زنگ سی‌شارپ – قسمت بیستم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت نوزدهم با آرایه‌های چند بعدی آشنا شدیم. در این قسمت سعی داریم با مفاهیم برنامه‌نویسی شی‌گرا (OOP) آشنا شویم.

برنامه‌های کامپیوتری روز به روز در حال پیچیده‌تر شدن هستند. برنامه‌ی ساده‌ی Hello World شما دیگر به سادگی قبل نیست و امروزه خط کد برنامه‌های‌تان آنقدر زیاد و پیچیده شده‌اند که خودتان هم در آن سردرگم می‌مانید. می‌خواهید قسمت‌هایی از کدتان را مجدداً مورد استفاده قرار دهید و به کدهای قدیمی‌تان ویژگی‌های جدیدی را (بدون دست‌کاری کدهای دیگر) اضافه کنید. می‌خواهید از زیاده‌نویسی بپرهیزید. دوست دارید برنامه‌های‌تان خیلی بیشتر از قبل قابل درک باشند و به دنیای واقعی نزدیک‌تر شوند. برنامه‌نویسی شی‌گرا آمده است تا مشکل شما را حل و کار شما را راحت کند طوری‌که واقعاً از برنامه‌نویسی لذت ببرید.

دو روش عمومی برای نوشتن برنامه‌های کامپیوتری وجود دارد که یکی Procedural Programming و دیگری Object-Oriented Programming (OOP) است. قبل از OOP استفاده از Procedural Programming مرسوم بود بدین صورت که برنامه به تعداد زیادی Procedure تقسیم بندی می‌شد. زبان‌هایی مثل C، Pascal، Cobol و ... Procedural style هستند. با آمدن OOP روش Procedural Programming رفته رفته منسوخ شد و امروزه کمتر کسی Procedural Programming را به OOP ترجیح می‌دهد.

برنامه نویسی شی گرا (Object-Oriented Programming)

همیشه افرادی که به‌صورت Procedural کار می‌کنند و همچنین افراد تازه‌کار از من درباره‌ی این‌که برنامه‌نویسی شی‌گرا چیست می‌پرسند و من همیشه سعی کرده‌ام با مثال‌هایی از دنیای واقعی، موضوع را برای آنان روشن سازم. به‌عنوان مثال، از آن‌ها درباره‌ی این‌که به چه چیزی وسیله‌ی نقلیه گفته می‌شود، می‌پرسم. پاسخ آن‌ها این است: "خوب، چیزی مثل ماشین، کامیون، وانت و..." سپس می‌پرسم "به نظر شما چه تفاوتی بین وانت و کامیون دیده می‌شود؟!" و آن‌ها نیز به مواردی چون اندازه، ظرفیت، هدف ساخت و غیره اشاره می‌کنند. سپس به آن‌ها می‌گویم که فعلاً فکر کردن درباره‌ی وانت یا کامیون را کنار بگذارید و بیشتر درباره‌ی چیزهایی که یک وسیله‌ی نقلیه دارد صحبت کنید و آن‌ها در پاسخ

می‌گویند: "خوب، یک وسیله‌ی نقلیه راننده، چرخ، پدال ترمز و گاز، فرمان، اتاق و ... دارد" و من در این لحظه به آن‌ها خاطرنشان می‌کنم که همه‌ی این‌ها خصوصیاتی هستند که یک وسیله‌ی نقلیه دارد سپس از آن‌ها می‌پرسم آیا این وسیله‌ی نقلیه با این خصوصیات کاری را هم انجام می‌دهد؟ آن‌ها پاسخ می‌دهند: "بله، مسلماً! اگر پدال گاز را فشار دهیم وسیله‌ی نقلیه توسط چرخ‌هایش شروع به حرکت می‌کند و هنگامی‌که فرمان را بچرخانیم مسیر حرکت آن عوض می‌شود." درست است! این وسیله‌ی نقلیه یک‌سری ویژگی و یک‌سری رفتار خاص دارد. بنابراین با این تعاریف (ویژگی‌ها و رفتار وسیله‌ی نقلیه) یک شرکت می‌تواند یک وسیله‌ی نقلیه مثل وانت، کامیون و ... را تولید کند.

