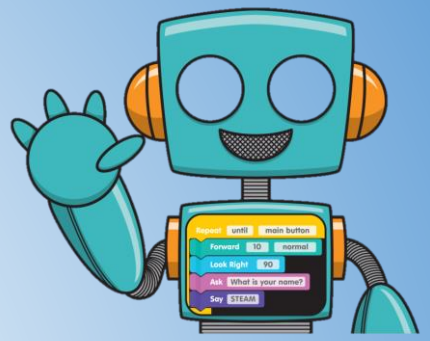


درس برنامه نویسی پیشرفته ۲

برنامه نویسی به زبان سی شارپ (C#)

مدرس: مسعود بایمانی

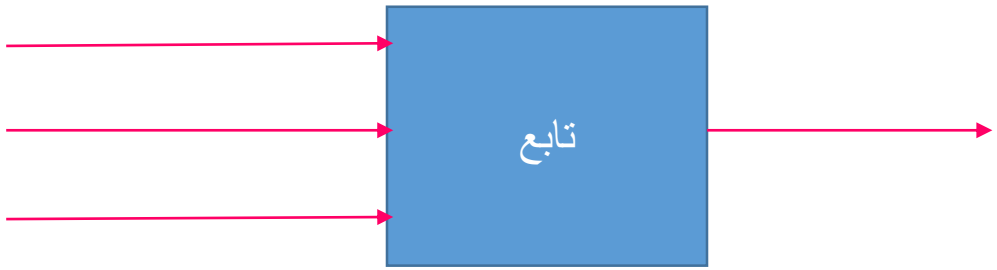
فصل پنجم

متدها، ساختار و کلاس ها

متد

- متد مجموعه ای از دستورات است که منظور محقق سازی یک هدف معین در کنار هم قرار گرفته اند. متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. متدها را به منظور قطعه قطعه کردن برنامه ها و ایجاد واحدهای مستقل که درک و بکارگیری آن‌ها آسان تر است استفاده می نمایند.
- ساختار یک متد به صورت زیر است:

```
returnType MethodName(Datatype parameter1, Datatype parameter2, ... )
{
    code to execute;
    return returnValue;
}
```



نکاتی در مورد متدها

- متد در داخل کلاس تعریف می‌شود.
- مکان تعریف متد در داخل کلاس مهم نیست.
- نمی‌توان یک متد را در داخل متد دیگر تعریف کرد.
- وقتی که شما در برنامه یک متد را صدا میزنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند.
- در سی‌شارپ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند باید از کجا شروع شوند، این متد Main () نام دارد.
- متدها دارای آرگومان‌هایی هستند که وظیفه متد را مشخص می‌کنند.
- پارامترها همان چیزهایی هستند که متد منتظر دریافت آن‌ها است در حالی که آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.
- گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار میروند.

مثال ۱

```
using System;
public class Program
{
    public static void Main()
    {
        PrintMessage();
    }

    static void PrintMessage()
    {
        Console.WriteLine("Hello World!");
    }

}
```

فراخوانی متد

تعریف متد

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

نکات مهم

- کلمه static به طور ساده به این معناست که می‌توان از متد استفاده کرد بدون اینکه از کلاس نمونه‌ای ساخته شود.
- متد Main() همواره باید به صورت static تعریف شود، چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می‌کند.
- متدها می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند.
- returnType در اینجا نوع داده ای مقدار برگشتی (int, float, double,...) را مشخص می کند.
- نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود.
- نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود. اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه void قبل از نام متد استفاده کنیم.

مثال ۲ : مقدار برگشتی از یک متد

```
using System;
public class Program
{
    static int CalculateSum( int firstNumber , int secondNumber )
    {
        int sum = firstNumber + secondNumber;
        return sum;
    }

    public static void Main()
    {
        int result = CalculateSum(10, 5);
        Console.WriteLine("Sum is {0}.", result);
    }
}
```

مثال ۳ : مقدار برگشتی از یک متد

```
using System;
public class Program
{
    static int GetNumber()
    {
        int number;
        Console.Write("Enter a number greater than 10: ");
        number = Convert.ToInt32(Console.ReadLine());
        if (number > 10)
        {
            return number;
        }
        else
        {
            return 0;
        }
    }
    public static void Main()
    {
        int result = GetNumber();
        Console.WriteLine("Result = {0}.", result);
    }
}
```

هنگامی که میخواهیم در داخل یک متد از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به این مثال توجه کنید.

