زنگ سی‌شارپ – قسمت بیست و یکم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبل به معرفی برنامه‌نویسی شی‌گرا، class ها، object ها و method ها پرداختیم. در این قسمت به ادامه‌ی مباحث قسمت قبل می‌پردازیم و همچنین با constructor و کلمات‌کلیدی this و static آشنا می‌شویم.

Constructor و چگونگی استفاده از آن

در قسمت قبل مشاهده کردید هنگامی‌که از یک کلاس، شیء می‌سازید به تک تک متغیرهای آن جداگانه مقدار می‌دهید:

```
BMW.Color = "Black";
BMW.Model = "M6";
BMW.MaxSpeed = 250;
```

اگر قصد دارید به‌صورت حرفه‌ای برنامه‌نویسی سی‌شارپ را دنبال کنید مسلماً این روش مقداردهی برای شما مناسب نیست. استفاده از این روش ممکن است باعث به‌وجود آمدن خطا در برنامه شود (شاید مقداردهی به یک فیلد فراموش شود). راه بهتر برای انجام این‌کار استفاده از constructor است. constructor یک شیء را همزمان با ساخت آن، مقداردهی می‌کند. نام constructor باید برابر با نام همان کلاسی باشد که constructor در آن قرار دارد. constructor از لحاظ syntax مشابه به method است با این تفاوت که هیچ مقداری را برنمی‌گرداند و بدون ret-type نوشته می‌شود.

فرم کلی constructor به شکل زیر است:

```
access class-name(param-list)
{
    // constructor code
}
```

اصولاً از constructor برای مقداردهی اولیه به متغیرهای کلاس یا اجرای یک‌سری عملیات، زمانی که شیء ساخته می‌شود، استفاده می‌کنید. نوع دسترسی (access modifier) به‌صورت public در نظر گرفته می‌شود زیرا constructor معمولاً خارج از کلاس خودش فراخوانی می‌شود. Param-list می‌تواند خالی باشد یا این‌که یک یا چندین پارامتر داشته باشد.

چه شما constructor تعریف کرده باشید چه نکرده باشید، همه‌ی کلاس‌ها یک constructor دارند. زیرا سی‌شارپ به‌صورت اتوماتیک یک default constructor برای هر کلاس به‌وجود می‌آورد که به همه‌ی متغیرهای کلاس یک مقدار پیش‌فرض را اختصاص می‌دهد. برای بیشتر value type ها مقدار پیش‌فرض صفر، برای بولین مقدار پیش‌فرض false و برای reference type ها مقدار پیش‌فرض null در نظر گرفته می‌شود. زمانی که شما از constructor خودتان استفاده می‌کنید دیگر از default constructor استفاده نمی‌شود.

در این مثال ساده، استفاده از constructor را مشاهده می‌کنید:

```
using System;
class Car
{
    public string Color;

    public Car()
    {
        Color = "White";
    }
}

class Example
{
    static void Main()
    {
        Car BMW = new Car();
        Car mercedesBenz = new Car();

        Console.WriteLine(BMW.Color);
        Console.WriteLine(mercedesBenz.Color);
    }
}
```

در این مثال، constructor برابر است با:

```
public Car()
{
    Color = "White";
}
```

همان‌طور که می‌بینید دسترسی به‌صورت public تعریف شده است چراکه constructor خارج از کلاس خودش صدا زده می‌شود. این constructor توسط کلمه‌کلیدی new زمانی که شیء ساخته شد فراخوانی می‌شود و رشته‌ی "White" را به متغیر Color اختصاص می‌دهد.

برای مثال در این خط کد:

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

Constructor این کلاس که Car() نام دارد توسط شیء BMW صدا زده می‌شود و به BMW.Color مقدار "White" را اختصاص می‌دهد. همین داستان برای شیء mercedesBenz نیز صدق می‌کند.

در مثال قبل constructor ما هیچ پارامتری نداشت. در بیشتر مواقع شما نیاز دارید که یک یا چندین پارامتر برای constructor تان مشخص کنید. پارامترها از همان روشی که در قسمت قبل برای متدها بیان شد، در این‌جا نیز استفاده می‌شوند.

به مثال زیر توجه کنید:

```
using System;
class Car
{
    public string Color;

    public Car(string carColor)
    {
        Color = carColor;
    }
}

class Example
{
    static void Main()
    {
        Car BMW = new Car("Black");
        Car mercedesBenz = new Car("Yellow");

        Console.WriteLine(BMW.Color);
        Console.WriteLine(mercedesBenz.Color);
    }
}
```

در این مثال، کانستراکتور Car() تنها یک پارامتر به نام carColor دارد که از آن برای مقدار دهی به متغیر Color استفاده می‌شود. بنابراین بعد از اجرای این خط کد:

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

رشته‌ی "Black" به متغیر carColor داده می‌شود و سپس به متغیر Color اختصاص می‌یابد.