برنامه‌نویسی شی‌گرا نیز این‌چنین است. شما در برنامه‌نویسی شی‌گرا می‌توانید دنیای اطراف خود را مدل کنید. شما می‌توانید در برنامه‌تان یک خودرو را تعریف کنید و مشخص کنید که این خودرو چه ویژگی‌ها و چه رفتارهایی داشته باشد (Class) و سپس یک خودرو بسازید (Object).

مفاهیم برنامه نویسی شی گرا

یک زبان برنامه‌نویسی در صورتی شی‌گرا است که شامل مفاهیم زیر باشد:

- CLASS
- OBJECT
- ABSTRACTION
- ENCAPSULATION
- INTERFACE
- DATA HIDING / INFORMATION HIDING
- INHERITANCE
- POLYMORPHISM

هر یک از مفاهیم بالا در این مقاله و مقالات آینده مورد بررسی قرار می‌گیرد.

CLASS چیست؟

Class قالب یک Object را مشخص می‌کند. در واقع Class مشخص کننده‌ی data و code ای است که روی data عمل می‌کند. سی‌شارپ از مشخصات یک Class برای ساخت Object استفاده می‌کند بنابراین Object نمونه‌ای از یک Class است. ما می‌توانیم به تعداد دلخواه از یک کلاس Object بسازیم.

OBJECT چیست؟

یک Object بیان کننده‌ی چیزی است که در دنیای واقعی قابل درک باشد. Object یک سری فعالیت‌های مرتبط به هم را انجام می‌دهد و هر Object ویژگی و رفتارهای خاص خود را دارد. همان‌طور که گفته شد، Object نمونه‌ای از یک Class است. به‌عنوان مثال "دست" را در نظر بگیرید. دست یک Class است. در Class دست، مشخص شده که یک دست باید چه ویژگی‌ها و چه رفتارهایی داشته باشد (۵ انگشت، بازو، ساعد، گرفتن، ضربه زدن و ...) و بدن ما دو Object از این Class را دارد که دست چپ و دست راست نامیده می‌شوند.

وقتی یک کلاس را تعریف می‌کنید، هم data و هم code ای که روی این دیتا عمل می‌کند را تعریف می‌کنید. در حالت کلی مواردی که در یک کلاس قرار می‌دهید در دو دسته‌ی Data Members و Function Members تقسیم‌بندی می‌شوند. در سی‌شارپ این دو قسمت به قسمت‌های مختلف دیگری نیز تقسیم بندی می‌شوند. به‌عنوان مثال data members (که به آن‌ها fields هم گفته می‌شود) شامل instance variable و static variable هستند. همین‌طور function members شامل method، constructor، destructor، indexer، event، operators و properties هستند.

یک کلاس با کلمه‌کلیدی class ساخته می‌شود. فرم کلی یک کلاس که شامل instance variable و method است، به‌شکل زیر است:

```
class className
{
    // declare instance variables
    access type var-name1;
    access type var-name2;

    // declare methods
    access ret-type methodName1(parameters)
    {
        // body of method
    }

    access ret-type methodName2(parameters)
    {
        // body of method
    }
}
```

در این‌جا، access نوع دسترسی به متغیر و متد مربوطه را مشخص می‌کند. مشخص کردن نوع دسترسی اختیاری است و اگر نوع دسترسی را مشخص نکنید، نوع دسترسی به‌طور پیش‌فرض private در نظر گرفته می‌شود. اعضایی از کلاس که

دسترسی به آن‌ها private در نظر گرفته می‌شود فقط توسط همان کلاس و اعضای همان کلاس قابل دسترسی هستند. اگر دسترسی اعضای کلاس به‌صورت public در نظر گرفته شود آن‌گاه آن اعضا در همه‌جای برنامه قابل دسترسی هستند.

