی دکمه Click دابل کلیک کنید تا خانه های آرایه درون Indexer قرار بگیرند. سپس کلمه Reza را در در textbox تایپ کرده و دکمه Search را کلیک کنید. و نتیجه را مشاهده کنید.

سرریزی Indexer ها

زمانی که از دو یا چند Indexer درون یک کلاس استفاده می کنیم، سرریزی (Overloading) Indexer ها رخ می دهد. در زمان فراخوانی Indexer ها، کلاس تنها از روی نوع بازگشتی Indexer و تعداد پارامتر های آن متوجه می شود که منظور فراخواننده استفاده از کدام Indexer بوده است.

مثال زیر نحوه سرریزی Indexer ها را نشان می دهد. ابتدا یک پروژه جدید از نوع Console ایجاد کرده و یک کلاس داخلی به نام OverIndexer ایجاد کرده و دستورات زیر را در آن بنویسید.

این مثال، نحوه سرریز کردن Indexer ها را نشان می دهد. اولین Indexer که دارای پارامتری از نوع int تحت عنوان pos است که دقیقا مشابه مثال قبلی است ولی در اینجا Indexer جدیدی نیز وجود دارد که پارامتری از نوع string دریافت می کند. بخش get، Indexer جدید رشته ای را بر می گرداند که نمایشی از تعداد آیتم هایی است که با پارامتر مقداری data مطابقت می کند. بخش set نیز مقدار هر یک از مقادیر ورودی آرایه را که مقدارش با پارامتر data مطابقت نماید را به مقداری که به Indexer تخصیص داده می شود، تغییر می دهد.

```
class OverIndexer
{
    private string[] myData;
    private int arrSize;
    public OverIndexer(int size)
    {
        arrSize = size;
        myData = new string[size];
        for (int i = 0; i < size; i++)
        {
            myData[i] = "EMPTY";
        }
    }
    public string this[int pos]
    {
        get
        {
            return myData[pos];
        }
        set
        {
            myData[pos] = value;
        }
    }
    public string this[string data]
    {
        get
        {
            int count = 0;
            for (int i = 0; i < arrSize; i++)
            {
                if (myData[i] == data)
                {
                    count++;
                }
            }
            return count.ToString();
        }
        set
        {
            for (int i = 0; i < arrSize; i++)
            {
                if (myData[i] == data)
                {
                    myData[i] = value;
                }
            }
        }
    }
}
```

پس در متد اصلی کلاس program.cs دستورات زیر را نوشته و برنامه را اجرا کنید.

```
static void Main(string[] args)
{
    int size = 10;
    OverIndexer myInd = new OverIndexer(size);
    myInd[9] = "Some Value";
    myInd[3] = "Another Value";
    myInd[5] = "Any Value";
    myInd["EMPTY"] = "No Value";
    Console.WriteLine("\nIndexer Output\n");
    for(int i=0;i<size;i++)
    {
        Console.WriteLine("myInd[{0}]:{1}",i,myInd[i]);
    }
    Console.WriteLine("\nNumber of \"No Value\" entries : {0}",myInd["No Value"]);
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

رفتار Indexer سرریز شده که پارامتری از نوع string دریافت می کند، در متد Main() نشان داده شده است. در بخش دستیاب set مقدار No Value را به تمام اعضای کلاس myInd مقدارشان برابر EMPTY بوده است، تخصیص می دهد. این دستیاب از دستور مقابل استفاده نموده است : myInd ["EMPTY"] = "No Value";

پس از اینکه تمام اعضای کلاس myInd چاپ شدند، تعداد اعضای که حاوی No Value بوده اند نیز نمایش داده می شوند. این امر با استفاده از دستور دستیاب get مقابل روی می دهد myInd ["No Value"]; خروجی برنامه به شکل زیر خواهد بود.

Indexer Output

```
myInd[0]:No Value
myInd[1]:No Value
myInd[2]:No Value
myInd[3]:Another Value
myInd[4]:No Value
myInd[5]:Any Value
myInd[6]:No Value
myInd[7]:No Value
myInd[8]:No Value
myInd[9]:Some Value
```

```
Number of "No Value" entries : 7
Press any key to continue . . .
```

علت همزیستی هر دو Indexer در یک کلاس مشابه، تفاوت اثر گذاری و فعالیت آنهاست. اثر گذاری و تفاوت Indexer ها از تعداد و نوع پارامتر های موجود در لیست پارامتر های Indexer مشخص می شود. در هنگام استفاده از Indexer ها نیز، کلاس با استفاده از تعداد و

نوع پارامترهای Indexer ها می تواند تشخیص دهد که در یک فراخوانی از کدام Indexer باید استفاده نماید. نمونه ای از پیاده سازی Indexer با چند نوع پارامتر در زیر آورده شده است :

```
public object this [int param1 , ... , paramN]
{
    get
    {
        // process and return some class data
    }
    set
    {
        // process and assign some class data
    }
}
```

انواع Indexer ها

Indexer های خواندنی و نوشتنی : می توان از یک Indexer در حالت خواندن / نوشتن استفاده کرد. در این صورت باید از هر دو متد set و get استفاده کرد. Indexer های فقط خواندنی : می توانید یک Indexer تعریف کنید که فقط متد get داشته باشد. در این صورت، فقط می توانید از Indexer در حالت خواندن استفاده کنید. برای مثال، IntBits در کد زیر یک Indexer فقط خواندنی دارد.

```
class IntBits
{
    ...
    public bool this [ int index ]
    {
        get
        {
            return (bits & (1 << index)) !=0;
        }
    }
    ...
    private int bits;
}
```

این Indexer، متد set ندارد. بنابراین اگر بخواهید دستور زیر را در آن بنویسید یک خطای زمان کامپایل دریافت خواهید کرد.

```
bits[6] = false; // compile-time error
```

Indexer های فقط - نوشتنی : همچنین می توانید یک Indexer تعریف کنید که فقط متد set داشته باشد. در این صورت فقط می توانید از آن در حالت نوشتن استفاده نمایید. برای مثال، IntBits در کد زیر یک Indexer فقط نوشتنی دارد.

```
class IntBits
{
    ...
    public bool this [ int Index ]
    {
        set
        {
            if (value)
                bits |= (1 << Index);
            else
                bits &= ~ (1 << Index);
        }
    }
    ...
    private int bits;
}
```

این Indexer متد get ندارد. بنابراین اگر می خواهید آن را بخوانید، یک خطای زمان کامپایل دریافت خواهید کرد.

```
Console.WriteLine (bits[6]); // compile-time error
bits[6] ^=false;             // compile-time error
```

مثال عملی استفاده از Indexer ها

یک نمونه بسیار جالب از استفاده Indexer ها کنترل ListBox است. (ListBox عنصری است کنترلی که با استفاده از آن لیستی از عناصر رشته ای نمایش داده می شوند و کاربر با انتخاب یکی از این گزینه ها با برنامه ارتباط برقرار می کند. در حقیقت این عنصر کنترلی یکی از روش های دریافت اطلاعات از کاربر است با این تفاوت که در این روش ورودی هایی که کاربر می تواند وارد نماید محدود شده هستند و از قبل تعیین شده اند. نمونه ای از یک ListBox قسمت انتخاب نوع فونت در برنامه Word است که در آن لیستی از فونت های موجود در سیستم نمایش داده می شود و کاربر با انتخاب یکی از آنها به برنامه اعلام می کند که قصد استفاده از کدام فونت سیستم را دارد.) ListBox نمایشی از ساختمان داده ای است شبیه به آرایه که اعضای آن همگی از نوع string هستند. علاوه بر این این کنترل می خواهد تا در هنگام انتخاب یکی از گزینه های خود بتواند اطلاعات را به صورت خودکار بروز نماید

و یا به عبارتی بتواند ورودی دریافت کند. تمامی این اهداف با استفاده از Indexer امکان پذیر است. Indexer شبیه خاصیت ها اعلان می شوند، با این تفاوت مهم که Indexer بدون نام هستند و نام آنها تنها کلمه کلیدی this است و همین this مورد Index شدن قرار می گیرد و سایر موارد به صورت پارامتر به Indexer داده می شوند.

چند ریختی واسطه ای (Interface)

اکنون که با چند ریختی وراثتی آشنا شده ایم، درک چند ریختی Interface کار بسیار ساده ای است. در حقیقت چند ریختی Interface ای همان چند ریختی وراثتی است با این تفاوت که در این چند ریختی، به جای کلاس پایه، یک Interface داریم.

به بیان ساده می توان گفت که Interface یک قرارداد در نوع رفتار یک کلاس است. هر کلاسی که یک Interface را به کار ببرد، در حقیقت تضمین می کند پیاده سازی تمام متد ها، خاصیت ها و ... که در آن Interface وجود دارد را در خود ایجاد کند.

Interface ها از لحاظ ظاهری بسیار شبیه به کلاس هستند با این تفاوت که دارای هیچ گونه پیاده سازی نمی باشند. تنها چیزی که در Interface قابل ملاحظه است تعاریف مختلف مانند رویداد ها، خاصیت ها و موارد دیگر است. یکی از دلایل اینکه Interface ها دارای تعاریف هستند و پیاده سازی ندارد آن است که یک Interface می تواند توسط چندین کلاس یا خاصیت مورد ارث بری قرار گیرد، از این رو هر کلاس یا خاصیت خواستار آن است که خود به پیاده سازی اعضا پردازد.

نحوه تعریف یک Interface مانند یک کلاس است با این تفاوت که به جای واژه Class از واژه interface استفاده می شود. در ضمن برای ایجاد یک Interface در یک پروژه، بعد از انتخاب گزینه های Add و سپس New Item گزینه Interface انتخاب شود. و نیز متداول است که نامگذاری Interface ها با i شروع شود.

interface Isum سطح دسترسی

```
{  
;دستورات  
}
```

اعضای تعریف شده در Interface نمی توانند سطح دسترسی داشته باشند ولی به طور تلویحی public هستند.

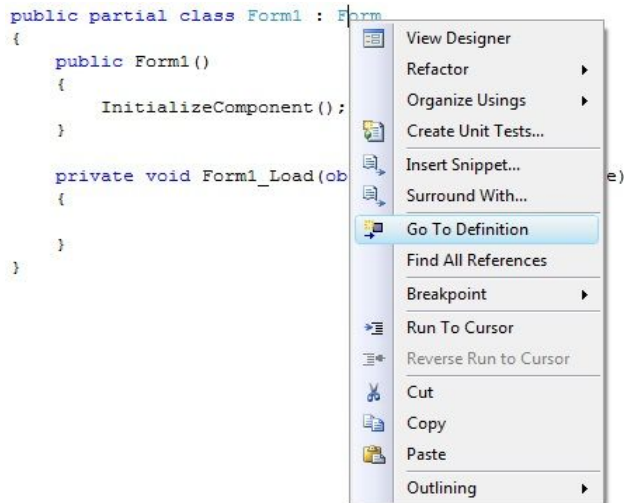
کلاس هایی که مستقیماً از Interface ارث می برند، باید تمام اعضای آن را پیاده سازی کنند. و این پیاده سازی باید با سطح دسترسی public انجام شود در صورتی که در پیاده سازی متد های کلاس abstract، چنین محدودیتی وجود ندارد (محدودیت سطح دسترسی public وجود نداشت هر چند پیاده سازی متد abstract توسط کلاس ارث برنده از کلاس abstract الزامی بود). با توجه به موضوع public بودن سطح دسترسی اعضا در Interface و اعضای آن در کلاس ارث برنده، به نظر می رسد که بهتر است متد ها و اعضای در Interface قرار گیرند که علاوه بر کلاس های ارث برنده، اشیای کلاس ها نیز به آنها نیاز داشته باشند.

از Interface نمی توان شیء ایجاد کرد، بلکه از کلاس های پیاده ساز آن، همانند کلاس های abstract می توان شیء ایجاد نمود.

قوانین ارث بری مانند فرزند و پدر، چند ریختی و ... به طور مشابه در اینجا نیز وجود دارند. کلاس های پیاده ساز Interface که از آن ارث می برند در حکم فرزندان آن هستند. برای مثال زمانی که متغیری از نوع یک Interface تعریف می کنید، آن متغیر را با هر کدام از اشیای کلاس هایی که آن Interface را پیاده سازی می کنند می توانید ایجاد کنید.

حال باید دید چرا با توجه به اینکه Interface ها دارای پیاده سازی نیستند مورد استفاده قرار می گیرند؟ تصور کنید که در یک برنامه با مولفه هایی سر و کار دارید که متغیرند ولی دارای فیلد ها یا متد هایی با نام های یکسان هستند و باید نام این متد ها نیز یکسان باشد. با استفاده از یک Interface مناسب می توان تنها متد ها و یا فیلد های مورد نظر را اعلان نمود و سپس کلاس ها و یا خاصیت های مورد نیاز از آن Interface ارث بری نمایند. در این حالت تمامی کلاس ها و خاصیت ها دارای فیلد ها و یا متد هایی همان هستند ولی هر یک پیاده سازی خاصی از آنها را اعمال می نمایند.

چند مورد از استفاده Interface ها را در محیط ویژوال استودیو بررسی می کنیم. در بخش دستورات Form به بخش Form : Form\ public partial class در روی دستور Form راست کلیک کرده و گزینه Go To Definition را کلیک کنید (پنجره تصویر ۵۱ - ۵).



تصویر ۵۱ - ۵

دستورات زیر ظاهر می شوند.

```
...public class Form : ContainerControl
{
    ...public Form();

    ...public IButtonControl AcceptButton { get; set; }
    ...public static Form ActiveForm { get; }
    ...public Form ActiveMdiChild { get; }
    ...public bool AllowTransparency { get; set; }
    ...public bool AutoScale { get; set; }
    ...public virtual Size AutoScaleBaseSize { get; set; }
    ...public override bool AutoScroll { get; set; }
    ...public override bool AutoSize { get; set; }
    ...public AutoSizeMode AutoSizeMode { get; set; }
    ...public override AutoValidate AutoValidate { get; set; }
    ...public override Color BackColor { get; set; }
    ...public IButtonControl CancelButton { get; set; }
    ...public Size ClientSize { get; set; }
    ...public bool ControlBox { get; set; }
    //
}
```

بر روی دستور ContainerControl راست کلیک کرده و دوباره گزینه Go To Definition را کلیک کنید.

```
...public class ContainerControl : ScrollableControl, IContainerControl
{
    ...public ContainerControl();
}
```

interface System.Windows.Forms.IContainerControl
Provides the functionality for a control to act as a parent for other controls.

در دستورات ظاهر شده در بالا به دستور IContainerControl و پنجره راهنمای ظاهر شده توجه کنید که نشان می دهد این دستور یک Interface است. روی این دستور هم راست کلیک کرده و گزینه Go To Definition را کلیک کنید تا دستورات زیر باز شود.

```
...public interface IContainerControl
{
    ...Control ActiveControl { get; set; }

    ...bool ActivateControl(Control active);
}
```

همانطور که در دستورات بالا مشاهده می کنید دو متد درون این Interface تعریف شده است که از نوع های Control و bool می باشد. در زیر یک مثال از Interface آورده شده است.

```
interface Icol
{
    string Funny
    {
        get;
    }
    void Honk();
}
```

همانطور که می بینید یک Interface نام Icol تعریف کرده ایم، این Interface دارای متد honk و دارای خصوصیت funny است. هم متد و هم خصوصیت فاقد بدنه هستند. باید کلاسی تعریف کنیم که Interface بالا را ارث بری کند و متد ها و خصوصیات در آن اعمال شوند. در این مثال کلاس مورد نظر علاوه بر ارث بری از Interface، متد ها و خصوصیت های آن را با بدنه دوباره تعریف می کند.

```

class TallGuy : Icol
{
    public string Name;
    public int Height;
    public void TalkAboutYourself()
    {
        MessageBox.Show("My name is " + Name + " and I'm " + Height + " inches tall.");
    }
    public string Funny
    {
        get
        {
            return "big shoes";
        }
    }
    public void Honk()
    {
        MessageBox.Show("Honk honk!");
    }
}

```

ارث بری چندگانه از Interface ها

در زبان C# ارث بری چندگانه برای کلاس ها موجود نمی باشد ولی برای Interface ها موجود است و به طور کلی یک کلاس می تواند از بیش از یک Interface ارث بری کند. در این حالت کلاس ارث برنده باید تمام اعضای Interface های ارث برده شده را پیاده سازی کند. برای ارث بردن از بیش از یک Interface نام Interface های بعدی با کاما (,) از هم جدا می شوند.

فرض کنید دو Interface به نام های Interface۱ و Interface۲ به صورت زیر داریم :

```

public interface Interface1
{
    void Method1();
}

public interface Interface2
{
    void Method2();
}

```

در نظر بگیرید که کلاس PClass۱ علاوه بر Interface۱ از Interface۲ نیز ارث می برد.

```

public class PClass1 : Interface1 , Interface2
{
    //...
}

```

در این حالت بایستی اعضای Interface^۲ نیز در کلاس PClass^۱ پیاده سازی شوند. پس پیاده سازی این متد به کلاس PClass^۱ اضافه می شود :

```
public void Method2()
{
    int i = 0;
    //...
}
```

در اینجا کلاس PClass^۱ فرزند Interface^۱ و Interface^۲ محسوب می شود و قوانین سلسله مراتب کلاس پایه و مشتق شده نیز به همان شکل در مورد آنها صادق است. برای مثال متغیری که از Interface^۲ تعریف می شود نیز می تواند با شیء ای از کلاس PClass ایجاد شود :

```
Interface1 c1;
Interface2 c2;
c1 = new PClass1();
c2 = new PClass2();
```

همان گونه که کد بالا نشان می دهد متغیرهای تعریف شده از Interface^۱ و Interface^۲ هر دو با شیء کلاس فرزند PClass^۱ می توانند ایجاد شوند.

نکته مهمی که می توان در اینجا به آن اشاره کرد این است که یک کلاس می تواند از یک کلاس و چند Interface ارث بری کند. برای نمونه، در مثال قبلی فرض کنید کلاس PClass^۱ علاوه بر دو Interface ذکر شده از کلاسی با نام کلاس Class^۰ نیز ارث می برد :

```
public class PClass1 : Class0(), Interface1, Interface2
{
    //...
}
```

در این حالت کلاس PClass^۱ فرزند کلاس Class^۰، Interface^۱ و Interface^۲ می باشد.

ترکیب Interface ها

یک Interface می تواند از چند Interface ارث ببرد و می تواند متد های جدیدی نیز تعریف کند که کلاس ارث برده باید تمام متد ها اعم از متد های ارث برده و جدید را پیاده سازی کند.

برای مثال فرض کنید Interface جدیدی به نام Interface۳ تعریف کرده ایم که از Interface۱ و Interface۲ ارث برده است و متد جدیدی به نام Method۳ را نیز اضافه کرده است :

```
public interface Interface3 : Interface1 , Interface2
{
void Method3();
}
```

در این حالت اگر کلاسی از Interface۳ ارث ببرد برای مثال کلاس Class۱، باید تمام متد های Interface۱، Interface۲، Interface۳ را نیز پیاده سازی کند و شیء این کلاس نیز فرزند Interface های مذکور می باشد.

مثال کاربردی پیاده سازی Interface و ارث بری

قبل از اینکه شروع به بررسی و پیاده سازی یک مثال واقعی بپردازیم، بهتر است به بررسی چند نکته مهم بپردازیم :

- ۱- با استفاده از کلمه کلیدی interface در واقع یک نوع مرجع جدید ایجاد می کنید.
- ۲- هدف از ایجاد یک Interface تعیین توانایی هایی است که می خواهیم در یک کلاس وجود داشته باشد.

فرض کنید می خواهید Interface ای ایجاد کنید که متد ها و خاصیت های لازم برای کلاسی که قابلیت خواندن و نوشتن از یک پایگاه داده یا فایل را داشته باشد، توصیف نماید. برای این منظور می توانید از Interface IStorable استفاده نمایید. در این Interface دو متد Read() و Write() وجود دارند که در بدنه Interface تعریف می شوند.

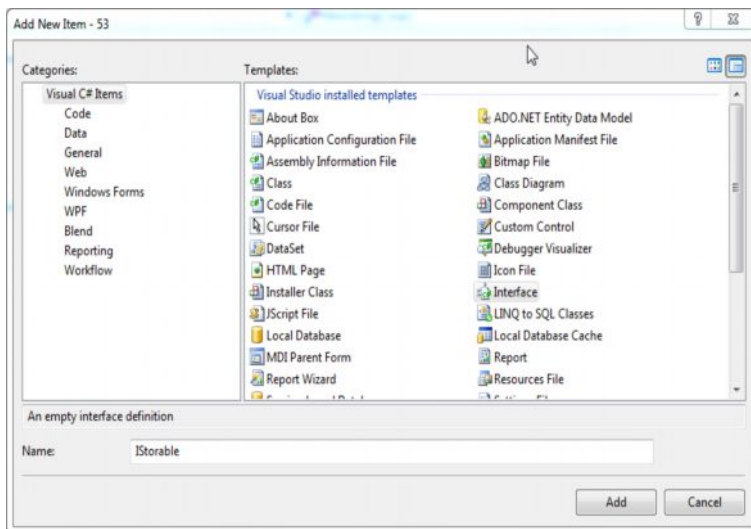
```
interface IStorable
{
void Read();
void write (object);
}
```

حال اگر می خواهید کلاسی با عنوان Document ایجاد نمایید که این کلاس باید قابلیت خواندن و نوشتن از پایگاه داده را داشته باشد، لذا می توانید از روی کلاس IStorable پیاده سازی کنید.