به مثال زیر توجه کنید:

```
using System;
class Car
{
    public string Color;
    public string Model;
    public int MaxSpeed;
```

```
    public Car(string carColor, string carModel, int carMaxSpeed)
    {
        Color = carColor;
        Model = carModel;
        MaxSpeed = carMaxSpeed;
    }
}

class Example
{
    static void Main()
    {
        Car BMW = new Car("Black", "M6", 250);
        Car mercedesBenz = new Car("Yellow", "McLaren", 300);

        Console.WriteLine(BMW.Color + "\t" + BMW.Model + "\t" + BMW.MaxSpeed);
        Console.WriteLine(mercedesBenz.Color + "\t" + mercedesBenz.Model + "\t"
            + mercedesBenz.MaxSpeed);
    }
}
```

در این برنامه، از چندین پارامتر برای constructor استفاده کرده‌ایم. در واقع توسط constructor شما برای ساخت اشیاء این اجبار را به‌وجود می‌آورید که همه‌ی field ها حتماً مقداردهی شوند.

کلمه‌کلیدی this

هنگامی‌که یک متد صدا زده می‌شود، متد به‌صورت اتوماتیک به شیء مربوط به خودش رجوع می‌کند. برای این‌که با کلمه‌کلیدی this آشنا شوید به مثال زیر توجه کنید:

```
using System;
class Rectangle
{
    public int Width;
    public int Height;
    public Rectangle(int w, int h)
    {
        Width = w;
        Height = h;
    }
    public int Area()
    {
        return Width * Height;
    }
}
class UseRect
{
    static void Main()
    {
        Rectangle r1 = new Rectangle(4, 5);
        Rectangle r2 = new Rectangle(7, 9);
        Console.WriteLine("Area of r1: " + r1.Area());
        Console.WriteLine("Area of r2: " + r2.Area());
    }
}
```

در این برنامه کلاسی به اسم Rectangle داریم که شامل دو متغیر، یک constructor و یک متد (که مساحت مستطیل را محاسبه می‌کند) است. درون متد، اعضای کلاس می‌توانند مستقیماً (بدون ساخت هیچ شیءای از کلاس) قابل دسترسی باشند. بنابراین درون متد Area() این خط کد:

```
return Width * Height;
```

به این معناست که Width و Height با توجه به شیءای که از کلاس Rectangle ساخته می‌شود مقدار متفاوتی می‌توانند داشته باشند. این مقادیر که برای هر شیء متفاوت است، در هم ضرب می‌شوند و حاصل این ضرب return می‌شود. این خط کد را به روش زیر نیز می‌توانید بنویسید:

```
return this.Width * this.Height;
```

در این‌جا، کلمه‌کلیدی this به همان شیءای رجوع می‌کند که در آن متد Area() صدا زده شده است. بنابراین this.Width با مراجعه به شیء مربوطه، شامل مقداری است که آن شیء برای این متغیر دارد. this.Height نیز مشابه با همین قضیه است. برای مثال اگر Area() از object ای به اسم x صدا زده شده باشد بنابراین (با توجه به خط کد قبلی) this به شیء x رجوع می‌کند.

همچنین می‌توانید از this در constructor استفاده کنید:

```
public Rectangle(int w, int h)
{
    this.Width = w;
    this.Height = h;
}
```

مثالی که پیش‌تر ذکر شد را با کلمه‌کلیدی this ببینید:

```
using System;
class Rectangle
{
    public int Width;
    public int Height;
    public Rectangle(int w, int h)
    {
        this.Width = w;
        this.Height = h;
    }
    public int Area()
    {
        return this.Width * this.Height;
    }
}
class UseRect
{
    static void Main()
```

```
    {
        Rectangle r1 = new Rectangle(4, 5);
        Rectangle r2 = new Rectangle(7, 9);
        Console.WriteLine("Area of r1: " + r1.Area());
        Console.WriteLine("Area of r2: " + r2.Area());
    }
}
```