کلاس‌هایی که می‌سازید باید هدف خاصی از ساخت آن‌ها داشته باشید و از قرار دادن هرگونه اطلاعات نامربوط در آن پرهیز کنید. تا پیش از این، کلاسی که از آن در برنامه‌هایمان استفاده می‌کردیم تنها یک method داشت که آن‌هم متد Main() بود. متد Main() تنها باید در کلاسی قرار داشته باشد که آن کلاس نقطه شروع برنامه‌تان است.

در مثال زیر یک کلاس به اسم Car می‌سازیم که شامل سه متغیر است:

```
class Car
{
    public string Color;
    public string Model;
    public int MaxSpeed;
}
```

در این کلاس نوع دسترسی هر سه متغیر public است. به این معنی که این سه متغیر می‌توانند در همه‌جای برنامه قابل دسترسی باشند. در این کلاس هیچ method ای تعریف نکرده‌ایم و این یک کلاس data-only است. هنگامی‌که یک کلاس را تعریف می‌کنید در واقع یک data type جدید به‌وجود می‌آورید که در این مثال اسم این دیتاتایپ، Car است. شما از این اسم برای ساخت اشیائی از جنس Car استفاده می‌کنید.

با تعریف یک کلاس شما فقط قالب یک شیء را مشخص می‌کنید و در واقع هنوز هیچ object ای از آن نساخته‌اید. در مثال قبل شما فقط مشخص کردید که یک ماشین می‌تواند سه مشخصه‌ی رنگ، ماکسیموم سرعت و مدل را داشته باشد اما شما هنوز هیچ ماشینی را نساخته‌اید و فقط مشخص کرده‌اید که اگر قرار است ماشینی ساخته شود باید شامل این سه مشخصه باشد!

برای این‌که از کلاس Car، یک شیء بسازید، می‌نویسید:

```
Car BMW = new Car(); // create an object of type Car
```

بعد از اجرای این خط، یک شیء از کلاس Car ساخته می‌شود و کلاس Car اکنون یک واقعیت فیزیکی دارد چراکه شما با توجه به مشخصاتی که برای یک ماشین تعریف کردید، یک BMW را ساختید. در حال حاضر نگران جزئیات این خط کد نباشید.

از هر کلاس می‌توانید به تعداد دل‌خواه شیء بسازید. هربار که از یک کلاس شیء می‌سازید هر شیء، کپی خودش را از متغیرهای کلاس مربوطه دارد. بنابراین هر شیء Car، کپی خودش را از متغیرهای Color، Model و MaxSpeed دارد. برای دسترسی به این متغیرها از عمل‌گر دسترسی به اعضا استفاده می‌کنید که این عملگر، یک نقطه (dot operator) است. dot operator اسم یک شیء را به اسم یکی از اعضای همان شیء متصل می‌کند:

```
object.member
```

برای مثال به این طریق می‌توانید رنگ BMW تان را تنظیم کنید:

```
BMW.Color = "Blue";
```