```
public class Document : IStorable
{
public void Read()
{
...
}
public void write (object obj)
{
...
}
//...
}
```

به عنوان یک برنامه نویس، شما وظیفه دارید تا به پیاده سازی این Interface بپردازید. به طوری که نیازهای شما را برآورده نماید. نمونه ای از این پیاده سازی در برنامه زیر آورده شده است.

ابتدا یک پروژه از نوع Console ایجاد کنید سپس در پنجره Solution Explorer بر روی پروژه خود راست کلیک کرده و گزینه Add و سپس New Item را انتخاب کنید تا پنجره New Item مطابق پنجره تصویر ۵۲ - ۵ باز شود.



تصویر ۵۲ - ۵

در بخش Name عنوان Interface مورد نظر خود را نوشته و روی دکمه Add کلیک کنید ما در اینجا عنوان Interface خود را IStorable تعیین کرده ایم. توجه کنید که می توانید شیء ای از نوع Class نیز ایجاد کرده و در آن Interface مورد نظر خود را پیاده سازی کنید ولی

از لحاظ اصول کاری این روش بهتر است.

```
interface IStorable
{
    void Read();
    void Write(object obj);
    int Status
    {get;set;}
}

public class Document : IStorable
{
    public Document(string s)
    {
        Console.WriteLine("Creating Document with : {0}", s);
    }
    public void Read()
    {
        Console.WriteLine("Implementing the Read Method for IStorable");
    }
    public void Write(object o)
    {
        Console.WriteLine("Implementing the Write Method for IStorable");
    }
    public int Status
    {
        get { return status; }
        set { status = value; }
    }
    private int status = 0;
}

static void Main(string[] args)
{
    Document doc = new Document("Test Documet");
    doc.Status = -1;
    doc.Read();
    Console.WriteLine("Document Status : {0}", doc.Status);
    IStorable isDoc = (IStorable)doc;
    isDoc.Status = 0;
    isDoc.Read();
    Console.WriteLine("IStorable Status : {0}", isDoc.Status);
}
```

در صورتی که برنامه را اجرا کنید، خروجی برنامه به صورت زیر خواهد بود.

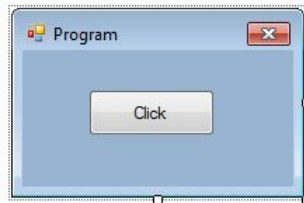
```
Creating Document with : Test Documet
Implementing the Read Method for IStorable
Document Status : -1
Implementing the Read Method for IStorable
IStorable Status : 0
Press any key to continue . . .
```

همچنان که در برنامه فوق مشاهده می کنید برای متد های Interface IStorable هیچ سطح دسترسی در نظر گرفته نشده است. در حقیقت تعیین سطح دسترسی باعث ایجاد خطا می شود چرا که هدف اصلی از ایجاد یک Interface ایجاد شیء است که تمامی اعضای آن برای

تمامی کلاس ها قابل دسترسی باشند.

مثال ۳۵ - ۵: برنامه زیر نحوه استفاده از Interface و ارث بری از آن را تحت یک پروژه ویندوزی نشان می دهد.

حل : ابتدا یک پروژه ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۵۳ - ۵ طراحی کنید.



تصویر ۵۳ - ۵

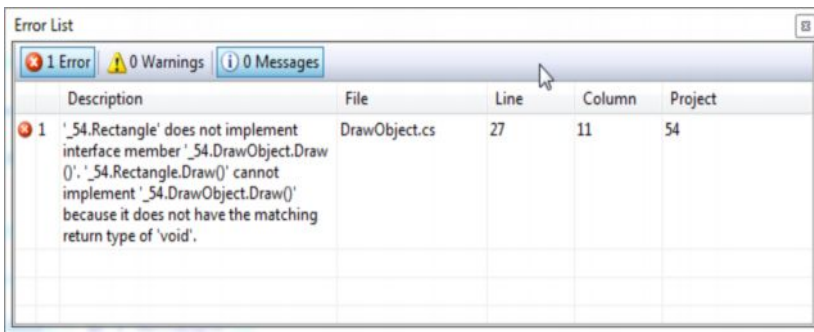
یک شیء Interface مطابق روشی که گفته شد ایجاد کرده و عنوان آن را DrawObject تعیین نمایید. توجه داشته باشید که در بخش using دستور using System.Windows.Forms; را اضافه کنید چون می خواهیم از دستور MessageBox استفاده کنیم.

```
interface DrawObject
{
    void Draw();
}
class Line : DrawObject
{
    public void Draw()
    {
        MessageBox.Show("I am a Line");
    }
}
class Circle : DrawObject
{
    public void Draw()
    {
        MessageBox.Show("I am a Circle");
    }
}
class Rectangle : DrawObject
{
    public void Draw()
    {
        MessageBox.Show("I am a Rectangle");
    }
}
```

همانطور که در دستورات بالا مشاهده می کنید یک Interface به نام DrawObject تعریف کرده ایم که شامل متد Draw است و هیچ پیاده سازی ندارد. سه کلاس Line, Circle, Rectangle از آن ارث می برند. هر کدام از کلاس ها دارای یک متد هستند که کار خاصی را انجام می دهند. حالا در Form برنامه بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    Line ln = new Line();
    ln.Draw();
    Circle cl = new Circle();
    cl.Draw();
    Rectangle re = new Rectangle();
    re.Draw();
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید. کافی است که در یکی از متد های سه کلاس تعریف به جای void نوع int را در نظر بگیرید، خطای زیر تولید می شود (پنجره تصویر ۵۴ - ۵).



تصویر ۵۴ - ۵

مفهوم این خطا بدین معناست که شما از چند ریختی نوع Interface استفاده می کنید شما در حال استفاده از یک نوع بازگشتی به صورت اشتباه هستید. زیرا بایستی تمام متد های تعریف شده درون کلاس ها هم نوع و هم نام با متد تعریف شده در بخش Interface باشد. در بخش ارث بری گفتیم که در زبان C# بیش از یک کلاس نمی توانیم ارث بری داشته باشیم ولی این مورد درباره ی Interface ها صادق نیست. می توانیم بیش از یک Interface ارث

بری داشته باشیم. در همین برنامه یک Interface جدید به نام DrawObject۲ تعریف می کنیم که کلاس Line از آن ارث بری می کند، این کار به آسانی امکان پذیر است.

```
interface DrawObject
{
    void Draw();
}
interface DrawObject2
{
    void Draw2();
}
class Line : DrawObject , DrawObject2
{
    public void Draw()
    {
        MessageBox.Show("I am a Line");
    }
    public void Draw2()
    {
        MessageBox.Show("I am a Line2");
    }
}
```

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    Line ln = new Line();
    ln.Draw();
    ln.Draw2();
    Circle cl = new Circle();
    cl.Draw();
    Rectangle re = new Rectangle();
    re.Draw();
}
```

تبدیل نوع به Interface

به طور مشابه در ارتباط با قوانینی که در مورد کلاس ها داشتیم، هر کدام از کلاس های پیاده ساز Interface یا فرزندان آنها می توانند به آن Interface، تبدیل شوند. در مورد عملگر is و کارکرد آن قبلا صحبت کردیم. به دستور زیر توجه کنید.

```
Interface1 c1;
c1 = new PClass1();
c1.Method1();
if (c1 is Interface2)
{
    (c1 as Interface2).Method2();
}
```

در دستورات بالا متغیر c1 از Interface1 تعریف شده است و با شیء ای از کلاس یکی از فرزندان آن یعنی PClass ایجاد شده است. از آنجا که متغیر از نوع پدر (Interface1) تعریف شده است در ویرایشگر کد تنها متدهای آن یعنی Method1 را نشان می دهد. فراخوانی آن به صورت زیر است :

```
c1.Method1();
```

اگر قصد داشته باشید Method2 را نیز فراخوانی کنیم، این کار با تبدیل نوع آن به Interface2 یا کلاس PClass1 امکان پذیر است که در اینجا به Interface2 تبدیل نوع شده است :

```
if (c1 is Interface2)
{
(c1 as Interface2).Method2();
}
```

نظربه اینکه در Interface پیاده سازی متد وجود ندارد، فراخوانی متد از کلاس PClass1 انجام می شود. در دستورات بالا نوع شیء c1 با عملگر is چک شده است. با توجه به اینکه نوع شیء در اینجا مشخص بوده، استفاده از عملگر is در این دستورات الزامی نبوده و به عنوان مثال ذکر شده است.

نکته!

به طور پیش فرض متدی که متد interface را پیاده سازی می کند، virtual نیست. اگر بخواهید که متد کلاس را در کلاس مشتق شده از آن بار گذاری کنید، باید متد را به صورت virtual تعریف کنید. برای مثال به دستورات زیر توجه کنید.

در این مثال این امر دقیقاً با این واقعیت تطابق دارد که کلمه کلیدی virtual، اولین پیاده سازی متد را تعریف می کند و پیاده سازی های بعدی در کلاس های مشتق شده انجام می شوند و متد غیر virtual، تنها پیاده سازی یک متد است.

```

interface IToken
{
    string Name();
}
class Token : IToken
{
    ...
    public virtual string Name()
    {
        ...
    }
}
class IdentifierToken : Token
{
    ...
    public override string Name()
    {
        ...
    }
}

```

یک کلاس می تواند کلاس دیگری را گسترش دهد و یک interface را پیاده سازی کند. در این مورد زبان C# با استفاده از کلمات کلیدی، بین کلاس پایه و interface پایه فرق قائل نمی شود بلکه از نظر ترتیب فرق قائل می شود. ابتدا اسم کلاس پایه، بعد یک کاما و بعد interface پایه نوشته شود. برای مثال دستورات زیر را در نظر بگیرید :

```

interface IToken
{
    ...
}
class DefaultTokenImpl
{
    ...
}
class IdentifierToken : DefaultTokenImpl , IToken
{
    ...
}

```

پیاده سازی صریح (Explicit Interface Implementation)

روش دیگری که یک کلاس بتواند یک متد از Interface را پیاده سازی کند وجود دارد. در این روش قبل از اسم متد، از اسم interface بایستی استفاده کرد و از مشخصه دستیابی استفاده نمی شود. برای مثال دستورات زیر را در نظر بگیرید :

```
interface Visitable
{
void Accept (Visitor visitor);
}
```

می توانید به صورت زیر متد interface را پیاده سازی کنید.

```
class IdentifierToken : DefaultTokenImpl , IToken , Visitable
{
...
void Visitable.Accept (Visitor visitor)
{
...
}
}
```

روش EII (Explicit Interface Implementation) پیاده سازی صریح interface نامیده می شود. متد EII برای پیاده سازی کلاس، private است. به مثال زیر توجه کنید.

```
IdentifierToken token = new IdentifierToken ("token");
token.Accept(visitor); //compile-time error : not accessible
```

با این حال متد می تواند از طریق یک interface که اسم آن صریحاً مشخص شده است و با استفاده از یک تبدیل نوع، فراخوانی شود. به مثال زیر توجه کنید.

```
IdentifierToken token = new IdentifierToken ("token");
((Visitable)token).Accept(visitor); //okay
```

بنابراین متد EII، نه کاملاً private و نه کاملاً public است. به این دلیل از یک مشخصه دستیابی استفاده نمی کنیم.

متد EII از دو جهت سودمند است :

۱- متد EII با روش بهتری یک interface را که چگونگی استفاده از یک object را تعریف می کند از یک کلاس که چگونگی ایجاد و پیاده سازی object را تعریف می کند جدا می کند. از آنجایی که متد های EII برای کلاس، private هستند تنها راه فراخوانی آنها استفاده از یک object از طریق interface آن است.

۲- وقتی که اسم متد ها در interface های مختلف با هم یکی باشند و ایجاد مشکل کنند، متد های EII می توانند این مشکل را حل کنند. برای مثال، فرض کنید که IToken و IVisitable هر دو یک متد به نام ToString دارند :

```
interface IToken
{
    ...
    string ToString();
}
interface Visitable
{
    ...
    string ToString();
}
```

کلاس زیر یک متد ToString را تعریف می کند که پیاده سازی دو متد بالا است.

```
class IdentifierToken : IToken , Visitable
{
    ...
    public string ToString()
    {
        ...
    }
}
```

EII راهی برای ایجاد پیاده سازی های مختلف از ToString را ارائه می کند که برای هر متد یک پیاده سازی انجام می شود. به مثال زیر توجه کنید.

```
class IdentifierToken : IToken , Visitable
{
    ...
    string IToken.ToString()
    {
        ...
    }
    string Visitable.ToString()
    {
        ...
    }
}
```

توجه داشته باشید که متد EII، virtual نیست و نمی تواند virtual شود. بنابراین نمی توان آن را در کلاس مشتق شده override کرد.

مثال ۳۶ - ۵: برنامه زیر یک کلاس را نشان می دهد که می تواند با یک متد Interface به صورت صریح پیاده سازی شود. در این رابطه متد Interface تنها می تواند از طریق شیء Interface دستیابی شود.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و یک شیء Interface ایجاد می کنیم و عنوان آن را I_baseInterface تعیین می کنیم.

```

interface I_baseInterface
{
    void I_F_Function_01();
    void I_F_Function_02();
}
public class CL_baseClass : I_baseInterface
{
    public void I_F_Function_01()
    {
    }
    //Explicit implementation of an Interface method
    void I_baseInterface.I_F_Function_02()
    {
        Console.WriteLine("I am a Student");
    }
}

```

اکنون در متد اصلی کلاس program.cs دستورات زیر را بنویسید. توضیحات لازم به صورت Comment در داخل برنامه آورده شده است.

```

static void Main(string[] args)
{
    //create a base Class object
    CL_baseClass o_baseClassObject = new CL_baseClass();
    /*Following is ERROR: I_F_Function_02 has been implemented explicitly within Class and
    it is not accesible via Class object */
    // o_baseClassObject.I_F_Function_02();
    //Following is OK:
    //create an Interface Object instance to assign base Class Object
    I_baseInterface iObj_exampleInterfaceObject = o_baseClassObject;
    iObj_exampleInterfaceObject.I_F_Function_02();
}

```

برنامه را اجرا کرده و نتیجه را بررسی کنید.

تعریف خاصیت ها در یک Interface

می توان برای interface هم خاصیت تعریف کرد. برای انجام این کار از کلمات کلیدی get و set می توان استفاده کرد. ولی به جای بدنه متد های set و get از سمی کالن (;) استفاده کنید. برای مثال به دستورات زیر توجه کنید.

```

interface IScreenPosition
{
int X {get; set;}
int Y {get; set;}
}
class ScreenPosition : IScreenPosition
{
...
public int X
{
get {return x;}
set {X = rangeCheckedX(value);}
}
public int Y
{
get {return y;}
set {Y = rangeCheckedY(value);}
}
...
private int x,y;
}

```

اگر خاصیت های interface را در یک کلاس پیاده سازی کنید می توانید پیاده سازی های خاصیت را به صورت virtual تعریف کنید که این امر به کلاس های مشتق شده اجازه می دهد تا این پیاده سازی ها را override کنند. برای مثال به دستورات زیر توجه کنید :

```

class ScreenPosition : IScreenPosition
{
...
public virtual int X
{
get {...}
set {...}
}
public virtual int Y
{
get {...}
set {...}
}
...
private int x,y;
}

```

همچنین می توانید با استفاده از روش پیاده سازی صریح Interface، خاصیت ها را پیاده سازی کنید. پیاده سازی صریح یک خاصیت غیر public و غیره virtual است و نمی توان آن را override کرد. برای مثال به دستورات زیر توجه کنید :

```

class ScreenPosition : IScreenPosition
{
    ...
    int IScreenPosition.X
    {
        get { return x;}
        set {x = rangeCheckedX(value);}
    }
    int IScreenPosition.Y
    {
        get {return y;}
        set { y = rangeCheckedY(value);}
    }
    ...
    private int x,y;
}

```

می توانید Interface ها را به صورت خاصیت های فقط خواندنی یا فقط نوشتنی تعریف کنید. حتی می توانید متد های set و get را بین Interface های مختلف تقسیم نمایید.

مثال ۳۷ - ۵: برنامه زیر نحوه استفاده از خاصیت ها را در یک Interface نشان می دهد.

حل : یک پروژه از نوع Console ایجاد کرده و یک شیء از نوع Interface ایجاد کرده و عنوان آن را IValue تعیین کنید و سپس دستورات زیر را در آن بنویسید.

```

interface IValue
{
    int Count { get; set; } // Property interface
    string Name { get; set; } // Property interface
}
class Image : IValue // Implements interface
{
    public int Count // Property implementation
    {
        get;
        set;
    }
    string _name;
    public string Name // Property implementation
    {
        get { return this._name; }
        set { this._name = value; }
    }
}
class Article : IValue // Implements interface
{
    public int Count // Property implementation
    {
        get;
        set;
    }
    string _name;
}

```

```

public string Name // Property implementation
{
    get { return this._name; }
    set { this._name = value.ToUpper(); }
}
}

```

در متد اصلی کلاس program دستورات زیر را اضافه نمایید.

```

static void Main(string[] args)
{
    IValue value1 = new Image();
    IValue value2 = new Article();
    value1.Count++; // Access int property on interface
    value2.Count++; // Increment
    value1.Name = "Reza bahrāmīrad"; // Use setter on interface
    value2.Name = "Resignation"; // Set
    Console.WriteLine(value1.Name); // Use getter on interface
    Console.WriteLine(value2.Name); // Get
}

```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

Interface های Indexer

در داخل یک Interface نیز می توان Indexer تعریف کرد. برای انجام این کار باید از کلمات کلیدی get و set استفاده کرد. ولی بایستی به جای بدنه get و set از یک سمی کالن (;) استفاده کرد. هر کلاسی که از این Interface ارث می برد باید متد های دستیابی get و set را نیز پیاده سازی کند. برای مثال دستورات زیر را در نظر بگیرید.

```

interface IRawInt
{
    bool this [int index]
    {get; set;}
}
class RawInt : IRawInt
{
    ...
    public bool this [int index]
    {
        get {...}
        set {...}
    }
    ...
}

```

اگر یک Indexer از Interface را در یک کلاس پیاده سازی می کنید می توانید این پیاده سازی را به صورت virtual تعریف کنید. این کار به کلاس های مشتق شده امکان می دهد تا متد های get و set را override کنند.

برای مثال دستورات زیر را در نظر بگیرید.

```
class RawInt : IRawInt
{
...
public virtual bool this [int index]
{
get {...}
set {...}
}
...
}
```

همچنین می توانید Indexer را با استفاده از روش پیاده سازی صریح Interface پیاده سازی کنید. پیاده سازی صریح یک Indexer، public و virtual نیست، بنابراین نمی توان آن را override کرد. به مثال زیر توجه کنید :

```
class RawInt : IRawInt
{
...
bool IRawInt.this [int index]
{
get {...}
set {...}
}
...
}
```

همچنین می توان Indexer هایی در یک Interface تعریف کرد که فقط خواندنی و یا فقط _نوشتنی باشند. حتی می توانید متد های دستیابی get و set را بین Interface های مختلف تقسیم کنید.