در حقیقت، هیچ برنامه‌نویسی به این طریق که در برنامه بالا از this استفاده کردیم، استفاده نمی‌کند. زیرا در برنامه بالا استفاده از آن نه تنها سودی ندارد، بلکه فرم استاندارد آن بدون this راحت‌تر و ساده‌تر است. اما این‌طور نیست که this بی‌فایده باشد. برای مثال، سی‌شارپ اجازه می‌دهد نام local variables (متغیرهای محلی)، پارامترها و instance variables یکی باشند. وقتی چنین اتفاقی می‌افتد، پارامترها یا متغیرهای محلی، باعث می‌شوند نتوانید نام instance variables را ببینید. در این موارد شما برای دسترسی به instance variables باید از کلمه‌کلیدی this استفاده کنید. برای مثال، اگر پارامترهای constructor را به‌شکل زیر بنویسیم:

```
public Rectangle(int Width, int Height)
{
    this.Width = Width;
    this.Height = Height;
}
```

در این‌جا، نام پارامترها با نام instance variables یکی است و اجازه نمی‌دهد به instance variables دسترسی داشته باشید. بنابراین در این‌جا با استفاده از this می‌توانید به instance variables دسترسی یابید و مشخص کنید که منظورتان فیلدهای همان شیءای است که با آن در ارتباط هستید.

String ها در سی‌شارپ

یکی از مهم‌ترین data type ها در سی‌شارپ، string است چراکه مشخص کننده‌ی رشته‌ای از کاراکترهاست. در بسیاری از زبان‌های برنامه‌نویسی، یک string آرایه‌ای از کاراکترهاست اما در سی‌شارپ چنین نیست. در سی‌شارپ، string ها object هستند. از این‌رو، string ها در دسته‌ی reference type قرار می‌گیرند. شما از ابتدای مقالات زنگ سی‌شارپ از string استفاده می‌کردید اما از ماهیت آن خبر نداشتید. هنگامی‌که بین دابل کوتیشن، رشته‌ای از کاراکترها را قرار می‌دادید در واقع یک شیء می‌ساختید که از جنس string بود.

ساده‌ترین روش تعریف string به این صورت است:

```
string str = "C# Strings are powerful.";
```

در این‌جا متغیر str یک reference variable است و در واقع آدرس مکانی که شیء ساخته شده از جنس string در آن قرار دارد را، در خود ذخیره می‌کند.

شما همچنین می‌توانید با آرایه‌ای از جنس char یک string بسازید:

```
using System;
class Example
{
    static void Main()
    {
        char[] charArray = {'C','#',' ', 'T', 'i', 'm', 'e'};
        string str = new string(charArray);
        Console.WriteLine(str);
    }
}
```

زمانی‌که شما یک شیء string می‌سازید تقریباً می‌توانید هرجایی‌که وارد کردن دابل کوتیشن مجاز است، از آن استفاده کنید. برای مثال می‌توانید از شیء string به‌عنوان argument در متد ()WriteLine استفاده کنید:

```
using System;
class Example
{
    static void Main()
    {
        char[] charArray = {'C','#',' ', 'T', 'i', 'm', 'e'};
        string str1 = new string(charArray);
        string str2 = "webtarget.ir";

        Console.WriteLine(str1);
        Console.WriteLine(str2);
    }
}
```

آرایه‌ی string ها

مانند دیگر دیتا تایپ‌ها، string نیز می‌تواند در یک آرایه ذخیره شود:

```
using System;
class Example
{
    static void Main()
    {
        string[] strArray = {"This", "is", "a", "string", "array." };

        Console.WriteLine("Orginal Array: ");
        for (int i = 0; i < strArray.Length; i++)
        {
            Console.Write(strArray[i] + " ");
        }
        Console.WriteLine("\n");

        // change a string.
```

```
        strArray[1] = "was";
        strArray[4] = "array, too.";

        Console.WriteLine("Modified Array: ");
        for (int i = 0; i < strArray.Length; i++)
        {
            Console.Write(strArray[i] + " ");
        }
        Console.WriteLine("\n");
    }
}
```

خروجی:

```
Original array:
This is a string array.

Modified array:
This was a string array, too.
```

به مثال بعدی توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        string str = "This is a string";