به مثال زیر توجه کنید:

```
// A program that uses the Car class.
using System;
class Car
{
    public string Color;
    public string Model;
    public int MaxSpeed;
}

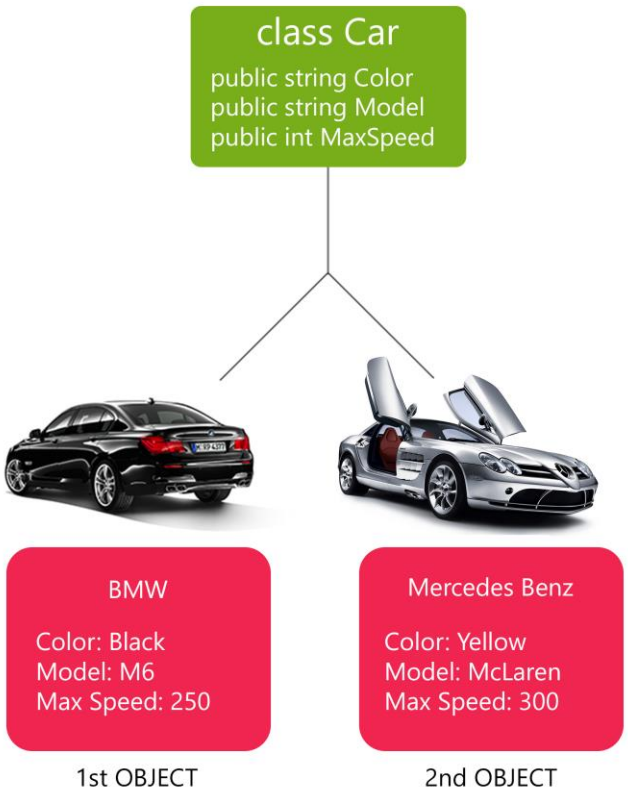
// This class declares two objects of type Car.
class Example
{
    static void Main()
    {
        // First Object
        Car mercedesBenz = new Car();
        mercedesBenz.Color = "Yellow";
        mercedesBenz.Model = "McLaren";
        mercedesBenz.MaxSpeed = 300;

        Console.WriteLine("Mercedes Benz has:");
        Console.WriteLine("  " + mercedesBenz.Color + " Color");
        Console.WriteLine("  " + mercedesBenz.Model + " Model");
        Console.WriteLine("  " + mercedesBenz.MaxSpeed + " Max Speed");

        Console.WriteLine();
        // Second Object
        Car BMW = new Car();
        BMW.Color = "Black";
        BMW.Model = "M6";
        BMW.MaxSpeed = 250;

        Console.WriteLine("BMW has:");
        Console.WriteLine("  " + BMW.Color + " Color");
        Console.WriteLine("  " + BMW.Model + " Model");
        Console.WriteLine("  " + BMW.MaxSpeed + " Max Speed");
    }
}
```

در این مثال از کلاس Car دو شیء مختلف را ساختیم که هرکدام رنگ، مدل و ماکسیموم سرعت مخصوص به خود را دارند. در این برنامه دو کلاس Example و Car موجود است. این‌ها دو کلاس جدا هستند و تنها ارتباط آن‌ها این است که در کلاس Example دو شیء از کلاس Car ساخته می‌شود. ما در کلاس Example میتوانیم به اعضای کلاس Car دسترسی داشته باشیم، چراکه دسترسی آن‌ها public است اما اگر دسترسی آن‌ها private باشد فقط در کلاس Car قابل دسترسی هستند.



چگونه یک Object ساخته می‌شود؟

در مثال قبل از خط کد زیر برای ساخت یک شیء استفاده کردیم:

```
Car BMW = new Car();
```

این خط کد، سه کار را انجام می‌دهد. ابتدا، یک متغیر به اسم BMW از جنس Car می‌سازد. این متغیر به خودی خود، یک شیء نیست بلکه متغیری است که به یک شیء رجوع می‌کند (آدرس یک شیء را در خود ذخیره می‌کند). این خط کد در مرحله‌ی دوم توسط کلمه‌ی کلیدی new یک شیء را می‌سازد و سپس در مرحله‌ی سوم توسط علامت مساوی (=)

متغیر و شیء ساخته شده را به هم وصل می‌کند (به متغیر BMW آدرس جایی که object ساخته شده قرار دارد اختصاص داده می‌شود). بنابراین بعد از اجرای این خط کد، متغیر BMW به یک شیء از جنس Car رجوع می‌کند.

خط کد بالا را به طریق زیر هم می‌توانید بنویسید:

```
Car BMW;
BMW = new Car();
```

در خط اول متغیری از جنس Car تعریف کرده‌اید که می‌تواند آدرس یک شیء را در خود ذخیره کند (می‌تواند به یک شیء رجوع کند). در خط دوم، یک شیء از جنس Car ساخته می‌شود و آدرس آن در متغیر BMW ذخیره می‌شود بنابراین بعد از اجرای این دستورات متغیر BMW و شیء ساخته شده به هم وصل هستند.