مثال ۳۸ – ۵: برنامه زیر نحوه استفاده از Indexer ها را روی Interface ها نشان می دهد؟
حل : ابتدا یک پروژه از نوع Console ایجاد کنید و سپس یک شیء به نام Interface ایجاد کرده و عنوان آن را IPerl قرار دهید. سپس دستورات زیر را در آن بنویسید.

```

interface IPerl
{
    int this[int number] { get; set; }
}
class Implementation : IPerl
{
    int[] _data = { 0, 10, 20, 30 }; // Default values
    public int this[int number]
    {
        get
        {
            // Get accessor implementation.
            return this._data[number];
        }
        set
        {
            // Set accessor implementation.
            this._data[number] = value;
        }
    }
}

```

در کلاس program.cs و در متد اصلی برنامه دستورات زیر را بنویسید.

```

static void Main(string[] args)
{
    // Create an object instance.
    // ... Use the indexer through the interface type.
    IPerl perl = new Implementation();
    Console.WriteLine(perl[0]);
    Console.WriteLine(perl[1]);
    Console.WriteLine(perl[2]);
    Console.WriteLine(perl[3]);
    // Use set accessor.
    perl[0] = -1;
    Console.WriteLine(perl[0]);
}

```

در پایان برنامه را اجرا کرده و نتیجه را بررسی کنید.

آشنایی با Delegate ها

همانطور که خاصیت یک فیلد هوشمند و Indexer یک آرایه هوشمند است، Delegate هم یک متد هوشمند است. Delegate ها برای ایجاد Framework های اعلان (Notification) و Callback بسیار مفید هستند. بدین معنا که یک کلاس می تواند به کلاس دیگر بگوید که (عمل خاصی را انجام بده و پس از انجام آن مرا نیز باخبر کن). با استفاده از Delegate ها، همچنین می توان متد هایی تعریف کرد که تنها در زمان اجرا قابل دسترسی باشند. Delegate ها، یکی دیگر از انواع مرجع زبان C# هستند که با استفاده از آنها می توانید مرجعی به یک

متد داشته باشید. بدین معنا که Delegate ها، آدرس متدی خاص را در خود نگه می دارند. اشاره گرها در زبان C مشابه Delegate ها در زبان C# هستند. اما برای افرادی که با مفهوم اشاره گرها آشنا نیستند، شاید این سوال در ذهن آنها بوجود می آید که چه نیازی به دانستن آدرس یک متد وجود دارد؟ برای پاسخ به این سوال باید اندکی تفکر نمود.

به طور کلی می توان گفت Delegate نوعی شبیه به متد است و همانند آن نیز رفتار می کند. در حقیقت Delegate انتزاعی از یک متد است. در برنامه نویسی ممکن است به شرایطی برخورد کرده باشید که در آنها می خواهید عمل خاصی را انجام دهید اما دقیقاً نمی دانید که باید چه متد یا شیء ای را برای انجام آن عمل خاص مورد استفاده قرار دهید. در برنامه های تحت ویندوز این گونه مسائل مشهودتر هستند. برای مثال تصور کنید در برنامه شما، دکمه ای قرار دارد که پس از فشار دادن این دکمه توسط کاربر شیء ای یا متدی باید فراخوانی شود تا عمل مورد نظر شما بر روی آن انجام گیرد. می توان به جای اتصال این دکمه به شیء یا متد خاص، آن را به یک Delegate مرتبط نمود و سپس آن Delegate را به متد یا شیء خاص در هنگام اجرای برنامه متصل نمود.

در زبان C#. Delegate ها به صورت زیر تعریف می شوند :

(پارامتر ها) نام مورد نظر نوع داده delegate سطح دستیابی

در مورد سطح دستیابی به اندازه کافی صحبت کرده ایم. delegate واژه کلیدی است که به همین صورت به کار می رود. نوع داده، مقدار که توسط متد مورد نظر برگردانده می شود. نام مورد نظر همان عنوان delegate ای است که تعریف کرده اید. پارامتر ها نیز پارامتر هایی را مشخص می کنند که متد های فرستاده شده به این delegate می توانند داشته باشند. نمونه ای از تعریف delegate به صورت زیر است :

public delegate int TestDelegate (Type ob1 , Type ob2);

دستور بالا یک delegate به نام TestDelegate را تعریف می کند که هر متدی را که نوع برگشتی آن int باشد و پارامتر های آن دو شیء باشد می تواند به آن تحویل داده شود.

پس از اینکه یک Delegate تعریف شد می توان متد عضو را با آن Delegate بسته بندی کرد. برای این کار، باید Delegate نمونه سازی شود و متدی به آن ارسال گردد که با نوع

برگشتی و پارامتر های آن سازگار باشد. به عبارت ساده، Delegate شیء ای است که به متد اشاره می کند. بنابراین وقتی Delegate را ایجاد می کنید می توانید شیء را از آن ایجاد نمایید که می تواند مرجع متد را نگهداری کند. علاوه بر این، متد می تواند از طریق این مرجع فراخوانی شود. هر چند که متد شیء نیست، ولی در جایی از حافظه قرار دارد و آدرس نقطه ورود به آن، قابل نسبت دادن به Delegate است. توجه به این نکته مهم است که یک Delegate می تواند در زمان اجرای برنامه، متد های مختلفی را اجرا نماید. برای این کار کافی است آدرس متد دیگری در Delegate قرار گیرد. به این ترتیب Delegate نوعی چند ریختی را فراهم می آورد.

نکته!

نوع Delegate به صورت ضمنی از کلاس System.Delegate مشتق می شود. انواع Delegate شامل متد ها، عملگر ها و خصوصیات می باشند، و شما می توانید آنها را به عنوان پارامتر به متد های دیگر در صورت تمایل منتقل نمایید. همچنین، از آنجایی که کلاس System.Delegate بخشی از Microsoft .NET Framework است، می توانید از آنها در زبان های دیگر استفاده کنید. برای مثال، می توانید یک Delegate را در زبان C# بسازید و از آن برای فراخوانی متدی در داخل زبان VB.Net استفاده نمایید.

نکته!

اگر Delegate فقط شامل یک متد باشد از کلاس Delegate مشتق می شود و Delegate یکنانی نام دارد. اگر Delegate شامل چند متد باشد Delegate چند تایی نام دارد و از کلاس MultiCastDelegate مشتق می شود. هر یک از این دو کلاس، در فضای نام System وجود دارند.

در واقع Delegate کلاسی است که اشیاء ساخته شده از آن می توانند متد های ثبت شده (Register) در خود را به ترتیب اجرا نمایند. برای استفاده از Delegate چهار مرحله اصلی وجود دارد که در زیر، هر مرحله با مثال مشخص خواهند شد :

مرحله اول :

در این مرحله اقدام به تعریف کلاس Delegate می کنیم :

```
public delegate void MyDelegate(int n);
```

معنی عبارت فوق این است که ما می خواهیم یک کلاس Delegate تعریف کنیم که اشیاء آن بتوانند متد هایی را در داخل خود ثبت کنند که پارامتر ورودی آنها یک عدد صحیح (int n) بوده و پارامتر خروجی (void) نداشته باشند. برای روشن شدن مطلب کلاسی به نام Employee و به شکل زیر تعریف می کنیم :

```
public class Employee
{
    public int Age;
    public string FullName;
    public Employee(string fullName, int age)
    {
        Age = age;
        FullName = fullName;
    }
    public void DoIt(int n)
    {
        System.Console.WriteLine("I'm " + FullName + ", I did it " + n + " times.");
    }
}
```

همانگونه که مشاهده می کنید، ما به طور عمد در این کلاس متدی با نام (DoIt) تعریف کرده ایم که پارامتر های ورودی و خروجی آن با آنچه که در تعریف Delegate عنوان گردیده است مطابقت داشته باشد.

مرحله دوم :

در این مرحله نسبت به ایجاد یک شیء از کلاس MyDelegate اقدام می کنیم :

```
MyDelegate DelegateInstance;
```

در این دستور یک شیء به نام DelegateInstance از کلاس (MyDelegate) تعریف نموده ایم. حال برای ادامه مسیر از کلاس Employee یک شیء به نام oEmployee به صورت زیر ایجاد می کنیم :

```
Employee oEmployee = new Employee("Reza bahramirad", 33);
```

مرحله سوم :

در این مرحله تنها کافی است که متد DoIt شیء oEmployee را در شیء DelegateInstance به شکلی که در زیر ذکر گردیده است ثبت نماییم :

```
DelegateInstance = new MyDelegate(oEmployee.DoIt);
```

مرحله چهارم (مرحله آخر) :

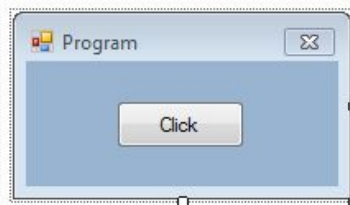
در این مرحله با صدا زدن (Call) شیء DelegateInstance همراه با یک پارامتر عددی، متد ثبت شده در داخل آن به همان پارامتر عددی مشخص شده اجرا می شوند. دقت کنید که در این مثال تنها یک متد ثبت شده در داخل شیء Delegate وجود دارد.

```
DelegateInstance(5);
```

کاملاً واضح است که به راحتی می توانستیم پس از ایجاد شیء oEmployee، با اجرا کردن متد DoIt همراه با همان پارامتر عددی، به همان نتیجه دست پیدا کنیم. ولی دقت کنید که همیشه این چهار مرحله به این شکلی که در اینجا مطرح گردیده است در کنار هم قرار نمی گیرند. نکته جالبی که در این فناوری وجود دارد این است که در یک پروژه واقعی این چهار مرحله، هر کدام در یک قسمت از برنامه تعریف و به کار گرفته می شوند و این مساله امکانات بسیار مفید و جذابی را برای زبان برنامه نویسی C# به ارمغان می آورد.

مثال ۳۹ - ۵: برنامه زیر یک delegate را اعلان و پیاده سازی می کند.

حل : ابتدا یک پروژه از نوع ویندوزی ایجاد کنید، Form برنامه خود را مطابق پنجره تصویر ۵۵ - ۵ طراحی کنید. سپس در قسمت public partial دستورات زیر را بنویسید.



تصویر ۵۵ - ۵

```
public partial class Form1 : Form
{
    public delegate void mydelegate(string name, string family);
    private void personshow(string name, string family)
    {
        MessageBox.Show(name + family);
    }
    private void personshow2(string name, string family)
    {
        MessageBox.Show(family + name);
    }
    private void test(mydelegate d, string s1, string s2)
    {
        d(s1, s2);
    }
}
```

بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس در رویداد Click آن دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    mydelegate d = new mydelegate(personshow2);
    test(d, "reza", "bahramirad");
}
```

ابتدا یک delegate به نام mydelegate تعریف کردیم که دارای دو پارامتر ورودی می باشد. سپس دو متد به نام های personshow و personshow2 برای این delegate تعریف کردیم که دارای دو پارامتر ورودی هستند و رشته ای را دریافت کرده و به هم متصل می کنند و نمایش می دهند. تفاوت آنها در این است که اولی ابتدا Name را نمایش می دهد و دومی نیز در ابتدا Family را قبل از Name نمایش می دهد. در متد test، اولین پارامتر ورودی یک متغیر از نوع delegate به نام d می باشد، دو متغیر رشته ای به نام های s1 و s2 نیز تعریف کردیم که به عنوان پارامتر های ورودی به کار می روند. کاری که این متد انجام می دهد این است که مقادیر s1 و s2 وارد واحد پارامتر ورودی mydelegate که به نام d تعریف شده می شوند.

در رویداد Click کنترل button نیز ابتدا یک متغیر جدید از mydelegate به نام d ایجاد کردیم و متد personshow را به عنوان پارامتر ورودی تعریف کردیم. در متد test دستور d(s1,s2) تعریف شده است، به جای d delegate، متد personshow2 قرار می گیرد. متد

personsShow2 نیاز به دو متغیر ورودی دارد که ما reza و bahramirad را به عنوان پارامترهای ورودی آن در نظر گرفتیم و بعد از آن متد personsShow2 اجرا می گردد. هر جا که دوست داشتید و لازم شد می توانید متد ها را عوض کنید. اگر از personsShow استفاده کنید delegate این تغییرات را برای ما انجام می دهد. کارکرد delegate ها در برنامه نویسی زیاد است.

مثال ۴۰ - ۵: برنامه زیر جمع و تفریق دو عدد را با استفاده از Delegate ها پیاده سازی می کند.

حل : ابتدا یک پروژه از نوع Console ایجاد کرده و یک کلاس داخلی با نام Test ایجاد کنید و سپس دستورات زیر را در آن بنویسید.

```
public delegate void One(int a, int b);
public class Test
{
    public static void Add(int a, int b)
    {
        int c = a + b;
        Console.WriteLine("{0}", c);
    }
    public static void Subtract(int a, int b)
    {
        int c = a - b;
        Console.WriteLine("{0}", c);
    }
}
```

در متد اصلی کلاس program.cs دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    One o1, o2;
    o1 = Test.Add; //gives error here
    o2 = Test.Subtract; //and here!!
    o1(33, 44);
    o2(45, 15);
    Console.ReadLine();
}
```

تعریف Event (رویداد)

ساختار دستوری تعریف یک Event بسیار شبیه به تعریف یک فیلد است، با این تفاوت که

نوع Event باید یک Delegate باشد و قبل از تعریف آن از کلمه کلیدی event استفاده می شود. برای مثال دستورات زیر را در نظر بگیرید.

دستور زیر یک Delegate به نام Tick را تعریف می کند.

```
delegate void Tick (int hourse, int minutes, int seconds);
```

دستور زیر نیز یک Event به نام tick از نوع Tick در کلاس Ticker تعریف می کند.

```
class Ticker
{
public event Tick tick;
...
}
```

یک Event دارای یک مجموعه داخلی از Delegate های متصل شده می باشد. بنابراین لازم نیست که خودتان این مجموعه را نگهداری کنید.

یک event به صورت خودکار، خود را مقدار دهی اولیه می کند و لازم نیست که مثل انواع دیگر، با استفاده از کلمه کلیدی new، یک نمونه از این event ایجاد کرد.

متصل شدن به یک Event

می توان با استفاده از عملگر $+$ یک Delegate را به Event متصل کرد. مثال زیر کلاس clock را نشان می دهد که یک فیلد از نوع Ticker به نام pulsed دارد. کلاس Ticker یک Event به نام tick از نوع Tick دارد که public است. متد Clock.start یک Delegate از نوع Tick را با استفاده از عملگر $+$ به pulsed.tick متصل می کند :

```

delegate void Tick (int hourse, int minutes, int seconds);
class Ticker
{
public event Tick tick;
...
}
class Clock
{
...
public void start()
{
pulsed.tick += new Tick (this.RefreshTime);
}
...
private void RefreshTime (int hourse, int minutes, int seconds)
{
...
}
private Ticker pulsed = new Ticker();
}

```

جدا شدن از یک Event

با دانستن اینکه عملگر $+=$ برای متصل کردن یک Delegate به یک Event به کار می رود، ممکن است حدس زده باشید که عملگر $-$ برای جدا کردن یک Delegate از یک Event به کار می رود. فراخوانی عملگر $-$ ، Delegate را از مجموعه داخلی Delegate های متصل شده Event حذف می کند. برای مثال، کد زیر متد Clock.stop را نشان می دهد که RefreshTime را از pulsed tick جدا می کند :

```

class Clock
{
...
public void stop()
{
pulsed.tick -= new Tick (this.RefreshTime);
}
private void RefreshTime (int hourse, int minutes, int seconds)
{
...
}
private Ticker pulsed = new Ticker();
}

```

نکته!

عملگر های += و -= بر پایه عملگر های + و - استوار هستند. برای کنار هم قرار دادن Delegate ها در کنار هم، می توانید از عملگر + و برای جدا کردن از هم از عملگر - استفاده کنید. برای مثال، کنار هم قرار دادن دو Delegate در کنار هم یک Delegate جدید ایجاد می کند که وقتی فراخوانی می شود، هر دو Delegate را فراخوانی می کند.

فراخوانی یک Event

مثل Delegate ها، Event ها را نیز می توان با استفاده از پرانتز فراخوانی کرد. وقتی که یک Event را فراخوانی می کنید تمام Delegate های متصل شده به آن به ترتیب فراخوانی می شوند. برای مثال کلاس Ticker در اینجا یک متد private به نام Notify دارد که یک Event به نام tick را فراخوانی می کند :

```
delegate void Tick (int hourse, int minutes, int seconds)
class Ticker
{
public event Tick tick;
...
private void Notify (int hourse, int minutes, int seconds)
{
if (tick != null)
{
tick (hourse,minutes,seconds);
}
}
...
}
```

بررسی null بودن در مثال قبل لازم است، زیرا یک فیلد Event به صورت ضمنی null است و فقط وقتی null نیست که یک Delegate با استفاده از عملگر = - به آن متصل شده باشد. اگر یک Event که null باشد را فراخوانی کنید یک NullReferenceException دریافت خواهید کرد.

در صورتی که یک Event از نوع public باشد (مثل tick) فقط می تواند از داخل یک متد هم سطح فراخوانی شود. برای مثال tick یک Event در داخل کلاس Ticker است. بنابراین، فقط متد های کلاس Ticker می توانند tick را فراخوانی کنند :

```

class Example
{
static void Main()
{
Ticker pulsed = new Ticker();
pulsed.tick (12, 29, 59); //Compile - Time Error
}
}

```

به دستورات زیر توجه کنید :

```

public delegate void EventHandler (object sender, System.EventArgs e);
class Publisher : System.Collections.ArrayList
{
    public event EventHandler Added;
    protected virtual void OnAdded (System.EventArgs e)
    {
        if (Added != null)
            Added(this, e);
    }
    public override int Add(object value)
    {
        int i = base.Add(value);
        OnAdded(System.EventArgs.Empty);
        return i;
    }
}
class Subscriber
{
    public void AddedEventHandler(object sender, System.EventArgs e)
    {
        System.Console.WriteLine("Add Event Occured");
    }
}

```

رویداد ها به یک شیء امکان می دهند تا اشیای دیگری را در هنگام وقوع رخدادی باخبر سازند. شیء ای که رویداد را فراخوانی می کند Publisher نامیده می شود و شیء ای که آن را مدیریت می کند Subscriber نامیده می شود.

در دستورات بالا برای نمایش کاربرد رویداد ها ابتدا کلاس Publisher را تعریف می کنیم که از کلاس ArrayList ارث می برد و زمانی که آیتمی به لیست اضافه شود رویدادی را اجرا می کند. پیش از ساخت رویداد نیاز به یک Delegate داریم که کل Subscriber را نگهداری کند این Delegate می تواند از هر نوعی باشد. روش اصولی تعریف به این نحو است که یک Delegate از نوع Void تعریف می کنیم که دو پارامتر را می پذیرد. اولین پارامتر شیء منبع رویداد شیء را مشخص می کند که می تواند از نوع System.EventArgs یا از کلاس آن

ارث بری داشته باشد. به طور معمول این پارامتر جزئیات مربوط به رویداد را در خود نگهداری می کند ولی لازم نیست داده ای را به آن ارسال کنیم و تنها از کلاس پایه EventArgs به عنوان نوع پارامتر استفاده می کنیم. پس از تعریف Delegate با آوردن کلمه کلیدی event و پس از آن نام Delegate و در انتها نام رویداد، رویدادی را ایجاد کنیم. کلمه کلیدی event برای ایجاد نوع خاصی از Delegate استفاده می شود که تنها از داخل کلاسی که در آن تعریف شده است می تواند فراخوانده شود و سطح دسترسی آن public است. بنابراین سایر کلاس ها اجازه دارند در این رویداد مشترک شوند. Delegate که پس از کلمه کلیدی event آورده می شود Event Delegate نامیده می شود. به جای آن می توانیم از Delegate از پیش تعریف شده خود Visual Studio یعنی Event Handler استفاده کنیم که دقیقا مشابه آنچه که در اینجا تعریف کردیم می باشد. این Delegate از پیش تعریف شده توسط کتابخانه کلاس NET. برای ساختن رویداد هایی استفاده می شود که هیچ داده رویدادی ندارند.

برای اجرای رویداد نیاز به ساخت یک متد فراخوانی رویداد داریم که در اینجا به نام OnAdded و با سطح دستیابی protected و به صورت virtual تعریف کرده ایم. این کار باعث می شود تا به کلاس های مشتق شده اجازه override داده شود. این متد تنها در شرایطی اجرا می شود که null نباشد، برای اجرای آن از this به عنوان ارسال کننده، و نیز از شیء e که به متد ارسال شده بود.

حالا که ما رویداد و متدی برای فراخوانی داریم، تنها کافی است که متد add از ArrayList را override کنیم تا بتواند رویداد ما را اجرا کند. در نسخه بازنویسی شده متد، ابتدا متد پایه add را فراخوانی می کنیم و نتیجه را ذخیره می کنیم و با استفاده از متد OnAdded و با فیلد Empty از کلاس EventArgs که نماینده رویدادی بدون داده است اجرا می کنیم. و در نهایت با دستور return نتیجه را به فراخواننده باز می گردانیم.