        for (int i = 0; i < str.Length; i++)
        {
            Console.Write(str[i] + " ");
        }
        Console.WriteLine();
    }
}
```

string ها read only هستند. به این معنا که، تنها می‌توانید مقدار آن‌را بخوانید و نمی‌توانید مقدار یک string را تغییر دهید. برای مثال، اگر بنویسید string name = "Amin" و در خط بعد بنویسید string name = "Hamed" رشته‌ی "Amin" هنوز در حافظه‌ی کامپیوتر شما موجود است اما متغیر name دیگر به این شیء رجوع نمی‌کند و در واقع آدرس شیء جدیدی را که تعریف کردیم در خود نگه می‌دارد. بنابراین نمی‌توانید محتوای یک string را تغییر دهید و در واقع یک شیء جدید به‌وجود می‌آورید.

کلمه‌ی کلیدی static

گاهی نیاز دارید که اعضای یک کلاس به هیچ شیءای وابسته نباشند. به‌طور معمول، اعضای کلاس، از طریق شیءای که از آن کلاس ساخته می‌شود قابل دسترسی هستند اما شما می‌توانید عضوی از کلاس را طوری تعریف کنید که بدون ساخت هیچ شیءای، مستقیماً (از طریق نام کلاس و عملگر نقطه) به آن دسترسی داشته باشید. برای ساخت چنین عضوی، قبل از تعریف آن عضو از کلمه‌ی کلیدی static استفاده می‌کنید. هنگامی که عضوی از یک کلاس به‌صورت static تعریف می‌شود آن عضو بدون ساخت هیچ object ای از کلاس قابل دسترسی و در واقع مستقل از اشیاء است و به هیچ شیءای از آن کلاس وصل نمی‌شود. شما می‌توانید هم متدها و هم متغیرها را به‌صورت static تعریف کنید و حتماً تا این لحظه‌ی متوجه‌ی این کلمه‌ی کلیدی در متد Main() شده‌اید. از آن‌جا که متد Main() نقطه‌ی شروع برنامه‌تان و یکی از اعضای کلاس است، باید قبل از هرچیز و پیش از ساخت هرگونه شیءای، صدا زده شود. به این دلیل است که متد Main() را به‌صورت static تعریف می‌کنیم تا قبل از اینکه شیءای از کلاس ساخته شود، متد Main() فراخوانی شده تا درون این متد بتوانیم کنترل برنامه را به‌دست بگیریم.

به مثال زیر توجه کنید:

```
// Use static.
using System;
class MyClass
{
    // A static variable.
    public static int Variable = 100;

    // A static method.
    public static int MyMethod()
    {
        return Variable + 20;
    }
}
class StaticDemo
{
    static void Main()
    {
        Console.WriteLine("Initial value of MyClass.Variable is " + MyClass.Variable);

        MyClass.Variable = 8;
        Console.WriteLine("MyClass.Variable is " + MyClass.Variable);

        Console.WriteLine("MyClass.MyMethod(): " + MyClass.MyMethod());
    }
}
```

در این برنامه کلاس MyClass شامل یک متغیر static و یک متد static است بنابراین برای استفاده از این دو، نیازی نیست که از کلاس آن‌ها یک شیء بسازیم. همان‌طور که می‌بینید در متد Main() بدون ساخت هیچ شیءای از کلاس MyClass به‌راحتی توانسته‌ایم به دو عضو static کلاس MyClass دسترسی داشته باشیم. هنگامی‌که در یک کلاس هم عضو عادی وجود داشته باشد و هم عضو static، شما تنها می‌توانید به اعضای static دسترسی داشته باشید.

برای مثال، این‌جور نحوه‌ی استفاده نادرست است:

```
class MyClass
{
    public int Val1 = 10; // instance variable
    public static int Val2 = 20; // static variable

    public static int MyMethod()
    {
        // This is illegal!
        return Val2 / Val1; // won't complie
    }
}
```

متد static تنها می‌تواند به اعضای static دسترسی داشته باشد و نمی‌تواند مستقیماً به اعضای عادی کلاس دسترسی پیدا کند زیرا اعضای عادی یک کلاس حتماً باید به یک شیء وصل شوند تا مقدارشان در آن شیء ذخیره شود اما اعضای static مستقل از اشیاء هستند و می‌توان مستقیماً به آن‌ها دسترسی پیدا کرد. اگر قصد دارید درون یک متد static به اعضای عادی نیز دسترسی داشته باشید باید از طریق یک شیء این‌کار را انجام دهید:

```
class MyClass
{
    public void NonStaticMethod()
    {
        Console.WriteLine("This is a non static method");
    }

    public static void StaticMethod(MyClass ob)
    {
        ob.NonStaticMethod(); // this is ok
    }
}
```

کلاس بالا کاملاً صحیح است. ما از طریق یک متد static به یک عضو غیر static دسترسی پیدا کردیم اما این کار را از طریق یک شیء پاس داده شده به متد انجام دادیم. هنگامی‌که در متد Main()، متد StaticMethod() را صدا بزنیم باید یک شیء را به آن به عنوان argument به آن بدهیم. درون متد StaticMethod() پارامتر ob این argument را (که یک شیء است) دریافت می‌کند و می‌تواند از طریق این شیء به اعضای غیر static کلاس آن دسترسی داشته باشد.

به مثال زیر توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        PrintHello();
        SayHello("World!");
        string str = MakeHello("Here i am...");
        Console.WriteLine(str);

        // Console.WriteLine(MakeHello("Here i am..."));
    }

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

    static void SayHello(string name)
    {
        Console.WriteLine("Hello " + name);
    }

    static string MakeHello(string name)
    {
        return "Hello " + name;
    }
}
```