مثال ۴ : مقدار برگشتی از یک متد

```
using System;
public class Program
{
    static void TestReturnExit()
    {
        Console.WriteLine("Line 1 inside the method TestReturnExit()");
        Console.WriteLine("Line 2 inside the method TestReturnExit()");
        return;
        //The following lines will not execute
        Console.WriteLine("Line 3 inside the method TestReturnExit()");
        Console.WriteLine("Line 4 inside the method TestReturnExit()");
    }
    public static void Main()
    {
        TestReturnExit();
        Console.WriteLine("Hello World!");
    }
}
```

کنترل اجرا با رسیدن به دستور return از بدنه متد خارج می شود و به سراغ ادامه کد در تابع اصلی برنامه می رود. این رو محل قرارگیری دستور return در برنامه بسیار حائز اهمیت است. به طور نمونه در مثال بالا دو خط آخر متد TestReturnExit هرگز اجرا نمی شوند.

پارامترها و آرگومان‌ها

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

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

    return returnValue;
}
```

پارامترها و آرگومان‌ها

- آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومان‌ها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومان‌ها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود.

```
using System;
public class Program
{
    static void ShowMessageAndNumber(string message, int number)
    {
        Console.WriteLine(message);
        Console.WriteLine("Number = {0}", number);
    }
    public static void Main()
    {
        ShowMessageAndNumber("Hello World!", 100);
        ShowMessageAndNumber(100, "Welcome to Gimme C#!");
    }
}
```

بروز خطا در برنامه به دلیل عدم
رعایت ترتیب ارسال آرگومان ها

پارامترها و آرگومان‌ها

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

void AnotherMethod(int number)
{
    Console.WriteLine(number + 2 );
}

public static void Main()
{
    AnotherMethod(MyMethod());
}
```

چون مقدار برگشتی متد MyMethod() عدد ۵ است و به عنوان آرگومان به متد AnotherMethod() ارسال می شود

خروجی کد مقابل هم عدد ۷ است.

نامیدن آرگومان‌ها

- یکی دیگر از راه‌های ارسال آرگومان‌ها استفاده از نام آن‌هاست. استفاده از نام آرگومان‌ها شما را از به یادآوری و رعایت ترتیب پارامترها هنگام ارسال آرگومان‌ها راحت می‌کند. در عوض شما باید نام پارامترهای متد را به خاطر بسپارید ولی از آنجا که ویژوال استودیو Intellisense دارد حتی نیازی به این کار نیست.
- استفاده از نام آرگومان‌ها خوانایی برنامه را بالا می برد، چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است. نامیدن آرگومان‌ها در سی‌شارپ ۲۰۱۰ مطرح شده است و اگر شما از نسخه‌های قبلی مانند سی‌شارپ ۲۰۰۸ استفاده کنید، نمی‌توانید از این خاصیت استفاده نمایید. در زیر نحوه استفاده از نام آرگومان‌ها وقتی که متد فراخوانی می‌شود، نشان داده شده است:

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

نامیدن آرگومان‌ها

```
using System;
public class Program
{
    static void SetSalaries(decimal jack, decimal andy, decimal mark)
    {
        Console.WriteLine("Jack's salary is {0:C}.", jack);
        Console.WriteLine("Andy's salary is {0:C}.", andy);
        Console.WriteLine("Mark's salary is {0:C}.", mark);
    }
    public static void Main()
    {
        SetSalaries(jack: 120, andy: 30, mark: 75);
        Console.WriteLine();
        SetSalaries(andy: 60, mark: 150, jack: 50);
        Console.WriteLine();
        SetSalaries(mark: 35, jack: 80, andy: 150);
    }
}
```

```
Jack' salary is $120.
Andy's salary is $30.
Mark's salary is $75.
Jack's salary is $50.
Andy's salary is $60.
Mark's salary is $150.
Jack's salary is $80.
Andy's salary is $150.
Mark's salary is $35.
```

ارسال آرایه به عنوان آرگومان

- آرایه ها را می توان به عنوان آرگومان به متد ارسال کرد. برای انجام این کار ابتدا می بایست پارامترهای متد را طوری تعریف کرد که آرایه دریافت کنند. به این مثال توجه کنید.

```
using System;
namespace ArraysAsArgumentsDemo1
{
    public class Program
    {
        static void TestArray(int[] numbers)
        {
            foreach (int number in numbers)
            {
                Console.WriteLine(number);
            }
        }

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

ارسال آرایه به عنوان آرگومان

نکته مهم:

آرایه ها هم به روش ارجاع به متدها ارسال می شوند. یعنی به جای آنکه یک کپی از آرایه به متد ارسال شود، آدرس محل ذخیره سازی آرایه به متد ارسال می گردد. از این رو با هر تغییر در مقادیرخانه های آرایه در بدنه متد، عیناً شاهد تغییر در مقدار خانه های آرایه در برنامه اصلی خواهیم بود.

ارسال آرایه به عنوان آرگومان