برای تست کردن کلاس Publisher نیاز به کلاسی داریم که به عنوان رویداد مشترک استفاده می شود. عنوان کلاس جدید خود را Subscriber قرار می دهیم. این کلاس شامل متدی است که اجرا کننده رویداد است و تعریف آن مشابه EventDelegate است.

حال در متد اصلی کلاس program نمونه هایی از کلاس های Publisher و Subscriber ایجاد می کنیم. برای آنکه اجرا کننده را که در شیء کلاس Subscriber قرار دارد در رویداد متعلق به کلاس Publisher مشترک کنیم نیاز داریم تا اجرا کننده رویداد را به رویداد اضافه کند، به صورتی که انگار Delegate است. برخلاف Delegate ما رویداد را مستقیماً از خارج از کلاس که دربرگیرنده آن است فراخوانی نمی کنیم. رویداد تنها می تواند از طریق Publisher اجرا شود وقتی این مورد اتفاق می افتد که آیتمی به شیء اضافه شود.

```
static void Main(string[] args)
{
    Subscriber s = new Subscriber();
    Publisher p = new Publisher();
    p.Added += s.AddedEventHandler;
    p.Add(10);
}
```

اکنون برنامه را اجرا کرده و نتیجه را مشاهده کنید.

توجه! می توان از کلاس Button ها یک متغیر واسطه گرفت و یک کنترل Button جدید ایجاد کرد.

```
private void button1_Click(object sender, EventArgs e)
{
    Button btn = new Button();
    this.Controls.Add(btn);
}
```

اکنون اگر برنامه را اجرا کنید و بر روی دکمه کلیک کنید، یک کنترل button جدید ایجاد می شود. می توان برای این کنترل خصوصياتی را تنظیم کرد:

```
private void button1_Click(object sender, EventArgs e)
{
    Button btn = new Button();
    btn.Location = new Point(10, 20);
    btn.Text = "Click Me";
    this.Controls.Add(btn);
}
```

اما یک مساله مهم باقیست، وقتی در زمان اجرا روی دکمه کلیک کردید بایستی رویدادی را برای ما دنبال کند. اینجاست که با مفهوم Event ها و Delegate ها می توانیم رویداد Click برای این کنترل بنویسیم و داخل آن عملی را انجام دهیم.

```
private void button1_Click(object sender, EventArgs e)
{
    Button btn = new Button();
    btn.Location = new Point(10, 20);
    btn.Text = "Click Me";
    btn.Click += new EventHandler(btn_Click);
    this.Controls.Add(btn);
}

void btn_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello Ardebil");
}
```

روندی که در اینجا طی شد به این صورت است که با استفاده از یک Delegate متدی به نام btn_Click فراخوانی شد که دارای دو پارامتر ورودی است.

EventHandler یک Delegate است. کافی است که روی آن راست کلیک کرده و گزینه Go To Definition را کلیک کنید.

```
namespace System
{
    ...public delegate void EventHandler(object sender, EventArgs e);
}
```

در فراخوانی رویداد ها به دلیل اینکه Delegate می تواند مرجع چندین رویداد باشد از آنها استفاده می کنیم. به این ترتیب می توانیم چندین رویداد Click را با یک Delegate به نتیجه رسانده و استفاده کنیم.

مثال ۴۱ - ۵: می خواهیم در برنامه جدید خودمان یک رویداد نوشته و از آن استفاده کنیم.

این برنامه دارای متد های زیر است :

متد RealPrint : این متد یک رشته را نمایش می دهد.

متد PrintFile : نمایش یک رشته و اضافه کردن کاراکتر ۱ به آن رشته است.

متد Display : دارای یک پارامتر ورودی از نوع Delegate است. این متد دارای یک پارامتر ورودی از Delegate و نیز یک پارامتر به صورت رشته ای دریافت می کند.

Delegate : یک Delegate به نام Print که از نوع void است و دارای یک پارامتر ورودی از نوع رشته ای است.

```

public partial class Form1 : Form
{
    public delegate void Print(string s);
    private void RealPrint(string s)
    {
        MessageBox.Show(s);
    }
    private void PrintFile(string s)
    {
        MessageBox.Show(s+"1");
    }
    public void Display(Print p, string s)
    {
        p(s);
    }
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        Print s = null;
        s += new Print(RealPrint);
        Display(s, "Reza");
    }
}

```

در بخش دستورات Form1_Load یک متغیر واسط از Delegate ایجاد کرده و مقدار آن را برابر null قرار داده ایم. مانند رویداد Click برای btn در مثال قبل که با + انجام دادیم در اینجا هم از این روش استفاده کردیم. برای فراخوانی رویداد از متد Display استفاده می کنیم و s Delegate و رشته Reza را به عنوان پارامتر ورودی برای آن در نظر می گیریم. کاری که انجام می شود به این صورت است که متد RealPrint را به عنوان یک رویداد برای Delegate Print تصور می کند، و پارامتر ورودی رشته Reza را به آن انتقال می دهد. یعنی رشته Reza را در پارامتر ورودی متد RealPrint قرار می دهد.

حال کافی است که برنامه را اجرا کنید ابتدا پیغام Reza نمایش داده می شود و سپس Form بارگذاری می شود. حال اگر دستور s += new Print (PrintFile); را بنویسید Reza را نمایش خواهد داد.

مثال ۴۲ - ۵: برنامه زیر نحوه کاربرد Delegate ها و Event ها را در قالب نمایش چند پیغام نمایش می دهد.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کرده و دستورات زیر بنویسید.

```
public delegate void EventHandler();
class Program
{
    public static event EventHandler show;
    static void Main(string[] args)
    {
        // Add event handlers to Show event.
        show += new EventHandler(C);
        show += new EventHandler(Java);
        show += new EventHandler(Delphi);
        show += new EventHandler(Delphi);
        // Invoke the event.
        show.Invoke();
    }
    static void C()
    {
        Console.WriteLine("C#");
    }

    static void Java()
    {
        Console.WriteLine("Java");
    }

    static void Delphi()
    {
        Console.WriteLine("Delphi");
    }
}
```

حال کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

آشنایی با ساختار ها (Struct)

ساختار چیست؟ همانطور که با استفاده از کلاس ها می توان انواع جدید و مورد نظر را ایجاد نمود، با استفاده از Struct ها نیز می توان انواع مقداری جدید و مورد نظر را ایجاد کرد. از آنجایی که Struct ها به عنوان انواع مقداری در نظر گرفته می شوند، از این رو تمامی اعمال مورد استفاده بر روی انواع مقداری را می توان برای Struct ها در نظر گرفت. Struct ها بسیار شبیه به کلاس ها هستند ولی تفاوت هایی با کلاس ها نیز دارند به همین دلیل، ساختار ها را کلاس های سبک وزن می نامند. Struct ها می توانند دارای متد، فیلد، خاصیت و سازنده باشند. البته سازنده ای که دارای پارامتر باشد نه سازنده بدون پارامتر یا اصطلاحاً سازنده پیش فرض. عموماً ساختار ها مجموعه کوچکی از عناصری هستند که به صورت

منطقی با یکدیگر دارای رابطه هستند. نمونه بارز Struct ها را می توان به نوع DateTime در فضای نام System در NET اشاره کرد.

ساختار ها در زبان C#, ارث بری و مخرب ها را پشتیبانی نمی کنند. ساختار ها نه به صورت پایه و نه به صورت مشتق شده می توانند مورد استفاده قرار گیرند، تنها می توان از آنها شیء ایجاد کرد.

توجه کنید که در ساختار ها نمی توان از سطح دسترسی Protected استفاده کرد چون از این سطح دسترسی تنها زمانی می توانیم استفاده کنیم که ارث بری داشته باشیم. ساختار ها نیز ارث بری را پشتیبانی نمی کنند.

با استفاده از ساختار ها می توان اشیایی با انواع جدید ایجاد کرد که این اشیاء می توانند شبیه به انواع موجود مانند float, int و ... باشند. حال سؤال این است که چه زمانی از ساختار ها می توان استفاده کرد؟

در ابتدا به نحوه استفاده از انواع موجود در زبان C# توجه کنید. این انواع دارای مقادیر و عملگر های معینی جهت کار با این مقادیر هستند. حال اگر نیاز به شیء ای دارید که همانند این انواع رفتار نماید لازم است تا از ساختار ها استفاده کنید.

تعریف ساختار ها بسیار شبیه به کلاس هاست. نحوه تعریف ساختار ها در زبان C# به صورت زیر است :

نام ساختار struct سطح دسترسی

```
{  
;دستورات  
...  
}
```

همانطور که در تعریف بالا مشاهده می کنید تعریف ساختار ها بسیار شبیه به کلاس هاست. کافی است که بعد از سطح دسترسی از کلمه کلیدی struct و سپس نام ساختار را ذکر کنیم و در داخل {} نیز دستورات مورد نظر خود را بنویسیم.

در زیر انواع تفاوت های یک ساختار و کلاس را مورد بررسی قرار می دهیم :

- ۱- ساختار ها، برخلاف کلاس ها ارث بری را پشتیبانی نمی کنند. آنها به طور تلویحی مانند تمام انواع داده ها در زبان C# از کلاس Object ارث می برند.
- ۲- در ساختار ها هیچ گونه ارث بری وجود ندارد.
- ۳- ساختار ها همانند کلاس ها می توانند چندین Interface را پشتیبانی کنند.
- ۴- ساختار ها نمی توانند مخرب داشته باشند و نیز نمی توان برای سازنده ها یک سازنده پیش فرض بدون پارامتر نوشت.
- ۵- اگر هیچ سازنده ای برای ساختار ها تعریف نکنید یک سازنده خودکار پیش فرض، به طور تلویحی برای ساختار ها ایجاد می شود که تمام فیلد های آن را مقدار دهی اولیه متناسب با نوع این فیلد ها می کند.
- ۶- اگر سازنده ای برای یک ساختار ایجاد کنید بایستی تمام فیلد های آن ساختار را در آن مقدار دهی اولیه کنید.
- ۷- مقدار دهی اولیه فیلد ها در ساختار ها قابل استفاده نیست. به این معنی که نمی توانید یک فیلد در ساختار را در زمان تعریف، مقدار دهی اولیه کنید.
- ۸- ساختار ها، از نوع داده مقدار (Value Type) هستند. نوع داده ارجاع (References Type) همیشه به طور خودکار از طریق ارجاع آدرس، به متد ارسال می شوند و ارسال از طریق Ref برای آنها الزامی نیست. یعنی به طور خودکار تغییرات درون متد را نگه می دارند. از آنجایی که ساختار ها، از نوع داده مقدار (Value Type) هستند، در صورتی که بدون ذکر واژه Ref به متد ارسال شوند، تغییرات اعمال شده روی آنها در متد، بعد از خروج از متد، موجود نخواهد بود.
- آخرین نکته ای که در مورد ساختار ها برای چندمین بار اشاره می کنم آن است که، ساختار ها انواع مقداری هستند و مستقیماً مقدار را در خود نگه می دارند از این رو در Stack نگهداری می شوند. استفاده از ساختار ها همانند سایر انواع مقداری است.

```
struct Time
{
    public int hours, minutes, seconds;
}
```

البته public کردن فیلد های یک struct خلاف قاعده کپسوله سازی است و معمولاً توصیه نمی شود. هیچ راهی وجود ندارد که تضمین کنید مقدار فیلد های public معتبر باشند، زیرا هر کسی می تواند مقدار minutes یا second را بزرگتر از ۶۰ قرار دهد. یک روش بهتر این است که فیلد ها را private کنیم و برای struct خود متد و سازنده تعریف کنیم. برای مثال به دستورات زیر توجه کنید :

```
struct Time
{
public Time (int hh, int mm, int ss)
{
hours = hh % 24;
minutes = mm % 60;
seconds = ss % 60;
}
public int Hours ()
{
return hours;
}
...
private int hours, minutes, seconds;
}
```

تعریف متغیر از نوع Struct

بعد از اینکه یک Struct را تعریف کردید می توانید از اسم این Struct به عنوان یک نوع استفاده کنید به همان صورت که از اسم یک نوع دیگر استفاده می کنید. اگر Time، اسم Struct باشد می تواند متغیر، فیلد یا پارامتر هایی از نوع Time را تعریف کنید. برای مثال به دستورات زیر توجه کنید :

```
struct Time
{
...
private int hours, minutes, seconds;
}
class Example
{
public void Method (Time parameter)
{
Time localVar;
...
}
private Time field;
}
```

نوع struct یک نوع مقداری است، یعنی متغیری از نوع struct، مقدار خود را مستقیماً روی پشته نگه می دارد. برای مثال Time سه فیلد به نام های hour, minutes, seconds دارد. پس وقتی که یک متغیر محلی از نوع Time به نام now تعریف می کنید در واقع سه متغیر محلی به نام های now.hours, now.minutes, now.seconds روی پشته تعریف می کنید.

فراخوانی سازنده های یک Struct

ساده ترین راه برای مقدار دهی اولیه به تمام فیلدها در یک متغیر از نوع Struct این است که سازنده پیش فرض آن را فراخوانی کنید. سازنده پیش فرض برای یک Struct همیشه وجود دارد و همیشه فیلدهای struct را با صفر یا False و یا null مقدار دهی اولیه می کند. در مثال زیر تمام فیلدهای int در now را با صفر مقدار دهی اولیه می کند و کامپایل می شود.

```
Time now = new Time();  
Console.WriteLine(now.hours); //compiles okay, write out 0  
Console.WriteLine(now.minutes); //compiles okay, write out 0  
Console.WriteLine(now.seconds); //compiles okay, write out 0
```

وقتی که یک سازنده را فراخوانی می کنید همیشه باید قبل از آن از کلمه کلیدی new استفاده کنید. این مساله، این واقعیت را که مقادیر یک Struct روی پشته قرار دارد تغییر نمی دهد. کلمه کلیدی new برای شرایط متفاوت، رفتار متفاوتی را از خود نشان می دهد. این کلمه کلیدی برای یک کلاس، حافظه ای را از heap اختصاص می دهد درحالی که برای یک Struct، حافظه ای را از پشته اختصاص می دهد.

اگر خودتان یک سازنده برای Struct نوشته باشید می توانید از این سازنده نیز برای مقدار دهی اولیه به یک متغیر از نوع struct استفاده کنید. همانطور که قبلاً در این فصل نیز گفته شد، سازنده یک Struct همیشه باید تمام فیلدهای Struct را صریحاً مقدار دهی اولیه کند به مثال زیر توجه کنید.

```

struct Time
{
public Time (int hh, int mm)
{
hours = hh;
minutes = mm;
seconds = 0;
}
...
private int hours, minutes, seconds;
}

```

مثال زیر، now را با فراخوانی این سازنده مقدار دهی اولیه می کند.

```

Time now = new Time (12,30);
Console.WriteLine (now.hours); //Compiles okay, writes out 12
Console.WriteLine (now.minutes); //Compiles okay, writes out 30
Console.WriteLine (now.seconds); //Compiles okay, writes out 0

```

کپی کردن متغیرها از نوع Struct

می توان یک متغیر از نوع Struct را مساوی یک متغیر دیگر از نوع Struct قرار داد، ولی فقط اگر متغیر سمت راست صریحاً مقدار دهی شده باشد (یعنی اگر تمام فیلد های آن صریحاً مقدار دهی شده باشد). مثلاً کد زیر کامپایل می شود زیرا now صریحاً مقدار دهی شده است.

```

Time now = new Time (12,30);
Time copy = now;

```

کد زیر کامپایل نمی شود زیرا now صریحاً مقدار دهی نشده است.

```

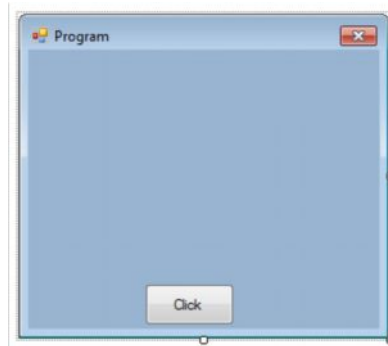
Time now;
Time copy = now; //compile time error

```

وقتی که یک متغیر از نوع struct را کپی می کنید هر فیلد در سمت چپ، مستقیماً از فیلد مربوط در سمت راست کپی می شود. این عمل کپی به صورت یک کپی گروهی سریع انجام می شود و هیچ وقت Exception ایجاد نمی کند.

مثال ۴۳ - ۵: می خواهیم با استفاده از ساختار ها برنامه ای بنویسیم که مستطیلی را بر روی Form رسم کند؟

حل : ابتدا یک پروژه جدید از نوع ویندوزی ساخته و Form آن را مطابق پنجره تصویر ۵۶ - ۵ طراحی کنید.



تصویر ۵۶ - ۵

دیگر دلیلی ندارد که کلاس جدیدی ایجاد کنید و درون آن ساختار خودمان را تعریف کنید ساختار و دستورات مربوط به آن را در بخش `public partial` به صورت زیر می توان تعریف کرد :

```
struct MyPoint
{
    int x;
    int y;

    public int X
    {
        get { return x; }
        set { x = value; }
    }
    public int Y
    {
        get { return y; }
        set { y = value; }
    }
    public MyPoint(int x_t, int y_t)
    {
        x = x_t;
        y = y_t;
    }
}
```

حال باید برنامه رسم مستطیل را بنویسیم، برای این کار از بخش گرافیکی زبان C# استفاده می کنیم. برای استفاده از متد های گرافیکی زبان C# باید از کلاس `Graphic` استفاده کرد.

```
private void button1_Click(object sender, EventArgs e)
{
    MyPoint pt1 = new MyPoint(40, 40);
    MyPoint pt2 = new MyPoint(80, 80);
    Graphics gr;
    gr = this.CreateGraphics();
    gr.DrawRectangle(Pens.Brown, pt1.X, pt1.Y, pt2.X, pt2.Y);
}
```

مثال ۴۴ - ۵: برنامه زیر نشان می دهد که نوع داده DateTime یک ساختار است. ابتدا یک نوع DateTime ایجاد می کنیم، سپس آن را در داخل یک متغیر از نوع DateTime کپی می کنیم. هنگامی که تغییرات در زمان ایجاد شد، مشاهده خواهید کرد که همان مقدار در حافظه باقیمانده است.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کنید، سپس دستورات زیر را در آن بنویسید.

```
static void Main(string[] args)
{
    // DateTime is a struct.
    DateTime date = new DateTime(2000, 1, 1);
    // When you assign a DateTime, a separate copy is created.
    DateTime dateCopy = date;
    // The two structs have the same values.
    Console.WriteLine(date);
    Console.WriteLine(dateCopy);
    // The copy is not affected when the original changes.
    date = DateTime.MinValue;
    Console.WriteLine(dateCopy);
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

```
1/1/2000 12:00:00 AM
1/1/2000 12:00:00 AM
1/1/2000 12:00:00 AM
```

مثال ۴۵ - ۵: می خواهیم برنامه ای بنویسیم که با استفاده از ساختار ها، مشخصات یک دانشجو را نمایش دهیم؟

حل : ابتدا یک پروژه جدید ویندوزی ایجاد کرده و Form آن را مطابق پنجره تصویر ۵۷ - ۵ طراحی کنید، سپس یک کلاس داخلی طبق روش های گفته شده ایجاد کنید، کد های بخش کلاس را حذف کرده ساختار Student را به صورت زیر در آن تعریف کنید.

```
public struct Student
{
    public string FirstName;
    public string LastName;
    public string Email;
}
```

به Form برنامه برگشته و بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس در رویداد Click آن دستورات زیر را بنویسید.

تصویر ۵۷ - ۵

```
private void button1_Click(object sender, EventArgs e)
{
    Student student;
    student.FirstName = "Morteza";
    student.LastName = "Soleymanpour";
    student.Email = "ma_soleymanpour@yahoo.com";
    DisplayStudent(student);
}
```

بعد از تعریف متغیر از نوع Student می توانید به هر یک از متغیرهای موجود در این ساختار با استفاده از student دسترسی پیدا کنید. برای این کار هنگامی که student را وارد کردید یک نقطه نیز قرار دهید. به این ترتیب تمام اعضای این ساختار به وسیله Visual Studio نمایش داده خواهند شد و می توانید مقدار هر یک از آنها را تغییر دهید.

اکنون متد DisplayStudent را به صورت زیر و در کلاس Form تعریف کنید.