مطمئناً تا این‌جا متوجه شده‌اید که متغیر BMW شیء را در خودش ذخیره نمی‌کند بلکه شیء در حافظه ذخیره می‌شود و متغیر فقط آدرس جایی که شیء قرار دارد را در خود ذخیره می‌کند. اگر به‌یاد داشته باشید بیان شد که سی‌شارپ شامل دو دسته‌بندی برای data type (نوع اطلاعات) است که یکی value type و دیگری reference type بود. تفاوت بین این‌دو مقداری است که متغیر در خودش نگاه می‌دارد. برای یک متغیر value type، متغیر مستقیماً خود مقدار را نگهداری می‌کند:

```
int x = 22;
```

در این‌جا متغیر x از جنس int و مستقیماً شامل مقدار ۲۲ است. به همین دلیل int را value type می‌نامیم.

اما در این مورد:

```
Car BMW = new Car();
```

متغیر BMW خود شیء را نگهداری نمی‌کند بلکه تنها آدرس آن را ذخیره دارد به همین دلیل کلاس‌ها reference type هستند.

Method چیست؟

Method ها شبیه به Procedure ها، Function ها و ... در دیگر زبان‌های برنامه‌نویسی هستند. Method ها و متغیرها (instance variables) تشکیل دهنده‌ی اجزای اصلی کلاس هستند. کلاسی که در مثال‌های قبلی ساخته بودیم فقط شامل متغیر بود (data-only). اکنون قصد داریم کلاسی بسازیم که شامل Method هم باشد. اصولاً Method ها، متغیرهایی

که در کلاس تعریف می‌شوند را تغییر می‌دهند و در بیشتر موارد باعث می‌شوند بتوانید به متغیرهای کلاس دسترسی داشته باشید. معمولاً قسمت‌های دیگر برنامه از طریق Method های یک کلاس با آن در تعامل هستند.

هر Method باید یک وظیفه را بر عهده داشته باشد، نه چند وظیفه. هر Method یک اسم دارد و شما از طریق همین اسم می‌توانید Method را صدا بزنید و آن را اجرا کنید. برای نام‌گذاری Method ها نباید از اسم کلمه‌های کلیدی سی‌شارپ و Main() که از پیش رزرو شده است استفاده کنید. بعد از نام Method پرانتز باز و بسته قرار می‌گیرد. به‌عنوان مثال اگر نام یک Method را CalculateSum انتخاب کرده باشد باید بنویسید CalculateSum() و این کار باعث می‌شود نام Method را از نام متغیرها تشخیص دهید.

فرم کلی یک Method به‌شکل زیر است:

```
access ret-type name(parameter-list)
{
    // body of method
}
```

در این‌جا access مشخص کننده‌ی نوع دسترسی است (access modifier) و دسترسی قسمت‌های دیگر برنامه را به این Method کنترل می‌کند. همان‌طور که پیش‌تر بیان شد، قرار دادن access modifier اختیاری است و درصورتی که آن را ننویسید، private محسوب می‌شود. ret-type مشخص کننده‌ی نوع مقداری است که Method برمیگرداند. اگر Method هیچ مقداری را برنگرداند ret-type باید void باشد. name مشخص کننده‌ی اسم Method است و در قسمت parameter-list لیستی از متغیرها قرار می‌گیرید که Method می‌تواند آن‌ها را دریافت کند. اگر Method هیچ پارامتری نداشته باشد این قسمت باید خالی گذاشته شود.