```
using System;
namespace ArraysAsArgumentsDemo2
{
    public class Program
    {
        static void IncrementElements(int[] numbers)
        {
            for (int i = 0; i < numbers.Length; i++)
                numbers[i]++;
        }
        public static void Main()
        {
            int[] array = { 1, 2, 3, 4, 5 };
            foreach (int num in array)
                Console.Write(num+" ");

            Console.WriteLine(" ");
            IncrementElements(array);

            foreach (int num in array)
                Console.Write(num+" ");
        }
    }
}
```

```
1 2 3 4 5

2 3 4 5 6
```

ساختارها (struct)

- سی شارپ دارای نوعی از value type است که ساختار (struct) نامیده می شود. یک ساختار (struct) می تواند با استفاده از کلمه رزرو شده struct تعریف شود.

```
public struct Discounts
{
        public int Cloths { get; set; }
        public int HomeDecor { get; set; }
        public int Grocery { get; set; }
 }
```

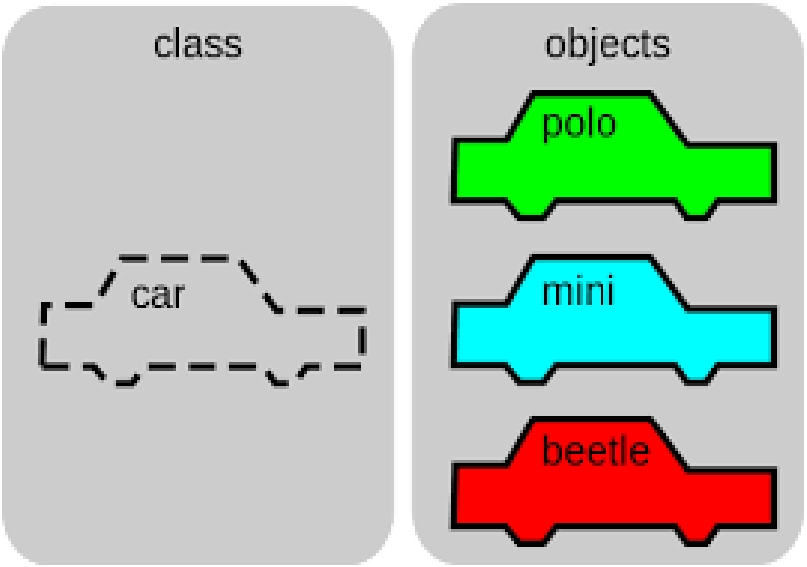
ساختارها (struct)

```
using System;
namespace Class1
{
    class Program
    {
        static void Main(string[] args)
        {
        Discounts saleDiscounts = new Discounts();

        saleDiscounts.Cloths = 10;
        saleDiscounts.HomeDecor = 5;
        saleDiscounts.Grocery = 2;

        Console.WriteLine(saleDiscounts.Cloths);
        Console.WriteLine(saleDiscounts.HomeDecor);
        Console.WriteLine(saleDiscounts.Grocery);

        Console.ReadKey();
        }
    }
```

```
public struct Discounts
{
    public int Cloths { get; set; }
    public int HomeDecor { get; set; }
    public int Grocery { get; set; }
}
```

کلاس

- یک کلاس مانند نقشه ای کامل از یک شی مشخص است. در جهان واقعی هر شی ایی دارای یک سری خصوصیات مانند رنگ، شکل و نوع عملکرد است.
- در دنیای برنامه نویسی شی گرا ، یک کلاس دارای تعدادی مشخص فیلد ، صفت ، رویداد و متد است. یک کلاس انواع داده و عملکردهایی که اشیا دارند را مشخص می کند. در یک کلاس میتوانید نوع مورد نظر خود را از طریق گروه بندی متغیر ها و دیگر انواع ایجاد کنید.



- **تعریف شی (Object):**

به یک نمونه از کلاس شی گفته می شود.

تعریف کلاس (class)

- در زبان برنامه نویسی C# یک کلاس میتواند با استفاده از کلمه ی رزرو شده ی class تعریف شود :