```
private void DisplayStudent(Student student)
{
    textBox1.Text = student.FirstName;
    textBox2.Text = student.LastName;
    textBox3.Text = student.Email;
}
```

وظیفه متد DisplayStudent این است که یک ساختار از نوع Student را به عنوان پارامتر ورودی دریافت کند و سپس با استفاده از مقادیر وارد شده در اعضای این ساختار، خاصیت Text هر کدام از TextBox های روی Form را تنظیم کند. اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

آشنایی با نوع داده های شمارشی

از این نوع داده ای، برای تعریف مقادیری به نام ثوابت شمارشی استفاده می شود. گاهی کار کردن با ثوابت نام دار آسان تر از کار کردن با مقادیر ثابت عددی است. با استفاده از نوع داده های شمارشی می توان مجموعه ای از ثابت های نامدار مرتبط را ایجاد کرد. چنین مجموعه ای مثلاً می تواند حاوی نام رنگ ها یا حاوی نام روز های هفته باشد. برای تعریف یک نوع شمارش، از کلمه کلیدی enum استفاده می شود و اعضای آن را در داخل یک جفت آکولاد نوشته و آنها را با کاما از هم جدا می کنیم. همچنین می توان به هر یک از اعضای نوع شمارشی مقادیری را تخصیص داد.

enum نوع خاصی از انواع مقداری است که شامل لیستی از ثوابت می باشد. در زیر نحوه تعریف چند نوع enum را در زبان C# می توانید مشاهده کنید :

```
enum Week
{
    Sunday = 0,
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
    Saturday = 6
};
```

اگر به یک عنصر نوع شمارشی مقدار ندهید، کامپایلر به آن مقداری می دهد که یکی بیشتر از مقدار عنصر قبلی است، به استثنای عنصر اول که کامپایلر به آن مقدار صفر را می دهد.

```
enum State
{
    Run = 0,
    Wait = 3,
    Stop = 5,
    Offline = Stop + 5
};
```

می توان از این نوع شمارشی به عنوان نگهدارنده ای برای این ثوابت استفاده نمود. برای دستیابی به این عناصر مانند دستیابی به عناصر Static یک کلاس عمل می شود.

```
private void Form1_Load(object sender, EventArgs e)
{
    State S = State.
}
```



دستور Switch در مثال زیر کاربرد خوبی از نوع داده شمارشی را برای ما ارائه می دهد.

```
State S;
switch (S)
{
    case State.Run: break;
    case State.Wait: break;
    case State.Stop: break;
    case State.Offline: break;
}
```

نوع مربوط به عناصر نیز بطور ضمنی به صورت int تعریف می شود، اما می توان آن را با آوردن دو نقطه پس از نام نوع داده شمارشی و سپس تعیین نوع مورد نظر تغییر داد.

```
enum State : byte
{
    Run = 0,
    Wait = 3,
    Stop = 5,
    Offline = Stop + 5
};
```

سطح دسترسی نوع داده شمارشی همانند کلاس ها می باشد و به صورت پیش فرض Internal هستند اما می توان آنها را به صورت public نیز تعریف کرد.

اگر چه نوع داده شمارشی معمولاً در قسمت namespace تعریف می شود اما آنها را در داخل کلاس نیز می توان قرار داد. به این صورت سطح دسترسی آن به صورت Private تعریف می شود که می توان آن را به هر یک از سطوح دسترسی تغییر داد. می توان یک ثابت نوع داده شمارشی را به نوع int تبدیل کرد و یا با استفاده از متد ToString() تبدیل به یک رشته کرد.

```
State S;
int i = (int)S;
string t = S.ToString();
```

مثال ۴۶ - ۵: برنامه زیر نحوه استفاده از نوع های داده ای enum را نشان می دهد.

حل : ابتدا یک پروژه جدید از نوع Console ایجاد کنید سپس یک نوع enum در داخل کلاس Program به صورت زیر تعریف کنید.

```
enum Importance
{
    None,
    Trivial,
    Regular,
    Important,
    Critical
};
```

حال در متد اصلی کلاس program.cs دستورات زیر را بنویسید.

```
static void Main(string[] args)
{
    // 1.
    Importance value = Importance.Critical;
    // 2.
    if (value == Importance.Trivial)
    {
        Console.WriteLine("Not true");
    }
    else if (value == Importance.Critical)
    {
        Console.WriteLine("True");
    }
}
```

اکنون کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

نکته!

تمام انواع شمارشی، به صورت خودکار چند متد سودمند شمارشی را از کلاس System.Enum به ارث می برند.

در این فصل به طور کامل در مورد مباحث شیء گرایی صحبت کردیم. مبحث خاصی نمانده است اما پیشنهاد ما به عنوان یک هم رشته ای به تمام دوستان عزیز این است که تا جایی که می توانید مباحث شیء گرایی را بررسی کنید. برنامه هایی را با استفاده از مباحث شیء گرایی و بدون استفاده از مباحث شیء گرایی طراحی کنید آن وقت خواهید دید که چقدر تفاوت در

سرعت اجرا، در سرعت تولید، میزان پویایی برنامه هایی که با استفاده از مباحث شیء گرایی نوشته می شوند وجود دارد. مشکل اصلی افرادی که می خواهند به سمت برنامه نویسی شیء گرا حرکت کنند این است که نمی دانند باید در چه مواقعی از آن استفاده کنند به همین خاطر ما پیشنهاد کردیم که ابتدا یکسری برنامه هایی بدون استفاده از مباحث شیء گرایی پیاده سازی کنید بعد از آنکه برنامه تمام شد یکسری در هایی به صورت خود به خود برای شما گشوده می شود. به طور مثال وقتی شما متد یا رویدادی را بار ها و بار ها اجرا کردید کم کم متوجه می شوید که متد ها و یا رویداد ها را به چه نحوی بنویسید و استفاده کنید.

در فصل ششم سعی می کنیم مباحثی را که در سه فصل گذشته خواندیم در قالب چند مثال عملی توضیح دهیم تا بیشتر با مباحث برنامه نویسی شیء گرا در دنیای برنامه نویسی آشنا شوید.



فصل ششم

طراحی و تولید اشیاء نرم افزاری

با بیشتر تکنیک های برنامه نویسی شیء گرا در فصل گذشته آشنا شدید سعی می کنیم تا با استفاده از این تکنیک ها برنامه ای کاربردی ایجاد کرده تا با نحوه استفاده از آنها در عمل بیشتر آشنا شوید.

مثال کاربردی ۱

در این بخش برنامه ای خواهیم نوشت که به وسیله آن بتوانید سایت های موجود در بخش Favorites مرورگر IE را مشاهده کنید. همچنین در این برنامه دکمه ای قرار می دهیم که بوسیله آن بتوانید آن سایت ها را با استفاده از مرورگر IE مشاهده کنید.

به احتمال زیاد تا به حال سایت های مورد علاقه خود را در بخش Favorites مرورگر IE ذخیره کرده اید. سوالی که شاید به ذهن شما رسیده این است که مرورگر IE چگونه اطلاعات موجود در قسمت Favorites را ذخیره می کند؟ در حقیقت گزینه های موجود در قسمت Favorites فقط مخصوص IE نیستند و هر برنامه ای می تواند به آنها دسترسی داشته باشد، فقط کافی است که محل ذخیره شده این فایل ها را بداند.

می خواهیم برنامه ای بنویسیم که پوشه Favorites را باز کرده و از لینک های درون آن استفاده کند، برای مثال آنها را به یک لیست اضافه کند، سایت مربوط به آنها را با استفاده از مرورگر IE نمایش دهد.

در فصول گذشته برنامه هایی ایجاد کردید که بیشتر عملیات های خود را روی Form انجام می دادند. در این قسمت می خواهیم کلاسی ایجاد کنیم که لیستی از گزینه های موجود در Favorites را نمایش دهد. به این ترتیب می توانید از لیستی که این کلاس ایجاد می کند در هر برنامه و به هر نحوی که بخواهید استفاده کنید. برای مثال می توانید گزینه های موجود در این لیست را با استفاده از یک کنترل LixtBox نمایش دهید و سپس به کاربر اجازه دهید که با استفاده از مرورگر IE آنها را مشاهده کند.

بهترین راه برای تولید چنین برنامه ای این است که کلاس هایی را به صورت زیر ایجاد کنیم : WebFavorite : هر شیء ای از این کلاس برای نگهداری یکی از گزینه های موجود در Favorites و نیز ویژگی های آن مانند Name و URL به کار می رود.

Favorites : این کلاس می تواند کامپیوتر کاربر را برای پیدا کردن Favorites جستجو کند، برای هر یک از گزینه های Favorites یکی از گزینه های Favorites یک شیء از کلاس WebFavorite ایجاد کرده و اشیای ایجاد شده را در یک آرایه قرار دهد.

این دو کلاس اصطلاحاً قسمت back_end برنامه را تشکیل می دهند. به عبارت دیگر می توانیم بگوییم تمام کلاس هایی که وظیفه خاصی را در برنامه انجام می دهند اما هیچ رابط گرافیکی خاصی ندارند تا در برنامه به کاربر نمایش داده شوند، قسمت back_end یک برنامه را تشکیل می دهند. جدا کردن این قسمت ها از ظاهر برنامه باعث می شود تا بتوانید از این کلاس ها در برنامه های دیگر خود نیز براحتمی استفاده کنید (استفاده مجدد از کد). همچنین برای تکمیل این برنامه به یک قسمت front_end نیاز دارید که رابط کاربری برنامه را تشکیل می دهد. در این برنامه، این قسمت شامل یک Form ویندوزی و چند کنترل عادی خواهد بود. ابتدا یک پروژه از نوع Windows ایجاد کرده و Form آن را مطابق پنجره تصویر ۱ - ۶ طراحی کنید.

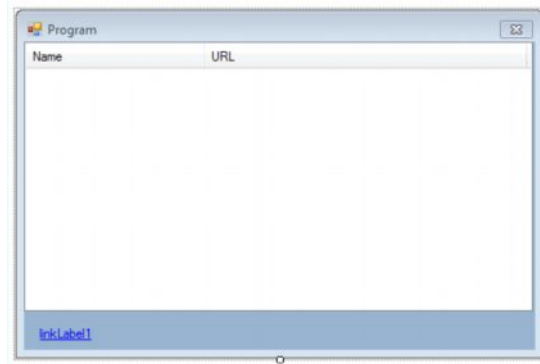
یک کنترل ListView بر روی Form خود اضافه کنید و برخی از خاصیت های آن را به صورت زیر تغییر دهید :

خاصیت Anchor آن را برابر با Left , Right , Top , Bottom قرار دهید.

خاصیت View این کنترل را برابر با Details قرار دهید.

بر روی خاصیت Columns این کنترل کلیک کنید، کادر ColumnHeader Collection Editor نمایش داده خواهد شد. در این کادر با کلیک روی دکمه Add دو ستون به نام های Name و Url ایجاد کنید.

یک کنترل LinkLabel اضافه کنید و خاصیت Anchor آن را برابر با Bottom, Left, Right قرار دهید و خاصیت TextAlign را برابر MiddleLeft تنظیم کنید.



تصویر ۱ - ۶

تمام کاری که تا به حال انجام داده اید تنها طراحی ظاهر برنامه است تا بتواند نتیجه بدست آمده را نمایش دهد. در این Form کنترل ListView برای نمایش نام و آدرس هر یک از گزینه موجود در Favorites به کار می رود. کنترل LinkLabel هم برای باز کردن برنامه مرورگر IE و نمایش سایت انتخاب شده در لیست استفاده می شود. حال می خواهیم بررسی کنیم که چگونه می توان کلاس های بخش back_end برنامه را ایجاد کرد.

یک کلاس داخلی به نام WebFavorite ایجاد کرده و دستورات زیر را در آن بنویسید. در این کلاس دستور `using System.IO;` را اضافه کنید.

توضیحات به صورت Comment در داخل برنامه آورده شده است. برای تکمیل شدن قسمت back_end برنامه کلاس دیگری نیاز دارید که بتواند آرایه ای از اشیای WebFavorite را در خود نگهداری کند. این کلاس باید بتواند پوشه Favorites را بررسی کرده و به ازای هر فایل که پیدا می کند یک شیء جدید از نوع WebFavorite متناسب با اطلاعات فایل ایجاد کرده و به آرایه اضافه کند.

```

class WebFavorite
{
    public string Name;
    public string Url;
    public void Load(string FileName)
    {
        // Declare Variable
        string strData;
        string[] strLines;
        FileInfo objFileInfo = new FileInfo(FileName);
        //Set the Name member to the file name
        // minus the extension
        Name = objFileInfo.Name.Substring(0,
            objFileInfo.Name.Length -
            objFileInfo.Extension.Length);
        // Read the input contents of the file
        strData = File.ReadAllText(FileName);
        //Split the lines of data in the file
        strLines = strData.Split('\n');
        // Process each line looking for the URL
        foreach (string strLine in strLines)
        {
            //Does the line of data start with URL =
            if (strLine.StartsWith("URL="))
            {
                //Yes, Set the URL Member to the actual URL
                Url = strLine.Substring(4);
                //Exit the foreach loop
                break;
            }
        }
    }
}

```

یک کلاس داخلی جدید به نام Favorites ایجاد کرده و دستورات زیر را در آن بنویسید. قبل از شروع به نوشتن کد درون این کلاس بهتر است تا بخش های مختلف این کلاس را به صورت زیر بررسی کنید.

۱- فیلدی از نوع ArrayList در برنامه ایجاد می کنیم تا اشیایی که از نوع WebFavorites ایجاد می شوند را در آن قرار دهیم.

۲- در این کلاس به خاصیتی نیاز داریم تا آدرس فولدر Favorites را در کامپیوتر کاربر برگرداند. بنابراین این خاصیت باید از نوع فقط_خواندنی باشد.

۳- در آخر نیز باید متدی به کلاس اضافه کنید که فایل های درون Favorites را بررسی کند. هنگامی که فایلی را پیدا کرد، یک شیء WebFavorite برای آن ایجاد کرده و به ArrayList اضافه کند. در این کلاس دو نسخه از این متد را ایجاد می کنیم، یکی از آنها برای بدست

آوردن آدرس پوشه Favorites، از خاصیت FavoritesFolder در کلاس استفاده می کند، دیگری این آدرس را به عنوان پارامتر دریافت می کند.

```
public class Favorites
{
    //Public Member
    public System.Collections.ArrayList FavoriteCollection = new
    System.Collections.ArrayList();
    public string FavoritesFolder
    {
        get
        {
            //Return the path to the user's Favorites folder
            return Environment.GetFolderPath(Environment.SpecialFolder.Favorites);
        }
    }
    public void ScanFavorites()
    {
        //Scan the favorites folder
        ScanFavorites(this.FavoritesFolder);
    }
    public void ScanFavorites(string FolderName)
    {
        //Process each file in the Favorites folder
        foreach (string strFile in System.IO.Directory.GetFiles(FolderName))
        {
            //If the file has a url extension
            if (strFile.EndsWith(".url", true, null))
            {
                //Create and use a new instance
                //of the webFavorite class
                WebFavorite objWebFavorite = new WebFavorite();
                //Load the file information
                objWebFavorite.Load(strFile);
                //Add the object to the collection
                FavoriteCollection.Add(objWebFavorite);
            }
        }
    }
}
```

همانطور که در فصل چهار مشاهده کردید برای ایجاد یک آرایه که طول مشخصی نداشته باشد و بتواند به راحتی اشیا را به آن اضافه و یا از آن حذف کند، باید از کلاس ArrayList استفاده کنید. در این قسمت نیز شیء ای از این نوع را ایجاد می کنیم تا بتوانیم اشیا را ایجاد شده از نوع WebFavorite که تعداد آنها نیز مشخص نیست را در آن قرار دهیم. در کلاس Favorites متد سربار گذاری شده به نام ScanFavorites ایجاد می کنیم. نسخه دوم این متد آدرس فولدری را به عنوان پارامتر دریافت می کند و درون آن پوشه به دنبال فایل هایی با پسوند url می گردد. اما بهتر است قبل از بررسی نحوه کار این متد، نحوه

کارکرد خاصیت FavoritesFolder را بررسی می کنیم. به دلیل اینکه بر حسب کاربری که هم اکنون در حال استفاده از کامپیوتر است ممکن است آدرس این پوشه Favorites تغییر کند، بهترین راه برای بدست آوردن آدرس این پوشه، پرسیدن آن از ویندوز است. برای این کار می توانیم از متد GetFolderPath که در کلاس System.Environment قرار دارد استفاده کنیم.

این متد یکی از گزینه های شمارنده Environment.SpecialFolder را به عنوان پارامتر دریافت می کند و آدرس آن پوشه را بر می گرداند. شمارنده SpecialFolder حاوی نام تعدادی از پوشه های خاص است که معمولاً هنگام برنامه نویسی زیاد مورد استفاده قرار می گیرد (مانند پوشه My Document و یا Program Files). برای اینکه در برنامه اصلی از کلاس Favorites بخواهیم تا با جستجوی پوشه Favorites، لیست خود را پر کند باید نسخه اول از متد ScanFavorites را فراخوانی کنیم. این نسخه با استفاده از خاصیت FavoritesFolder، آدرس پوشه Favorites را بدست آورده و آن را به عنوان پارامتر به نسخه دوم از این متد می فرستد.

کاری که نسخه دوم از این متد باید انجام دهد این است که فایل های موجود در آدرسی که به آن فرستاده شده است را بدست آورده و آنها را پردازش کند. برای بدست آوردن لیست فایل های موجود در آن پوشه می توانیم از متد GetFiles در کلاس System.IO.Directory استفاده کنیم. این متد آدرس پوشه ای را که باید بررسی کند را به عنوان پارامتر دریافت می کند و آرایه ای از نام فایل های موجود در آن را بر می گرداند. به این ترتیب می توانیم با استفاده از یک حلقه foreach آنها را بررسی کنیم. متغیری را از نوع رشته ای و به نام strFile در حلقه تعریف می کنیم تا با هر بار گردش حلقه نام یکی از فایل ها در آن قرار گیرد.

درون حلقه foreach ابتدا بررسی می کنیم که آیا پسوند فایل برابر با url است یا خیر؟ برای این کار می توانیم از متد EndsWith در کلاس string استفاده کنیم. متد EndsWith یک متد سربار گذاری شده است و نسخه ای که در این برنامه از آن استفاده می کنیم سه پارامتر دریافت می کند. پارامتر اول عبارتی است که باید با انتهای رشته مقایسه شود، در اینجا عبارت "url" را برای این پارامتر استفاده می کنیم. پارامتر دوم یک مقدار بولین است که مشخص می

کند آیا متد EndsWith در هنگام مقایسه حالت حروف را نیز در نظر بگیرد یا خیر؟ در این قسمت نمی خواهیم مقایسه متن به صورت حساس به بزرگی و یا کوچکی حروف انجام شود. پس مقدار true را برای این پارامتر استفاده می کنیم. پارامتر سوم هم شیء ای را از نوع System.Globalization.CultureInfo دریافت می کند، این شیء مشخص کننده تنظیمات محلی است که باید هنگام مقایسه در نظر گرفته شوند، اما در اینجا برای اینکه تنظیمات محلی پیش فرض به کار گرفته شود از عبارت null برای این پارامتر استفاده می کنیم.

اگر فایلی که در حال بررسی است دارای پسوند url باشد باید شیء جدیدی از نوع WebFavorite ایجاد کنید، اطلاعات آن فایل را در داخل شیء قرار دهید و سپس شیء را به آرایه Favorites اضافه کنید. برای این کار ابتدا باید شیء ای را از نوع WebFavorite ایجاد کرده و سپس متد Load را در آن شیء فراخوانی کنید و آدرس فایل را به آن متد بفرستید، در انتها نیز شیء را در آرایه قرار دهید.

در Form اصلی برنامه، روی آن دو بار دابل کلیک کنید تا وارد رویداد Load آن شوید، سپس دستورات زیر را بنویسید.