در این برنامه، در کلاس Example، علاوه بر متد Main()، سه متد static دیگر وجود دارد که هرکدام کار ساده‌ای را انجام می‌دهند. برای این‌که بتوانیم در متد Main() مستقیماً از این متدها (بدون ساخت شیء) استفاده کنیم باید آن‌ها را به‌صورت static تعریف کنیم.

به مثال زیر توجه کنید:

```
using System;
class Example
{
    static void Main()
    {
        double i, j;
        i = 10;
        j = 20;

        Console.WriteLine(Average(i, j));

        Console.WriteLine(Average(2, 6));

        Console.WriteLine(Average(5.3, 6.2));

    }
```

```
    static double Average(double a, double b)
    {
        return (a + b) / 2;
    }
}
```

در این برنامه از متدی استفاده کرده‌ایم که دو عدد را می‌گیرد و میانگین آن‌ها را محاسبه می‌کند. این متد را به‌صورت static تعریف کرده‌ایم، بنابراین می‌توانیم مستقیماً در متد Main() از آن استفاده کنیم. توجه کنید که اگر این متد را به‌صورت static تعریف نکنید نمی‌توانید مستقیماً در متد Main() به آن دسترسی داشته باشید. زیرا متد Main() به‌صورت static تعریف می‌شود و درصورتی می‌تواند مستقیماً به متد Average() دسترسی داشته باشد که آن هم static باشد.

در صورتی‌که متد Average() به‌صورت static نباشد باید این‌گونه عمل کنیم:

```
using System;
class Example
{
    static void Main()
    {
        double i, j;
        i = 10;
        j = 20;

        Example ob = new Example();
        Console.WriteLine(ob.Average(i, j));
        Console.WriteLine(ob.Average(2.3, 5.4));
    }
    private double Average(double a, double b)
    {
        return (a + b) / 2;
    }
}
```

همان‌طور که می‌بینید، برای استفاده از متد Average() که دیگر static نیست، مجبوریم در ابتدا یک شیء از کلاس آن بسازیم تا بتوانیم به آن دسترسی داشته باشیم.

کلاس static

شما همچنین می‌توانید یک کلاس را به‌صورت static تعریف کنید. هنگامی‌که یک کلاس را به‌صورت static تعریف می‌کنید (۱) دیگر نمی‌توانید از روی این کلاس شیء بسازید (۲) همه‌ی اعضای کلاس باید static باشند.

در مثال زیر یک ماشین حساب ساده را، از طریق کلاس static می‌سازیم:

```
using System;
static class Calculator
{
    public static double Add(double a, double b)
    {
```

```
        return a + b;
    }
    public static double Subtract(double a, double b)
    {
        return a - b;
    }
    public static double Multipliction(double a, double b)
    {
        return a * b;
    }
    public static double Division(double a, double b)
    {
        if (b == 0)
        {
            Console.WriteLine("Connat divide by zero");
            return -1;
        }
        return a / b;
    }
}
class MainClass
{
    static void Main()
    {
        int i, j;
        i = 4;
        j = 2;

        Console.WriteLine("i: {0}, j: {1}", i, j);
        Console.WriteLine("Add: {0}", Calculator.Add(i, j));
        Console.WriteLine("Subtract: {0}", Calculator.Subtract(i, j));
        Console.WriteLine("Multipliction: {0}", Calculator.Multipliction(i, j));
        Console.WriteLine("Division: {0}", Calculator.Division(i, j));

        j = 0;
        Calculator.Division(i, j);

        // Calculator ob = new Calculator(); // Wrong!
    }
}
```

همان‌طور که می‌بینید، تمام اعضای کلاس static باید static باشند و همچنین دیگر برای استفاده از این کلاس نیازی نیست از آن شیء بسازید (اگر بخواهید هم نمی‌توانید). درکل زمانی یک کلاس را به‌صورت static تعریف می‌کنید که همه‌ی اعضای آن static باشند و نیازی به ساختن شیء از آن کلاس نداشته باشید. همچنین زمانی‌که از extension method استفاده می‌کنید، باید کلاس مربوطه static باشد. در مقالات آینده با extension method آشنا می‌شوید.