```
public class ClassName
{
    //Fields, properties, methods and events go here...
}
```

تعریف شی از کلاس:

```
ClassName objectname = new ClassName();
```

اجزای یک کلاس

```
public class MyClass
{
    public string myField = string.Empty;
    public MyClass()
    {
    }

    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}", parameter1, parameter2);
    }

    public int MyAutoImplementedProperty { get; set; }

    private int myPropertyVar;

    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}
```

Access Modifier

Class name

field

Constructor

Method\Function

Auto-implemented property

Property

سطوح دسترسی (Access Modifiers)

سطوح دسترسی کلمات رزرو شده ای هستند که بر روی اعلان یک کلاس ، متد ، صفت ، فیلد و دیگر اعضای یک کلاس میتوانند اعمال شوند و از این طریق نحوه دسترسی به اعضای کلاس را معین می سازند.

کلمات رزرو شده برای سطوح دسترسی در زبان سی شارپ عبارت اند از :

- `public`
- `private`
- `protected`
- `internal`

این کلمات، چگونگی و سطح دسترسی یک کلاس و یا اعضای آن را در برنامه مشخص میکنند.

فیلد (Field)

- متغیری که در سطح یک کلاس تعریف میشود فیلد نامیده میشود. فیلدها میتوانند مقادیری از یک نوع مشخص را در خود نگه دارند.
- عموماً فیلد ها در کلاس دارای سطح دسترسی private (یعنی فقط قابل دسترسی در محدوده ی همان کلاس) هستند و در صفت ها (Property) استفاده میشوند.

```
private int _myPropertyVar;
```

صفت (Property)

- یک صفت میتواند با استفاده از کلمات رزرو شده ی get و set مانند نمونه کد زیر ایجاد شود.

```
private int _myPropertyVar;

public int MyProperty
{
    get { return _myPropertyVar; }
    set { _myPropertyVar = value;}
}
```

- عموما صفات در زبان C # دارای سطح دسترسی public (قابل دسترسی در خارج از محدوده ی کلاس) هستند. به عبارت دیگر فیلد _myPropertyVar در خارج از کلاس به طور غیر مستقیم از طریق صفت MyProperty قابل دسترسی است.

صفت (Property)

نکته :

الزامی برای وجود هر دو کلمه ی رزرو شده ی get و setدر تعریف یک صفت وجود ندارد. برای مثال اگر صفتی فقط داری قسمت get باشد آن صفت فقط خواندنی است. حتی میتوان منطقی خاص را در قسمت های get و set برای یک صفت به کار برد.

```
public int MyProperty
{
    get
    {
        return _myPropertyVar / 2;
    }

    set
    {
        if (value > 100)
            _myPropertyVar = 100;
        else
            _myPropertyVar = value;
    }
}
```

سازنده (Constructor):

یک کلاس میتواند دارای سازنده های پارامتر دار و یا بدون پارامتر باشد. سازنده ها در هنگام تعریف یک شی از یک کلاس فراخوانی میشوند. سازنده ها به وسیله ی یک کلمه ی سطح دسترسی و کلمه ای که همنام با نام کلاس باشند تعریف میشوند.

متد سازنده یک کلاس متدی است که در هنگام تعریف یک شی از کلاس به طور خودکار اجرا می گردد.

```
class MyClass
{
    public MyClass()
    {
        // Constructor code
    }
}
```

متد (Method)

- یک متد در زبان برنامه نویسی سی شارپ میتواند به شکل الگوی زیر تعریف شود :

```
{access modifier} {return type} MethodName({ parameterType parameterName})
```

```
public void MyMethod(int parameter1, string parameter2)
{
        // write your method code here..
}
```

صفات Auto-implemented

این صفات برای زمانی است که نیاز به اعمال منطق خاصی در صفت خود نداریم.

- نمونه مثال زیر یک صفت Auto-implemented را نشان میدهد :

```
public int MyAutoImplementedProperty { get; set; }
```

- دقت داشته باشید که هیچ فیلدی برای این صفت تعریف نشده است و بعداً در زمان اجرای یک فیلد به صورت ضمنی توسط کامپایلر ایجاد شده و این نوع صفات را مدیریت میکند.
- با صفات Auto-implemented میتوان همانند یک صفت معمولی رفتار کرد و تا زمانی که منطقی نباشد تفاوتی با دیگر صفات ندارد.