```
private void Form1_Load(object sender, EventArgs e)
{
    //Create a new instance of the Favorites class
    Favorites objFavorites = new Favorites();
    //Scan the Favorites Folder
    objFavorites.ScanFavorites();
    //Process each objWebFavorite object
    //in the Favorites collection
    foreach (WebFavorite objWebFavorite in objFavorites.FavoriteCollection)
    {
        //Declare a ListViewItem object
        ListViewItem objListView = new ListViewItem();
        //Set the properties of ListViewItem object
        objListView.Text = objWebFavorite.Name;
        objListView.SubItems.Add(objWebFavorite.Url);
        //Add the ListViewItem object to the ListView
        listView1.Items.Add(objListView);
    }
}
```

در متد مربوط به رویداد Form1_Load، ابتدا شیء ای را از نوع Favorites ایجاد کرده و سپس نسخه ای بدون پارامتر از متد ScanFavorites را فراخوانی می کنیم. به این ترتیب از

کلاس می خواهیم که هنگام Load شدن Form، آرایه ای جدید به نام FavoritesCollection ایجاد کرده و به ازای هر یک از گزینه هایی که در پوشه Favorites وجود دارد یک شیء از نوع WebFavorite تشکیل دهد و آن را به آرایه اضافه کند.

بعد از اتمام متد ScanFavorites، آرایه FavoritesCollection با عناصری از نوع WebFavorite پر شده است. حال باید با استفاده از عناصر این آرایه در کلاس Favorites، آیتم های درون لیست را کامل کنیم، برای این کار مانند قسمت های قبل از یک حلقه foreach استفاده می کنیم تا بتوانیم تمام عناصر آرایه را پیمایش کنیم.

به این نکته توجه کنید که آیتم های درون کنترل ListView، اشیائی از کلاس ListViewItem هستند که در یک آرایه قرار دارند. بنابراین برای اینکه یک آیتم به این کنترل اضافه کنیم، باید یک شیء از نوع ListViewItem ایجاد کرده و آن را به کنترل اضافه کنیم. درون حلقه foreach ابتدا شیء ای را از کلاس ListViewItem نمونه سازی کرده و خاصیت Text آن را با توجه به فیلد Name در کلاس WebFavorite کامل می کنیم. سپس آدرس لینک مورد نظر را نیز که در فیلد url قرار دارد به خاصیت SubItems از ListViewItem اضافه می کنیم.

در پایان نیز شیء ListViewItem را به خاصیت Items از کنترل ListView اضافه می کنیم تا در Form نمایش داده شود.

اکنون کافی است که برنامه را اجرا کرده و نتیجه را مشاهده کنید.

تا به اینجا برنامه ما قادر است گزینه های موجود در منوی Favorites مرورگر IE را به صورت یک لیست نمایش دهد. اما هنوز نمی تواند سایتی که این لینک ها به آن اشاره می کنند را باز کند. کنترل listView1 را انتخاب کرده و در پنجره Properties و از بخش رویداد ها، رویداد Click را دابل کلیک کنید تا باز شود، و متد مربوط به آن ایجاد شود. سپس کد های زیر را در آن بنویسید.

```
private void listView1_Click(object sender, EventArgs e)
{
    //Update the link label control Text property
    linkLabel1.Text = "Visit" + listView1.SelectedItems[0].Text;
    //Clear the default hyperlink
    linkLabel1.Links.Clear();
    linkLabel1.Links.Add(6,
        listView1.SelectedItems[0].Text.Length,
        listView1.SelectedItems[0].SubItems[1].Text);
}
```

هنگامی که روی یکی از آیتم های درون کنترل ListView کلیک می کنید، متد مربوط به رویداد Click این کنترل فراخوانی می شود. در این متد کدی را قرار می دهیم تا براساس آیتمی که در لیست انتخاب شده است متن مناسبی را در کنترل LinkLabel نمایش دهد.

همانطور که گفتیم هر یک از آیتم های درون کنترل ListView یک شیء از کلاس ListViewItem است. برای دسترسی به آیتمی که هم اکنون در لیست انتخاب شده است می توانیم از خاصیت SelectedItems استفاده کنیم که به صورت آرایه ای از نوع ListViewItem است. دلیل آرایه ای بودن این خاصیت این است که در کنترل ListView کاربر می تواند بیش از یک آیتم را از لیست انتخاب کند، بنابراین خاصیت SelectedItems باید به صورت یک آرایه باشد تا بتوانیم به تمام آیتم های انتخاب شده دسترسی داشته باشیم. البته در این برنامه، ما فقط آیتم اول را در کنترل LinkLabel نمایش می دهیم بنابراین در آرایه SelectedItems فقط به عنصر اول با اندیس صفر نیاز داریم. برای دسترسی به اطلاعات دیگر ستون های فیلد انتخاب شده در لیست نیز می توانیم از خاصیت SubItems استفاده کنیم. مثلاً برای دسترسی به آدرس لینک انتخاب شده که در ستون اول قرار دارد، از SubItems[۱] استفاده می کنیم.

برای تنظیم متنی که در کنترل Linklabel نمایش داده می شود، خاصیت Text آن را برابر با ثابت رشته ای “ Visit ” به اضافه نام آیتم انتخاب شده در لیست قرار می دهیم. برای دسترسی به نام آیتم انتخاب شده نیز می توانیم از خاصیت Text آن استفاده کنیم.

خاصیت Links از کنترل LinkLabel، به صورت آرایه ای از نوع LinkCollection است و شامل لینک هایی اینترنتی می شود که این کنترل به آنها اشاره می کند. بنابراین باید Url آیتمی

که در لیست `listview1` انتخاب شده است را به این خاصیت اضافه کنیم. ولی بهتر است که محتویات آن را با استفاده از متد `Clear` پاک کنیم.

حال می توانیم با استفاده از متد `Add`، لینک مربوط به آیتم انتخاب شده در لیست را به آن اضافه کنیم. متد `Add`، یک متد سربار گذاری شده است و نسخه ای از این متد که از آن استفاده می نماییم، سه پارامتر دریافت می کند :

پارامتر `Start` : مشخص می شود که از کجای رشته ای که به این متد فرستاده می شود باید به عنوان لینک در نظر گرفته شود.

پارامتر `Length` : مشخص کننده طول رشته ای است که باید به عنوان لینک مشخص شود.

پارامتر `LinkData` : رشته حاوی لینک را مشخص می کند. در اینجا برای اینکه عبارت "Visit" حذف شود، نقطه شروع را ۶ در نظر گرفته ایم، همچنین طول لینک را نیز برابر با طول لینکی که در لیست انتخاب شده است قرار داده ایم.

مجدداً به قسمت طراحی `Form` برگردید و روی کنترل `linkLabel1` دابل کلیک کنید تا متد مربوط به رویداد `LinkClicked` آن ایجاد شود. سپس کد زیر را به این رویداد اضافه کنید :

```
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    System.Diagnostics.Process.Start(e.Link.LinkData.ToString());
}
```

هنگامی که کاربر روی کنترل `LinkLabel` کلیک کند متد مربوط به رویداد `Click` این کنترل فراخوانی می شود، بنابراین در این متد باید دستوری را بنویسیم که محتویات لینک را نمایش دهد. به این متد پارامتری به نام `e` از نوع `LinkLabelLinkClickedEventArgs` فرستاده می شود که حاوی اطلاعاتی مانند آدرس لینک مورد نظر است. برای نمایش لینک کافی است این آدرس را بدست آوریم و آن را به عنوان پارامتر به متد `Start` از کلاس `System.Diagnostics.Process` ارسال کنیم. به این ترتیب، مرورگر اینترنت باز می شود و محتویات لینک انتخاب شده را نمایش می دهد.

در نهایت برنامه را اجرا کرده و نتیجه را بررسی کنید.

طراحی و تولید اشیاء نرم افزاری

مثال ۱: نجاری را در نظر بگیرید که برای ساخت یک میز باید از اشیای مختلفی مثل میخ، چکش، ارّه و... استفاده کند. هر کدام از این ابزارها اشیای جدا گانه ای هستند که بدون دخالت در کار یکدیگر به ساخت میز مورد نظر نجار کمک می کنند. چکش میخ را می کوبد بدون این که از ماهیت میخ با خبر باشد... ارّه چوب را تکه تکه می کند بدون این که در کار چکش دخالت کند.... در نهایت حاصل کار میزی خواهد بود که نجار با استفاده از این ابزارها پدید آورده است. در حقیقت نجار برای ساخت میز خود از اشیای مختلفی که هر کدام ماهیت و خصوصیات خاص خودشان را دارند کمک گرفت.

مثال ۲: فرض کنید قصد داریم قطعه ای را به طور انبوه تولید کنیم، روش اول آن ست که قطعات را یک به یک تراش دهیم ولی این کار ممکن است ماه ها و شاید سال ها طول بکشد و اگر نتیجه کار را هم بررسی کنیم متوجه می شویم که قطعات همگی یک شکل و یک دست نیستند همچنین هزینه تمام شده آن نیز بسیار بالا خواهد بود. اما روش بهتری نیز وجود دارد و آن این است که ما یک قالب برای آن قطعه بسازیم و بعد با استفاده از آن قالب، قطعات مورد نیاز خود را تولید نماییم. پس اگر ما فقط در ساختن قالب دقت لازم را به خرج دهیم در پایان کار مشاهده می کنیم که قطعات تولید شده همگی یک شکل و یک دست از آب درآمده اند. در برنامه نویسی شیء گرا نیز از این روش استفاده می شود، یک قالب (کلاس) طراحی می کنیم، سپس آن را امتحان می کنیم تا از کارکرد صحیح آن مطمئن شده و در پروژه های خود از آن استفاده کنیم.

شیء گرایی در دنیای برنامه نویسی نیز به همین صورت است. برنامه نویس در طول کار خود اشیاء مورد نیاز خود را ایجاد می کند و یا از اشیای آماده ای که توسط برنامه نویسان دیگر ایجاد شده است استفاده می کند، تا با کنار هم قرار دادن آنها برنامه ای را به وجود آورد که از اشیای مختلفی تشکیل شده است. این روش نوین برنامه نویسی دارای ویژگی های زیر می باشد :

زمان توسعه نرم افزار کاهش می یابد : همان طور که گفتیم اکثر پروژه های بزرگ نرم افزاری از زمانبندی خودشان عقب اند. برای کاهش زمان توسعه نرم افزار می توان از ابزار های آماده

موجود در بازار استفاده کرد. در این صورت دیگر نیازی به طراحی و تولید مجدد ابزار ها نیست.

قابلیت اطمینان سیستم افزایش می یابد : کارایی پروژه های بزرگ، پایین است، زیرا ممکن است سیستم به درستی تست نشود. چون اشیاء در سیستم های زیادی استفاده می شوند، خطا های آنها پیدا شده بر طرف می گردد.

انعطاف پذیری افزایش می یابد : برای یک عمل می توان از چندین شیء با توجه به ماهیت آن، استفاده کرد.

کاهش هزینه تولید نرم افزار یا استفاده مجدد از کد ها : وقتی شما یک شیء جدید را خلق می کنید می توانید تا مدت ها از آن استفاده کنید و یا آن را با دیگران به اشتراک بگذارید. این مزیت هنگام ساخت کتابخانه های شیء گرا بسیار کارآمد است.

استفاده موثر از متخصصین : به جای اینکه متخصصین کار های تکراری انجام دهند، می توان از آنها در تولید اشیاء مختلف نرم افزاری استفاده کرد.

اشیاء نرم افزاری را می توان با چندین روش تولید کرد :

۱- مدل شیء CORBA از شرکت OMG

۲- مدل شیء Java Bean از شرکت Sun

۳- مدل شیء COM و DCOM از شرکت Microsoft

چون اصول کاری ما بر اساس نرم افزار و ویژوال استودیو و زبان برنامه نویسی C#.Net است.

مدل اشیاء شرکت Microsoft را بررسی خواهیم کرد.

آشنایی با مدل شیء COM و DCOM

وقتی ویندوز ۳ توسط شرکت مایکروسافت معرفی شد روش اولیه برای برقراری ارتباط بین Application ها تبادل داده داینامیک یا DDE نام داشت. DDE غیر قابل انعطاف بود و باعث بروز مشکلاتی در سیستم می شد، و تنها در یک سیستم قابل استفاده بود. چندین سال برنامه ها برای ارتباط با هم در یک سیستم از DDE استفاده می کردند.

بعد از این سال ها میکروسافت کم کم DDE را کنار گذاشت و استفاده از مدل شیء مشترک COM (Component Object Model) و DCOM (Distributed COM) را پیشنهاد کرد.

COM برای ارتباط برنامه ها در یک سیستم، و DCOM که نوع توزیع یافته ای از COM بود برای ارتباط دستگاه با سرور دور دست استفاده می شد.

یک ائتلاف از چند شرکت بزرگ شامل (Apple, SUN, IBM) یک رابط تبادل داده متفاوت بنام CORBA ایجاد کردند. برخلاف COM، CORBA در فرستادن پیام بین سیستم های مختلف بهتر عمل می کرد. اما متأسفانه در برنامه نویسی بسیار مشکل بود و زمان استفاده کوتاهش اجازه گسترش به آن نداد. در طول این مدت میکروسافت تکنولوژی خود را گسترش میداد و COM+ و MTS و DNA را معرفی کرد.

Common object Model plus (COM+)

Microsoft Transaction Server (MTS)

Distributed Network Architecture (DNA)

این تکنولوژی ها اجازه بیشتری را در ارتباطات پیچیده میان Component ها مانند Dada Pooling و Transaction فراهم می کردند، متأسفانه این تکنولوژی ها نیاز داشتند که هر برنامه اطلاعات کاملی از برنامه مقابل داشته باشد، بنابراین وقتی سیستم عامل ها متفاوت بودند مثل لینوکس و ویندوز این سیستم ها بخوبی کار نمی کردند. این حالت تا سال ۲۰۰۱ (یعنی تا آغاز تکنولوژی Net. که تلفیقی از تکنولوژی COM و انعطاف CORBA است) ادامه داشت. گرچه تکنولوژی Net. اصولاً عضوی از میکروسافت است، اما بنظر می رسد که انعطاف پذیری آن بزودی قابل استفاده در سیستم های عامل دیگر خواهد بود.

نکته!

بسیاری از اشیایی که در دنیای واقعی با آنها سر و کار داریم به همین صورت هستند. به عنوان مثال، اتومبیل، کامپیوتر ها، هواپیما ها و غیره، از قسمت های مختلفی که استاندارد شده اند به وجود می آیند. به عنوان مثال، کامپیوتر متشکل از اجزایی است که قابل تعویض هستند. در مورد نرم افزار هم می توان همینطور فکر کرد. بسته های نرم افزاری موجودند و برنامه نویس می تواند برنامه ها را با استفاده از این بسته های نرم افزار ایجاد کند. مرکز سازنده این بسته ها COM نام دارد. COM در سطح باینری است و مفهوم آن این است که آنها را می توان در

زبان های گوناگون نوشت و به هم ارتباط داد. COM موجب قابلیت انعطاف در ایجاد برنامه های کاربردی می شود. زبان C# همچون سایر زبان های برنامه نویسی شرکت مایکروسافت می تواند از COM استفاده کند. جزئیات COM از دید برنامه نویس پنهان است.

استفاده مجدد از نرم افزار

همان طور که تاکنون مشاهده کردید، برنامه نویسان C# می توانند کلاس های جدیدی بنویسند و از کلاس های موجود در کتابخانه FCL نیز استفاده کنند. FCL مجموعه ای گسترده از کلاس ها را برای هر موردی از دستیابی داده تا طراحی رابط کاربر، امنیت، شبکه و... را ارائه می دهد. کتابخانه FCL همچنین شامل تعاریف مورد نیاز برای تمامی نوع های داده ای اولیه مانند float , int و... است.

برنامه نویسان، نرم افزار خود را از طریق ترکیب کلاس های جدید با کلاس های موجود در FCL که به دقت تست و متد سازی شده اند، ایجاد می کنند. این نوع استفاده مجدد از نرم افزار، توسعه نرم افزار قدرتمند و با کیفیت بالا را سرعت می بخشد. امروزه توسعه سریع نرم افزار (RAD) جذابیت خاصی دارد.

زبان C#.NET برای توسعه سریع نرم افزار RAD و کلاس های FCL توسعه داده شد. دسترسی آسان و سریع به پایگاه داده ها با استفاده از ADO.NET و ایجاد کنترل های Activex از جمله مواردی هستند که این زبان را برای توسعه سریع نرم افزار مناسب کرده اند. برنامه نویسان C#، بر نکات برنامه نویسی سطح بالا تاکید دارند و به جزئیات پیاده سازی سطح پایین کلاس ها در FCL کاری ندارند. به عنوان مثال، برنامه نویسی که یک برنامه گرافیکی در C# می نویسد، نیاز نیست به جزئیات قابلیت گرافیکی سکوی NET. آشنا باشد. در عوض، برنامه نویس C# به یادگیری و به کارگیری کلاس های گرافیکی FCL می پردازد. کلاس های FCL نه تنها به توسعه نرم افزار سرعت می بخشند، بلکه توانایی های کاربر را برای اشکال زدایی و نگهداری نرم افزار ها افزایش می دهد، زیرا از اشیاء امتحان شده و مطمئن استفاده می کنند.

آشنایی با مفهوم Activex

Activex در واقع یک انقلاب در تحولات نرم افزاری محسوب می شود و با عرضه این تکنولوژی از سوی مایکروسافت، در واقع تعریفی تازه در حیطه برنامه نویسی ارائه شد که منجر به افزایش سرعت و کارایی در مباحث توسعه نرم افزاری گردید.

به واسطه این تکنولوژی، درنوردیدن بسیاری از محدودیتها ساده تر از قبل شد و در واقع این تکنولوژی منجر به تعریف ساختاری کاملاً متفاوت و کارآمد در زمینه برنامه نویسی شد. هدف از ارائه این تکنولوژی آماده سازی یک بستر و الگوی ارتباطی استاندارد جهت برقرای ارتباط بین نرم افزارهای مختلف می باشد.

به طور خلاصه، Activex ها مجموعه ای از کدها و کنترلهای از پیش نوشته و طراحی شده ای هستند که امکاناتی را در اختیار برنامه نویسان و توسعه دهنده گان قرار خواهند داد، که به واسطه آنها قادر به انجام فعالیت هایی در محیط های توسعه هستند. به طور مثال در خود محیط ویژوال استودیو، Form برنامه و یا انواع کنترل هایی که در پنجره Toolbox قرار دارند نمونه ای از کنترل های Activex هستند. برنامه نویسان در صورت عدم استفاده از Activex ها مجبور هستند که زمان و انرژی زیادی را جهت تولید آنها بپردازد.

کنترل های Activex دو نوع هستند :

ActiveX Dll - ۱

ActiveX COM - ۲

این دو گونه، جدا از اینکه مرزهای مشترک وسیعی با هم دارند ولی در حالت کلی اختلافی که به وضوح مشهود هست در این نکته ظهور پیدا می کند که Activex COM دارای یک رابط کاربری بصری هست، ولی در Activex Dll بدین گونه نیست.

مفهوم رابط کاربری را اینگونه می توان بیان کرد که تنظیمات Activex COM را می توان از طریق پنجره Properties تغییر داد. در واقع هر نوع Activex ای که قابلیت تنظیم پارامترهای خودش رو در اختیار برنامه نویس قرار دهد الزاماً از نوع Activex COM خواهد بود.

البته در بسیاری موارد Activex COM ها علاوه بر امکان اعمال تنظیمات به روش فوق، امکان پذیرش بسیاری از تنظیمات را از طریق کد نویسی در اختیار برنامه نویس قرار خواهند داد (بخش مشترک با Activex Dll).

Activex COM ها معمولاً بر روی Form ها اضافه می شوند و شما قادر هستید پس از اضافه شدن به برنامه، آنها را ببینید. ولی در Activex Dll ها بدین گونه نیست، Activex Dll ها از طریق بخش References به برنامه اضافه می شوند و شما می توانید نام Activex های اضافه شده را در داخل آن بخش ملاحظه کنید.

امکان برقراری ارتباط با Activex Dll تنها از طریق کد نویسی امکان پذیر است، شما در Activex Dll ها شاهد یک محیط کاربری تعاملی که به واسطه آن امکان درج تنظیمات را نخواهید داشت. در واقع بخشی مانند پنجره Properties موجود برای ActiveX COM وجود ندارد.

آشنایی با مفهوم Component

Component یک جزء نرم افزاری است که قابلیت استفاده مجدد را داراست. از مهمترین اهداف تولید Component به اشتراک گذاری کد در سطح باینری است. کد نویسی مبتنی بر اجزاء (Component Based) این امکان را فراهم می آورد که هر جزء پروژه نرم افزاری خود را مستقل از اجزای دیگر طراحی و توسعه دهید، سپس با کنار هم قرار دادن این اجزاء سیستم نهایی خود را ایجاد کنید. تقسیم بندی اجزای نرم افزاری به واحد های کوچکتر باعث افزایش انعطاف نرم افزاری و نیز نگهداری آسان آن خواهد شد.

برای اینکه بتوانید از قطعات توسعه داده شده توسط خود و یا دیگران استفاده کنید باید از استانداردهای خاصی مانند COM پیروی کنید.

در نهایت باید گفت که Component یک قطعه اجرایی به صورت EXE, DLL و یا OCX با استاندارد های مشخص می باشد که سرویس هایی را در اختیار کاربران قرار می دهد تا بتوان از آن در نرم افزار های مختلف استفاده کرد.

توجه! چون قطعات با پسوند OCX. قدیمی شدند به همین دلیل در این فصل به تولید قطعات DLL می پردازیم.

آشنایی با فایل های DLL



تصویر ۲ - ۶

در کامپیوتر، DLL مجموعه ای از برنامه های کوچک است، که هر کدام می تواند توسط یک برنامه بزرگتر که در کامپیوتر در حال اجراست احضار شود. برنامه کوچکی که به برنامه بزرگتر اجازه برقراری ارتباط با یک وسیله ویژه مثل چاپگر (پرینتر) و یا اسکنر را می دهد همواره به عنوان یک برنامه DLL بسته بندی می شود (معمولاً به "فایل DLL" اشاره می شود). مزیت فایل های DLL این است که، از آنجا که آنها در حافظه کامپیوتر (RAM) با هم به همراه برنامه اصلی بارگذاری نمی شوند، فضا در RAM صرفه جویی می شود. وقتی که یک فایل DLL نیاز است، بعداً بارگذاری و اجرا می شود.

به طور مثال تا مادامی که یک کاربر Microsoft Word یک سند را ویرایش می کند، فایل DLL چاپگر نیاز نیست تا در RAM بارگذاری شود. اگر کاربر تصمیم به پرینت گرفتن آن سند بگیرد، برنامه کاربردی Word باعث می شود تا فایل DLL چاپگر بارگذاری و اجرا شود. به یک فایل DLL اغلب پسوند نام فایل dll. داده می شود. فایل های DLL به صورت پویایی به برنامه ای که از آنها در طول زمان اجرای برنامه استفاده می کند متصل هستند تا اینکه با برنامه اصلی گردآوری شوند.

در واقع دلایل استفاده از DLL ها را می توان موارد زیر نام برد :

۱- توانایی اشتراکی کردن کد بین چند برنامه و حتی خود ویندوز

۲- استفاده مجدد از کدهای نوشته شده

۳- استفاده بهینه از منابع ویندوز و منابع سیستم

۴- جدا کردن کدهای مختلف از هم

نکته!

فایل های DLL خاصیتی مستقل از زبان برنامه نویسی دارند. یعنی اگر DLL با استفاده از زبان C# ایجاد شده باشد می توان در Visual Basic یا در Visual C++ از آن استفاده کرد.

مثال کاربردی ۲

یکی از مباحث و مسائل مهم برای برنامه نویسان و کاربران ایرانی، مساله تاریخ فارسی است. تاریخ میلادی در برنامه های فارسی، زیاد مناسب نیست و کاربردی ندارد. تا قبل از نسخه NET ۲۰۰۵ مجبور بودیم بر روی متد ها و کلاس های تاریخ میلادی تغییراتی را انجام دهیم تا فرمت و قالب آن را به صورت شمسی در آورده و نمایش دهیم. از NET ۲۰۰۵ امکانات جدیدی در اختیار برنامه نویسان از طرف شرکت Microsoft قرار داده شد تا براحتی بتوان از تاریخ قمری و شمسی استفاده کرد.

یکی از مزایا و کاربرد های خوب این متد ها کارکرد صحیح و درست تقویم می باشد، ممکن بود شما برنامه ای برای تبدیل تاریخ قمری به شمسی بنویسید یا از برنامه های آماده سایر برنامه نویسان استفاده کنید، این برنامه ها به خاطر آنکه اصول درست الگوریتم نویسی رعایت نمی شد، در سال های کبیسه دچار مشکل می شد یا اصلا دقت لازم را نداشت.

شرکت Microsoft این مشکل را حل کرد و برای ما کلاس و متد هایی قرار داد که به سادگی بتوانیم تقویم را در حالت قمری و شمسی استفاده کنیم و مشکل خاصی هم در این زمینه وجود ندارد، یعنی تا این لحظه هیچ گزارشی در دنیا مبنی بر دقیق نبودن این تاریخ وجود ندارد.

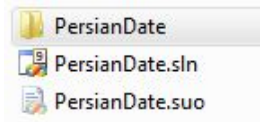
می خواهیم فایل Dll ای تولید کنیم که بتوان با استفاده از آن، از تاریخ شمسی در پروژه های خود استفاده کنیم.

یک پروژه از نوع Class Library ایجاد کرده و عنوان آن را PersianDate قرار دهید. با پروژه های نوع Class Library در فصول گذشته کار کردیم و راجع به آنها صحبت کردیم. نکته! برای استفاده از فرمت های مختلف تاریخ، Code Page های مختلف و زبان های مختلف بایستی دستور using System.Globalization; را در بخش using ها اضافه کنید.

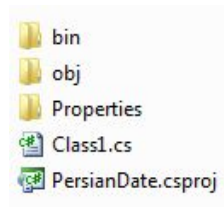
اگر این دستور را ننویسید متد های تعریف کننده تاریخ، تبدیل کننده تاریخ در پروژه، غیر قابل استفاده خواهد بود.