به مثال زیر توجه کنید:

```
using System;

class HelloWorld
{
    public void SayHello(string name, string family)
    {
        Console.WriteLine("Hello " + name + " " + family);
    }
}

class Example
{
    static void Main()
    {
        HelloWorld myWorld = new HelloWorld();
        myWorld.SayHello("Masoud", "Darvishian");
```

```
    }
}
```

در این مثال، در کلاس HelloWorld متدی را به اسم ()SayHello قرار دادیم که هیچ مقداری را برنمی‌گرداند (void) و تنها دو مقدار را به‌عنوان ورودی دریافت و پیغام خوش‌آمدگویی را چاپ می‌کند. در متد ()Main برای این‌که به متد ()SayHello دسترسی داشته باشیم ابتدا باید یک شیء از کلاس HelloWorld بسازیم سپس می‌توانیم از طریق نام آن شیء به متد دسترسی پیدا کنیم (زیرا دسترسی public است).

Return کردن از یک Method

در دو حالت یک متد return می‌شود:

- زمانی که برنامه به کروشه پایانی (}) متد برسد.
- زمانی که با کلمه return مواجه شود.

برای استفاده از کلمه‌ی return در متد، دو حالت وجود دارد. حالت اول زمانی است که از void methods استفاده می‌کنید (متدهایی که هیچ مقداری را برنمی‌گردانند) و حالت دوم برای متدهایی است که مقداری را برمی‌گردانند.

استفاده از return در یک void method موجب می‌شود اجرای متد در همان‌جایی که هست متوقف شود.

به مثال زیر دقت کنید:

```
public void MyMeth()
{
    int i;
    for (i = 0; i < 10; i++)
    {
        if (i == 5) return; // stop at 5
        Console.WriteLine();
    }
}
```

در این Method زمانی‌که i برابر با ۵ باشد، اجرای مابقی خط کدهای متد متوقف شده و داستان از همان نقطه‌ای که متد صدا زده شده است ادامه می‌یابد. همچنین می‌توانید در void methods از چندین return در جاهای مختلف آن استفاده کنید اما ترجیحاً این کار انجام ندهید چراکه معمولاً این‌جور استفاده در نقاط مختلف یک Method موجب خرابی کد می‌شود.

البته return کردن به این شکل از یک void method رایج نیست و بیشتر Method ها یک مقدار را برمیگردانند. return کردن یک مقدار به منظورهای مختلفی در برنامه‌نویسی استفاده می‌شود. در بعضی موارد مقداری که return می‌شود نتیجه‌ی یک‌سری محاسبات است. در بعضی موارد دیگر مقدار بازگشتی نشان‌دهنده‌ی موفقیت یا عدم موفقیت یک عملیات خاص است. درکل، مقدار بازگشتی بستگی به هدف شما از نوشتن آن Method دارد.

به Method زیر توجه کنید:

```
public double Addition(double x, double y)
{
    double result = x + y;
    return result;
}
```

این Method دو مقدار را دریافت، آن‌ها را جمع و در متغیر result ذخیره می‌کند و در نهایت مقدار result را return می‌کند (مقدار result را برمی‌گرداند). فرض کنید ما مقدار ۲ و ۵ را به این Method بدهیم. برای این‌که از این Method و مقداری که return می‌کند استفاده کنیم می‌نویسیم:

```
using System;
class Calculator
{
    public double Addition(double x, double y)
    {
        return x + y;
    }
}
class Example
{
    static void Main()
    {
        double result;
        Calculator myCalculate = new Calculator();

        result = myCalculate.Addition(5, 2);
        Console.WriteLine("Addition: " + result);
    }
}
```

همان‌طور که می‌بینید از کلاسی که Method در آن قرار داشت یک شیء ساختیم و سپس متد را صدا زدیم و مقادیر ۲ و ۵ را به آن دادیم. این متد، ۲ و ۵ را جمع و حاصل این جمع (عدد ۷) را return می‌کند. سپس عدد ۷ در متغیر result ذخیره می‌شود.