```
public class PersianDate
{
    public static string GetPersianDate()
    {
        //بدست آوردن تاریخ امروز
        DateTime thisDate = DateTime.Now;
        string Rooz="",Mah="",Sal="";
        PersianCalendar pc = new PersianCalendar();
        //تاریخ فعلی را بر می دارد و تبدیل به شمسی می کند
        Sal = pc.GetYear(thisDate).ToString();
        Mah = pc.GetMonth(thisDate).ToString();
        Rooz = pc.GetDayOfMonth(thisDate).ToString();
        return (Sal + "/" + Mah + "/" + Rooz);
    }
}
```

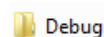
حال از منوی Build گزینه Build Solution را کلیک کنید یا کلید F6 را از صفحه کلید فشار دهید تا فایل dll مربوط به پروژه PersianDate ساخته شود. همانطور که در فصول گذشته گفته شد پروژه های از نوع Class Library خروجی ندارند. برای دسترسی به فایل Dll این پروژه از پوشه محتویات پروژه (پوشه ای که پروژه خود را در آن ساخته و ذخیره کرده اید) بروید.



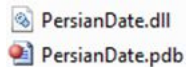
بر روی پوشه ای که عنوان آن مشابه عنوان فایل پروژه است کلیک کنید تا محتویات آن نمایش داده شود. که در اینجا عنوان پوشه PersianDate است.



بر روی پوشه bin کلیک کنید تا محتویات آن نمایش داده شود.

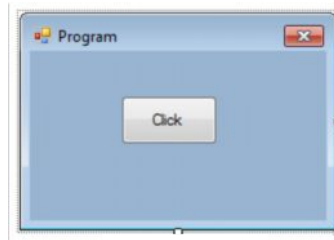


بر روی پوشه Debug کلیک کنید تا محتویات آن نیز نمایش داده شود.



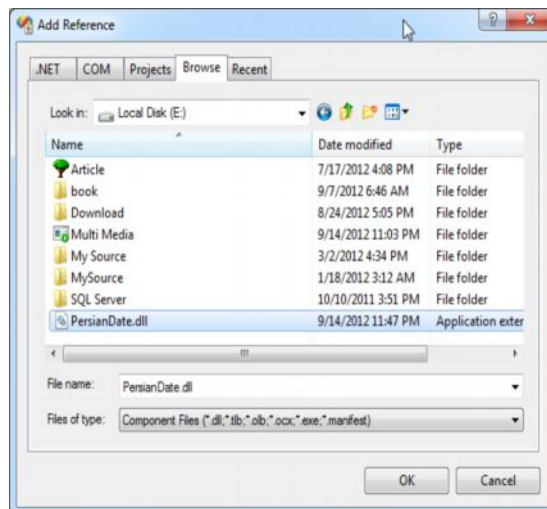
کافی است که فایل PersianDate.dll را کپی کرده و در محلی مناسب از کامپیوتر خود ذخیره کنید. از این فایل DLL می توانید در هر پروژه ای استفاده کنید.

یک پروژه جدید از نوع ویندوز ساخته و Form آن را مطابق پنجره تصویر ۳ - ۶ طراحی کنید.



تصویر ۳ - ۶

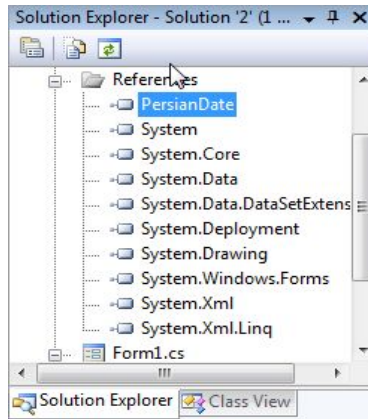
در پنجره Solution Explorer بر روی پوشه References راست کلیک کرده و گزینه Add Reference را کلیک کنید. در پنجره Add Reference به تب Browse رفته و به محلی که در آن فایل DLL خود را ذخیره کرده اید فایل خود را انتخاب کنید (پنجره تصویر ۴ - ۶).



تصویر ۴ - ۶

پس از انتخاب فایل DLL خود بر روی OK کلیک کنید تا این فایل به پروژه شما اضافه شود.

حال اگر به پوشه References در پنجره Solution Explorer کلیک کنید می توانید فایل PersianDate.dll را مشاهده کنید (پنجره تصویر ۵ - ۶).



تصویر ۵ - ۶

حال دوباره به Form برنامه برگشته و بر روی دکمه Click دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(PersianDate.PersianDate.GetPersianDate());
}
```

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

همانطور که در دستورات بالا مشاهده می کنید خودمان را درگیر دستورات بدنه PersianDate نکردیم، ممکن است شما به عنوان برنامه نویس در موقعیت های مختلف باشید. مثلاً شرکتی شما را به عنوان برنامه نویس استخدام کند، شما همیشه بهتر است که یک حافظه جانبی مانند Flash در کنار خود داشته باشید و در آن فایل های DLL خود را ذخیره کنید تا در پروژه هایی که شما در آن شرکت انجام می دهید از انجام کار های تکراری بپرهیزید و به راحتی کار های خود را پیش ببرید.

یکی دیگر از مزایای فایل DLL در محیط NET این است که می تواند Posrtable باشد، یعنی می تواند در جاهای مختلف کپی شود و بتوان براحتی از آن استفاده کرد. و نیز زبان هایی که تحت نرم افزار Visual Studio پشتیبانی می شوند می توانند از این فایل DLL تولید شده شما استفاده کنند.

مثال کاربردی ۳

یک پروژه جدید از نوع ویندوزی ساخته و دو عدد کنترل TextBox روی Form قرار دهید. می خواهیم در زمان اجرا کاربر با زدن کلید Enter به کنترل بعدی منتقل شود. برای این کار می توان از رویداد KeyDown کنترل ها استفاده کرد که در اینجا کنترل ما کنترل TextBox است. با نوشتن دستورات زیر در رویداد KeyDown هر یک از کنترل ها می توان در زمان اجرا با زدن کلید Enter به کنترل بعدی رجوع کرد.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
        SendKeys.Send("{TAB}");
}
```

این دستور کلید Enter را مانند کلید TAB شبیه سازی می کند. حال اگر برنامه را اجرا کرده و کلید Enter را فشار دهید به صورت خودکار به کنترل بعدی می رود.

اگر پروژه شما، پروژه بزرگی بود و در حدود ۳۰ یا ۴۰ کنترل TextBox یا ComboBox داشت چطور؟ از نظر منطقی درست نیست که در رویداد KeyDown هر کدام از کنترل ها دستورات بالا را بنویسیم. بهتر است که ما یک فایل DLL تولید کنیم که حاوی کلاس TextBox باشد و رویداد TextBox آن را خودمان برنامه نویسی کنیم، و مجموعه را به صورت یک بسته و فایل DLL استفاده کنیم.

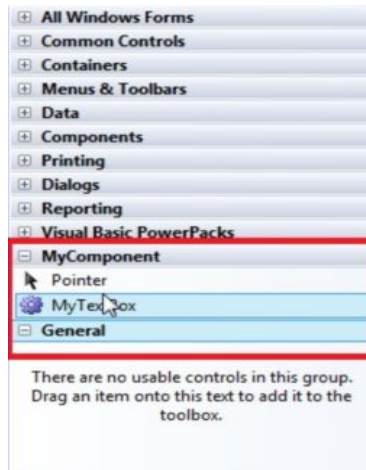
یک پروژه جدید از نوع Class Library ساخته و عنوان آن را MyTextBox قرار دهید. پروژه ای که ما در حال حاضر داریم هیچ مفهوم و شناختی از کنترل TextBox ندارد، پس بایستی ابتدا کلاس TextBox را به این پروژه معرفی کنیم. در پنجره Solution Explorer روی پوشه References راست کلیک کرده و گزینه Add Referenc را انتخاب کنید، در تب NET. گزینه System.Windows.Forms را انتخاب کرده و روی دکمه OK کلیک کنید. حالا دستور using System.Windows.Forms را بنویسید. عنوان کلاس خود را نیز به MyTextBox تغییر می دهیم. همانطور که با مباحث ارث بری هم که آشنا هستید از کلاس TextBox برای کلاس MyTextBox ارث بری می کنیم. یعنی کلاس MyTextBox، تمام خاصیت ها و متد های کلاس TextBox محیط NET را داراست.

همانطور که دیدید تمام دستورات ما در رویداد KeyDown رخ می داد، از متد KeyDown کلاس TextBox به صورت Override استفاده می کنیم.

```
public class MyTextBox : TextBox
{
    protected override void OnKeyDown(KeyEventArgs e)
    {
        base.OnKeyDown(e);
        if (e.KeyCode == Keys.Enter)
        {
            SendKeys.Send("{TAB}");
        }
    }
}
```

حال دکمه F6 را از صفحه کلید فشار دهید تا پروژه Build شود و فایل DLL مورد نظر ساخته شود. طبق روش گفته شده در مثال قبل فایل DLL خود را کپی کرده و در محل مناسبی از کامپیوتر خود ذخیره کنید.

دوباره به پروژه ویندوزی برگشته و در پنجره Toolbox راست کلیک کرده و گزینه Add Tan را انتخاب کنید. برای تب ایجاد شده یک نام تعیین کنید ما عنوان MyComponent را انتخاب می کنیم، حال بر روی تب ایجاد شده راست کلیک کرده و گزینه Choose Items... را انتخاب کنید. این کار کمی زمانبر است زیرا بایستی تمام DLL را جمع کرده و یک جا نشان دهد که کمی طول می کشد. در تب NET Framework Component روی دکمه Browse... کلیک کرده و در پنجره باز شده مسیر فایل DLL که ساختید را بدهید و دکمه Open را کلیک کنید. فایل DLL ساخته شده به پروژه شما اضافه می شود (پنجره تصویر ۶ - ۶). بهتر است فایل های DLL ای که قصد دارید به پروژه خود اضافه کنید را در محلی که پروژه خود را ساخته و ذخیره کرده اید قرار دهید، زیرا پس از این، این فایل ها جزوه فایل های پروژه شما محسوب می شود و در صورت عدم وجود به درستی کار نخواهند کرد.



تصویر ۶-۶

در صورت نیاز کنترل MyTextBox را به هر تعداد که نیاز داشتید به پروژه خود اضافه کنید و مانند سایر کنترل های موجود در پنجره Toolbox استفاده کنید.

مثال کاربردی ۴

یک پروژه از نوع Class Library ایجاد کنید. سپس از طریق پوشه References کلاس System.Windows.Form و کلاس System.Drawing را مانند مثال قبلی به پروژه اضافه کنید و دستورات using System.Windows.Form; و using System.Drawing; را به پروژه خود اضافه کنید. عنوان کلاس را نیز به MyPanel تغییر دهید، سپس دستورات زیر را

بنویسید.

```
public class MyPanel : Panel
{
    Label lbl = new Label();
    string lbltxt = "";
    public string Lbltxt
    {
        get { return lbltxt; }
        set { lbltxt = value; }
    }

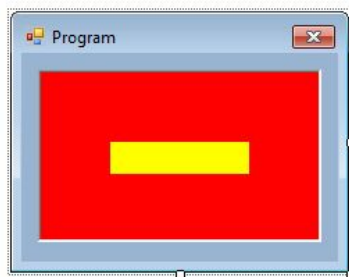
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        this.BackColor = Color.Red;
        this.BorderStyle = BorderStyle.Fixed3D;
        lbl.Text = Lbltxt.Trim();
        lbl.BackColor = Color.Yellow;
        lbl.ForeColor = Color.Black;
        lbl.Location = new Point(50, 50);
        this.Controls.Add(lbl);
    }
}
```

تا به حال با متد های بیشتری کار کردیم و با نحوه override کردن متد ها آشنا شدیم. در دستورات بالا یک کلاس به نام MyPanel ایجاد کردیم که از کلاس Panel ارث بری می کند. رویدادی به نام OnPaint برای اشیائی مثل Panel وجود دارد، این رویداد مربوط به تنظیمات نمایش شیء بر روی صفحه می باشد. ما از این رویداد به صورت override شده استفاده می کنیم. درون این متد override شده برخی از تنظیمات عمومی مانند رنگ پس زمینه و ... را اعمال کردیم.

اما کاری که انجام شده این است که یک متغیر به نام lbl از کلاس Label ایجاد کرده و به وسیله دستور this.Controls.Add(lbl) به کنترل Panel اضافه نموده ایم. در متد OnPaint یک خاصیت به نام lbltext تعریف کردیم و هر آنچه را که به عنوان Text در نظر گرفته شود به عنوان Text آن Label قرار دادیم، این کار را در دستور lbl.Text = lbltext.Trim(); در متد OnPaint انجام دادیم.

حال پروژه را با فشار دادن دکمه F۶ از صفحه کلید اجرا کنید تا فایل DLL مورد نظر ما ساخته شود. فایل DLL ساخته شده را از محل گفته شده در پروژه های قبلی کپی کرده و در محلی از کامپیوتر خود ذخیره کنید.

یک پروژه ویندوز جدید ایجاد کرده و از طریق پنجره Toolbox در ناحیه خالی کلیک راست کرده و گزینه Choose Items... را کلیک کنید و از طریق تب .NET Framework Component از طریق دکمه Browse فایل DLL خود را در محلی که ذخیره کرده اید انتخاب کرده و دکمه Open و سپس OK را کلیک کنید تا به پروژه اضافه شود. حال این کنترل را از پنجره Toobox به Form خود اضافه کنید (پنجره تصویر ۷ - ۶).



تصویر ۷ - ۶

از این به بعد هر زمان که این کنترل را روی Form خودمان قرار دهیم یک Panel با یک کنترل Label خواهیم داشت. می خواهیم در داخل Label موجود در کنترل Panel مقداری را نمایش دهیم. در رویداد Form\Load دستورات زیر را می نویسیم :

```
private void Form1_Load(object sender, EventArgs e)
{
    myPanel1.Lbltxt = "Ardebil";
}
```

در دستورات بالا خاصیت Lbltxt از کلاس mypanel\ با مقدار Ardebil مقدار دهی شده است. اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال کاربردی ۵

برنامه ای که قصد داریم در این بخش بنویسیم خیلی کاربردی تر است. بار ها در طراحی پروژه های پایگاه داده در محیط های مختلف مانند ویندوز و وب برای برنامه نویسان مشکل ساز می شود طراحی TextBox هایی که تنها فقط کاراکتر دریافت کنند و اجازه ورود سایر کاراکتر ها را به کاربران ندهد. در محیط NET. چنین کنترلی تحت عنوان MaskedTextBox وجود دارد، که در صورت تنظیم خاصیت Mask این کنترل می توانید برای آن حالت مختلف دریافت کاراکتر را تنظیم کنید. ولی تنظیم خاصیت باعث بروز یک خط در داخل این کنترل می شود و مشکلاتی در طراحی و برنامه نویسی برای برنامه نویس ایجاد می کند. به همین خاطر بعضی از برنامه نویسان ترجیح می دهند که خود Component ای طراحی کنند که فقط عدد از کاربر دریافت کند.

می خواهیم Component ای بنویسیم که تنها عدد از کاربر دریافت کند. یک پروژه جدید از نوع Class Library ایجاد کرده و عنوان آن را intTextBox قرار دهید. باز هم از طریق پوشه References در پنجره Solution Explorer کلاس System.Windows.Form را به پروژه خود اضافه کنید. و دستور using System.Windows.Form; را بنویسید.

```
public class intTextBox : TextBox
{
    protected override void OnKeyPress(KeyPressEventArgs e)
    {
        base.OnKeyPress(e);
        e.Handled = true;
        if (char.IsDigit(e.KeyChar))
        {
            e.Handled = false;
        }
    }
}
```

متد IsDigit یک مقدار بولین برای ما بر می گرداند به شرطی که عبارت داخل پارانتز IsDigit عددی باشد مقدار True، در غیر اینصورت مقدار False برای ما برگردانده می شود. یک مساله هنوز باقی است و آن این است که وقتی کلید BackSpace را از صفحه کلید فشار می دهیم، عمل حذف کاراکتر ها را انجام می دهد. می توانیم با اندکی کد نویسی جلوی این امر را بگیریم.

```
public class intTextBox : TextBox
{
    protected override void OnKeyPress(KeyPressEventArgs e)
    {
        base.OnKeyPress(e);
        e.Handled = true;
        if (char.IsDigit(e.KeyChar) || (e.KeyChar.ToString() == Keys.Back.ToString()))
        {
            e.Handled = false;
        }
    }
}
```

کد ها را به نحوی تغییر دادیم که اگر ورودی اطلاعات عدد بود مقدار Handled برابر False و در غیر آن مقدار Handled برابر True خواهد بود. حال با استفاده از دکمه F6 در صفحه کلید، پروژه را اجرا کنید تا فایل DLL ساخته شود.

یک پروژه جدید از نوع ویندوزی اجرا کرده و طبق روش های گفته شده در مثال قبل این فایل DLL را به پنجره Toolbox اضافه کرده و سپس روی Form قرار دهید. کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

ساخت این کنترل ها کار ساده ای است متد ها را override می کنیم و درون متد هر چیزی را که نیاز داریم به صورت استاندارد می نویسیم. در حال حاضر Component های پیچیده ای

توسط شرکت های بزرگ تجاری تولید شده اند و در اختیار برنامه نویسان قرار دارند. شما به عنوان برنامه نویس می توانید از آنها استفاده کنید، یعنی هر چه قدر قدرت و مهارت شما در برنامه نویسی شیء گرا بیشتر شود می توانید Component های جالب و حرفه ای تری تولید کنید.

کنترل های ویندوزی

ممکن است ابتدا از خود سوال کنید ایجاد کنترل های سفارشی چه دلیلی ممکن است داشته باشد. خوب، دلایل زیادی برای ایجاد کنترل های سفارشی تحت ویندوز وجود دارند :

با استفاده از کنترل های سفارشی می توانید از یک کنترل در چند قسمت برنامه و یا حتی در چندین برنامه مختلف استفاده کنید، بنابراین مقدار کد مورد نیاز به شدت کاهش می یابد (استفاده مجدد از کد).

می توانید کد های مربوط به یک کنترل را در کلاس همان کنترل قرار دهید و به این ترتیب باعث شوید که کد برنامه بسیار واضح تر شود و ساده تر بتوان آن را درک کرد. برای مثال می توانید کنترل Button را به صورتی ایجاد کنید که بتواند رویداد Click خود را کنترل کند، به این ترتیب دیگر نیازی ندارید این رویداد را در کد مربوط به Form برنامه کنترل کنید.

برای استفاده مجدد از کنترل ها در برنامه ها دو روش کلی وجود دارد :

روش اول این است که سورس اصلی کنترل را به هر برنامه ای که می خواهید از کنترل در آن استفاده کنید اضافه کرده و سپس برنامه را کامپایل کنید. به این ترتیب کد مربوط به کنترل در فایل اجرایی برنامه قرار می گیرد. در طی این فصل به علت سادگی این روش از آن استفاده خواهیم کرد تا بتوانیم بیشتر تمرکز خود را بر نحوه عملکرد کنترل ها متمرکز کنیم.

روش دوم ایجاد کتابخانه کنترل است. کتابخانه های کنترل همانند کتابخانه های کلاس هستند. در حقیقت کتابخانه های کنترل، کتابخانه های کلاسی هستند که دارای یک رابط کاربری نیز می باشند. همانند کتابخانه های کلاس، کتابخانه های کنترل نیز در فایل اسمبلی جداگانه مربوط به خودشان قرار می گیرند و می توانید بعد از ثبت آنها در GAC از آنها در چندین برنامه استفاده کنید. این روش بسیار بهتر از روش قبلی است، زیرا به این ترتیب می توانید

فایل DLL مربوط به این کنترل ها را در اختیار دیگر برنامه نویسان قرار دهید تا بتوانند از آن در برنامه های خود استفاده کنند.

ایجاد و امتحان کنترل های سفارشی

هنگام طراحی یک برنامه ممکن است به کنترلی نیاز داشته باشید که به یک بانک اطلاعاتی وصل شود و اطلاعات خاصی مانند نام کاربری و کلمه عبور یک کاربر را استخراج کند. اگر بخواهید یک کنترل قوی و کارآمد برای این کار ایجاد کنید، باید به نحوی آن را طراحی کنید که آن کنترل در بیشتر موارد قابل استفاده باشد و همچنین کاربر بتواند به راحتی آن را تنظیم کند تا وظیفه مد نظر او را انجام دهد. در این موارد بهتر است اموری را مانند متصل شدن به بانک اطلاعاتی، دریافت نتایج مورد نیاز و قرار دادن نتایج بدست آمده در کنترل های دیگر را دور از چشم کاربر انجام دهید، به صورتی که کاربرانی که از کنترل شما استفاده می کنند در رابطه با این موارد هیچ اطلاعی نداشته باشند. به این ترتیب می توانید از درگیر شدن آنها در این گونه موارد جلوگیری کنید و به آنها اجازه دهید که روی مسائل و وظایف مربوط به خودشان تمرکز کنند.

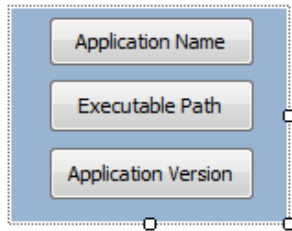
طراحی یک کنترل سفارشی از پایه، کار سختی نیست. از جهتی انجام چنین کاری مشابه طراحی یک Form در برنامه های ویندوزی است. در این قسمت می خواهیم یک برنامه تحت ویندوز ایجاد کنیم که از یک کنترل سفارشی استفاده می کند.

نکته! کنترل های سفارشی که در طراحی آنها از چندین کنترل دیگر استفاده شده است عموماً به عنوان کنترل های متراکم شناخته می شوند.

مثال کاربردی ۶

نرم افزار ویژوال استودیو را اجرا کرده و از منوی File گزینه New و سپس Project را کلیک کنید. در پنجره New Project در بخش Template گزینه Windows Forms Control Library را انتخاب کنید، سپس در قسمت Name عنوان مورد نظر را وارد کنید که ما عنوان MyNamespaceControl را انتخاب می کنیم. در بخش Location محل ساخت و ذخیره پروژه خود را تعیین کنید و روی دکمه OK کلیک کنید تا پروژه ساخته شود. همانطور که

مشاهده می کنید یک Form شبیه Form های پروژه های ویندوزی باز می شود که فاقد دکمه های کنترلی و نوار است. Form را مطابق پنجره تصویر ۸ - ۶ طراحی کنید.



تصویر ۸ - ۶

بر روی دکمه Application Name دابل کلیک کرده و وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ProductName);
}
```

حال بر روی دکمه Executable Path دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

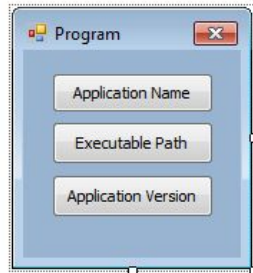
```
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ExecutablePath);
}
```

بر روی دکمه Executable Version دابل کلیک کرده و وارد رویداد Click آن شوید، سپس دستورات زیر را بنویسید.

```
private void button3_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ProductVersion);
}
```

دستورات دکمه Application Name با استفاده از توابع استاتیک موجود در کلاس Application، نام برنامه ای که در حال اجراست را در یک کادر پیغام نمایش می دهد. دستورات دکمه Executable Path با استفاده از خاصیت ExecutablePath از کلاس Application مسیر برنامه در حال اجرا را بدست آورده و آن را در یک کادر پیغام نمایش می دهد. دستورات دکمه Application Version نسخه برنامه مورد استفاده را نمایش می دهد.

برنامه را اجرا کرده و نتیجه را بررسی کنید. خروجی برنامه مطابق تصویر زیر در یک Form نمایش داده می شود که دارای پنجره ای مشابه پنجره Properties است (پنجره تصویر ۹ - ۶).

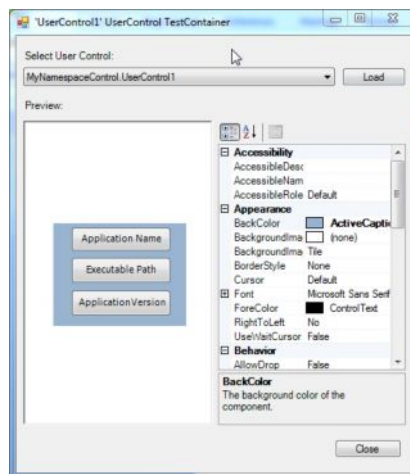


تصویر ۹ - ۶

اگر به مسیر گفته شده در پروژه های از نوع Class Library در همین پروژه نیز طی کنید می توانید فایل DLL تولید شده را مشاهده کنید.

می خواهیم از این فایل DLL تولید شده در یک پروژه ویندوزی استفاده کنیم، فایل DLL تولید شده را در یک مکان مناسب ذخیره کنید تا بتوانید از آن در پروژه ویندوزی استفاده کنید.

یک پروژه جدید ویندوزی ایجاد کرده و طبق روش گفته شده این کنترل را به پنجره Toolbox اضافه کنید. سپس آن را بر روی Form قرار دهید (پنجره تصویر ۱۰ - ۶).



تصویر ۱۰ - ۶

اکنون برنامه را اجرا کرده و نتیجه را بررسی کنید.

مثال کاربردی ۷

یک کنترل سفارشی همانند یک کلاس ایجاد می شود. بنابراین می توانیم تمام عضو هایی را که برای یک کلاس ایجاد می کردید برای یک کنترل سفارشی نیز ایجاد کنید. به عبارت دیگر می توانید خاصیت ها، متد ها و یا رویداد هایی را که به یک کنترل سفارشی اضافه کنید تا افرادی که آن کنترل را در برنامه خود به کار می برند بتوانند از آنها استفاده کنند. ابتدا نحوه اضافه کردن یک خاصیت را به کنترل سفارشی مشاهده بررسی می کنیم.

کنترل های سفارشی می توانند دارای دو نوع خاصیت باشند، خاصیت هایی که می توانند در زمان طراحی و با استفاده از پنجره Properties تغییر داده شوند و خاصیت هایی که باید با استفاده از کد نویسی و در زمان اجرا تغییر داده شوند. برای مثال ممکن است بخواهید در زمان طراحی با استفاده از خاصیت های یک کنترل، رنگ مورد استفاده در آن و یا فونت نوشته های آن را تغییر دهید. اما بخواهید در زمان اجرای برنامه با استفاده از خاصیت ها برای مثال به آدرس بانک اطلاعاتی که به آن متصل شده اید دسترسی داشته باشید.

دوباره به پروژه MyNamespaceControl بروید. برای اینکه بتوانید یک خاصیت را به کنترل اضافه کنید، ابتدا باید یک فیلد در کلاس مربوط به آن کنترل ایجاد کنید تا مقدار آن خاصیت را نگهداری کند. به ویرایشگر کد MyNamespaceControl رفته و کد مشخص شده در زیر را در بخش public partial بنویسید.

```
public partial class UserControl1 : UserControl
{
    //private member
    private string strApplicationName = "";

    public string ApplicationName
    {
        get { return strApplicationName; }
        set { strApplicationName = value; }
    }
}
```

ابتدا یک فیلد به نام strApplicationName تعریف کردیم، سپس یک خاصیت به نام StrApplication تعریف کردیم که به وسیله آن می توان مقدار موجود در این فیلد را تغییر داد. برای اینکه مقدار موجود در این خاصیت در نوار عنوان کادر های پیغام نمایش داده شود،

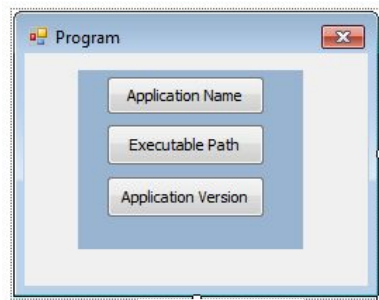
باید پارامتر Caption از متد Show را با مقدار این خاصیت تنظیم کنیم. برای این کار کد موجود در رویداد کلیک هر سه کنترل Button موجود در Form را به صورت زیر تغییر دهید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ProductName,
        this.ApplicationName);
}

private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ExecutablePath,
        this.ApplicationName);
}

private void button3_Click(object sender, EventArgs e)
{
    MessageBox.Show("Application Name is : " + Application.ProductVersion,
        this.ApplicationName);
}
```

برای اینکه بتوانید از خاصیت اضافه شده در کنترل استفاده کنید، کافی است یک بار پروژه را با فشار دادن دکمه F6 از صفحه کلید Build کنید تا تغییرات اعمال شود و از فایل DLL تولید شده آن استفاده کنید. می توانید به همین پروژه جاری خود یک Form ویندوزی ایجاد کنید تا دیگر لازم نباشد فایل DLL را کپی کرده و در جاهای دیگر استفاده کنید. در پنجره Solution Explorer در بالاترین قسمت بر روی Solution راست کلیک کرده و از منوی Add گزینه New Project را انتخاب کنید در پنجره Add New Project گزینه Application Windows Form را انتخاب کرده و بر روی دکمه OK کلیک کنید، یک پروژه ویندوزی ایجاد می شود. حال کافی است که در پنجره Toolbox کنترل UserControl1 را به داخل Form خودتان بکشید (پنجره تصویر ۱۱ - ۶).



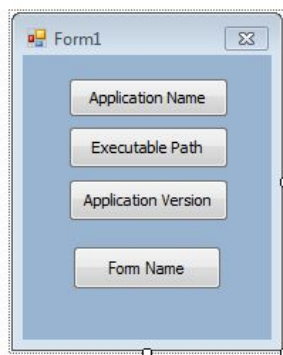
تصویر ۱۱ - ۶

در پنجره Solution Explorer روی Solution Windows Form Application راست کلیک کرده و گزینه Set as StartUp Project را کلیک کنید، سپس برنامه را اجرا کرده و نتیجه را بررسی کنید.

همانطور که ممکن است تاکنون حدس زده باشید، وقتی می توانید یک خاصیت را برای یک کنترل سفارشی ایجاد کنید، مسلماً می توانید یک متد را نیز به آن اضافه کنید. تمام کاری که برای اضافه کردن یک متد به کنترل باید انجام دهید این است که یک متد از نوع public را به کنترل اضافه کنید. به این ترتیب می توانید آن متد را در زمان کار با کنترل فراخوانی کنید. در پنجره Solution Explorer بر روی UserControl1.cs کلیک کنید و سپس به ویرایشگر کد آن بروید و متد زیر را به کلاس UserControl1 اضافه کنید.

```
public string FormCaption()
{
    return Application.OpenForms[0].Text;
}
```

متدی که در این قسمت ایجاد می کنیم عمل خاصی را انجام نمی دهد، فقط با استفاده از خاصیت openForms در کلاس Application لیستی از تمام Form هایی از برنامه که هم اکنون باز هستند را دریافت می کند و نام اولین Form را در یک کادر پیام نمایش می دهد. حال دوباره به پروژه ویندوزی برگشته و بر روی Form1 کلیک کنید تا باز شود، از طریق پنجره Toolbox یک کنترل button بر روی Form قرار دهید و خاصیت Text آن را برابر Form Name قرار دهید (پنجره تصویر ۱۲ - ۶).



تصویر ۱۲ - ۶

روی دکمه Form Name دابل کلیک کنید تا وارد رویداد Click آن شوید، سپس دستورات زیر را در آن بنویسید.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(userControl11.FormCaption(), "Form1");
}
```

اکنون کافی است که برنامه را اجرا کرده و نتیجه را بررسی کنید.

منابع و مآخذ

منابع و مآخذ فارسی

- ۱- آموزش گام به گام C#.NET، عین الله جعفرنژاد قمی، رمضان عباس نژاد، انتشارات علوم رایانه، ویراست سوم، پائیز ۱۳۸۶.
- ۲- کتاب الکترونیکی آموزش C# ۲۰۰۵، محمد هاشمیان
- ۳- نرم افزار آموزشی زبان برنامه نویسی C#.NET ۲۰۰۸، شرکت گسترش دنیای نرم افزار.
- ۴- آموزش گام به گام C#.NET، مهرداد توانا، عاطفه شیجونی، انتشارات مؤسسه فرهنگی هنری نقش سیمرخ، چاپ هفتم، زمستان ۱۳۸۸.
- ۵- نرم افزار آموزشی زبان برنامه نویسی C#.NET سطح مقدماتی و پیشرفته، شرکت نرم افزاری پارسیان.
- ۶- شیء گرایی در C#.NET، امیر دربندی، شهره ابراهیم نژاد، انتشارات ناقوس، چاپ اول، ۱۳۸۸.
- ۷- نرم افزار آموزشی OOP در زبان C#.NET، شرکت نرم افزاری بزرگراه.
- ۸- کتاب الکترونیکی برنامه سازی پیشرفته، نیک محمد بلوچی زهی.
- ۹- نرم افزار آموزشی برنامه نویسی C#.NET، شرکت نرم افزاری مانا.
- ۱۰- راهنمای برنامه نویسی C#.NET ۲۰۰۸، بهتاش مرادی شکر، امین محمدی، دیباگران تهران، چاپ اول، خرداد ماه ۱۳۸۸.
- ۱۱- نرم افزار آموزشی زبان C#.NET، شرکت مهرگان مهر.
- ۱۲- نرم افزار آموزشی ساخت دیکشنری در زبان C#.NET، شرکت گسترش دنیای نرم افزار.
- ۱۳- کتاب الکترونیکی گزیده کتاب ویژوال C# جان شارپ، مهدی محبیان.
- ۱۴- کتاب الکترونیکی برنامه نویسی پیشرفته در C#، مسعود نظیفی اصل، جلد اول.
- ۱۵- کتاب الکترونیکی راهنمای برنامه نویسی شیء گرا در C#، استاد شکاری، شهرام برنا.
- ۱۶- نرم افزار آموزشی زبان C#.NET، شرکت بهکامان.
- ۱۷- نرم افزار آموزشی زبان C#.NET، شرکت مهرگان.

۱۸- حل مسائل C# (مرجع کامل)، رمضان عباس نژاد ورزی، انتشارات فناوری نوین، چاپ اول، زمستان ۱۳۸۸.

۱۹- آموزش برنامه نویسی GDI در محیط C#، میثم ناظمی، انتشارات کیان رایانه، چاپ اول، ۱۳۹۰.

۲۰- ساختمان داده ها در C#، عین الله جعفرنژاد قمی، انتشارات علوم رایانه، چاپ اول، زمستان ۱۳۸۸.

۲۱- آموزش مبانی کامپیوتر و برنامه نویسی به زبان C++، ناصر قاسم آقائی، مهدی جابرزاده انصاری، علی دهقان، شرکت ناقوس اندیشه، چاپ اول، ۱۳۸۶.

۲۲- زبان برنامه نویسی Visual Basic ۶.۰ جلد دوم، منصور ولی نژاد، موسسه فرهنگی هنری دیباگران تهران، چاپ اول، ۱۳۸۴.

۲۳- درس و کنکور زبان C، حمید رضا مقسمی، انتشارات گسترش علوم پایه، چاپ نهم، ۱۳۸۵.

۲۴- مبانی برنامه سازی، فنی حرفه ای (گروه تحصیلی کامپیوتر)، ۱۳۸۳.

۲۵- برنامه نویسی به زبان پاسکال، عین الله جعفرنژاد قمی، انتشارات علوم پایه، چاپ پانزدهم، پاییز ۱۳۸۴.

۲۶- سایت اینترنتی www.barnamenevis.org

۲۷- سایت اینترنتی www.csharpblog.blogfa.com

۲۸- سایت اینترنتی www.hamkelasi.com

۲۹- سایت اینترنتی www.sourcegozar.com

۳۰- سایت اینترنتی www.sourcebaran.com

۳۱- سایت اینترنتی www.hpkclasses.ir

۳۲- حل تمرین های C به همراه فلوچارت؛ غلامرضا رحیمی (افشین)، موسسه فرهنگی، الماس دانش - ترقی - آموزشگان، چاپ اول، زمستان ۱۳۸۷.

منابع و مآخذ انگلیسی

۱- Deitel, "Visual C# ۲۰۰۵ How to program", Prentice hall, ۲۰۰۶, ۲e.

۲- John Sharp, "Microsoft Visual C# step by step", Microsoft press, ۲۰۰۵.

۳- Website (www.msdn.com).

۴- Website (www.dotnetperls.com).

۵- Website (www.codeproject.com).

۶- Website (www.c-sharpcorner.com).

۷- Website (www.sourcecode.com).

۸- Andrew Troelsen, “C# and the .NET platform”, Apress, ۲d ed, ۲۰۰۳