به مثال زیر توجه کنید:

```
using System;

class Calculator
{
    public double Multipliction(double x, double y)
    {
        return x * y;
    }

    public double Division(double x, double y)
    {
        return x / y;
    }

    public double Subtraction(double x, double y)
    {
        return x - y;
    }

    public double Addition(double x, double y)
    {
        return  x + y;
    }
}

class Example
{
    static void Main()
    {
        double result;
        Calculator myCalculate = new Calculator();

        Console.Write("Enter first number: ");
        double firstNum = Convert.ToDouble(Console.ReadLine());
        Console.Write("Enter second number: ");
        double seconNum = Convert.ToDouble(Console.ReadLine());

        Console.WriteLine();
        Console.WriteLine("Calculations");
        result = myCalculate.Addition(firstNum, seconNum);
        Console.WriteLine("  Addition: " + result);

        result = myCalculate.Subtraction(firstNum, seconNum);
        Console.WriteLine("  Subtraction: " + result);

        result = myCalculate.Multipliction(firstNum, seconNum);
        Console.WriteLine("  Multipliction: " + result);

        result = myCalculate.Division(firstNum, seconNum);
        Console.WriteLine("  Division: " + result);

        result = Math.Pow(firstNum, seconNum);
        Console.WriteLine("  Power: " + result);
    }
}
```

در واقع این همان مثال قبلی است با این تفاوت که اندکی آن را توسعه دادیم. در این مثال، کلاسی به اسم Calculator داریم که در آن متدهایی قرار دارد که عملیات ضرب، تقسیم، جمع و تفریق را انجام می‌دهند و نتیجه را باز می‌گردانند. در این برنامه مقادیری که باید به این متدها داده شود را از کاربر دریافت کرده‌ایم و در نهایت نتیجه‌ی محاسبات را نمایش داده‌ایم. همچنین از متد Pow() که در کلاس Math قرار دارد (یکی از متدهای از پیش تعریف شده در دات‌نت فریم‌ورک است) برای به توان رساندن استفاده کرده‌ایم. توجه کنید که اگر ret-type یک متد را از جنس double تعریف کرده‌اید آن‌گاه مقداری را که return می‌کنید نیز باید double باشد.

مطمئناً تا این لحظه متوجه شده‌اید هنگامی‌که می‌نویسیم Console.WriteLine() درحال استفاده کردن از متد WriteLine() هستیم که این متد در کلاس Console قرار دارد. همین‌طور موارد مشابهی چون Console.ReadLine() و غیره.

استفاده از پارامترها

در مثال‌های قبل‌تر چگونگی استفاده از پارامتر را مشاهده کردید. هنگامی که یک Method را صدا می‌زنید می‌توانید یک یا چندین مقدار را به آن بدهید. مقداری که به یک Method داده می‌شود argument نام دارد. درون Method، متغیری که این argument را دریافت می‌کند پارامتر (parameter) نامیده می‌شود. پارامترها درون پرانتزی که جلوی نام متد قرار می‌گیرد تعریف می‌شوند و طریقه‌ی تعریف کردن پارامتر به همان شکل است که به‌صورت عادی یک متغیر تعریف می‌شود. این پارامترها فقط در محدوده‌ی خود آن متد شناخته می‌شوند.

به مثال زیر توجه کنید:

```
using System;
class ChkNum
{
    public bool isEven(int number)
    {
        if (number % 2 == 0)
            return true;
        else
            return false;
    }
}

class Example
{
    static void Main()
    {
```

```
        ChkNum ob = new ChkNum();
        for (int i = 0; i < 10; i++)
        {
            if (ob.isEven(i))
                Console.WriteLine(i + " is even");
            else
                Console.WriteLine(i + " is odd");
        }
    }
}
```

در این برنامه، ۱۰ مرتبه متد isEven() صدا زده می‌شود و هربار مقداری متفاوت به‌عنوان argument به آن داده می‌شود و درون متد، پارامتر number هربار این argument را دریافت می‌کند. متد isEven() عدد دریافتی را بررسی نموده و درصورتی که زوج باشد مقدار true را بازمی‌گرداند.

کلیه حقوق مادی و معنوی برای وب‌سایت وب‌تارگت محفوظ است.
استفاده از این مطلب در سایر وب‌سایت‌ها و نشریات چاپی تنها با ذکر و درج لینک منبع مجاز است.