زنگ سی‌شارپ – قسمت بیست و ششم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبل با چگونگی ارسال argument از طریق reference و همچنین با کلمات کلیدی out، ref و params آشنا شدید. در این قسمت قصد داریم با return کردن object از متد، Method Overloading و overload کردن constructor آشنا شویم.

Return کردن object از متد

تا این‌جا type های مختلفی را از یک متد return می‌کردیم البته در سی‌شارپ شما می‌توانید هر data type ای را از یک متد return کنید مثل int، double، float و... اما در این‌جا قصد داریم چیز دیگری را از متد بازگردانیم: class types.

در برنامه‌ی زیر کلاسی به اسم Person وجود دارد که در این کلاس متدی به اسم CreateNewPerson() است که یک شیء از جنس Person می‌سازد و این شیء را return می‌کند:

```
using System;
class ReturnObExample
{
    static void Main()
    {
        Person firstPerson = new Person("Catherine", "Gilbert");
        Console.Write("Fist Person = ");
        firstPerson.Show();

        Person secondPerson = firstPerson.CreateNewPerson("Damon", "Salvatore");

        Console.Write("Second Person = ");
        secondPerson.Show();
    }
}
class Person
{
    string Name, Family;

    public Person(string name, string family)
    {
        Name = name;
        Family = family;
    }

    public Person CreateNewPerson(string name, string family)
    {
        Person ob = new Person(name, family);
        ob.Name = name;
```

```
        ob.Family = family;
        return ob;
    }

    public void Show()
    {
        Console.WriteLine("Name: {0}, Family: {1}", Name, Family);
    }
}
```

همان‌طور که می‌بینید secondPerson را توسط متد CreateNewPerson() از شیء firstPerson به‌وجود آوردیم. در این متد یک شیء از جنس Person ساخته شده و نام و نام‌خانوادگی (از طریق پارامتر) به فیلدهای این شیء اختصاص می‌یابد و سپس reference این شیء return می‌شود. در متد Main() متد CreateNewPerson() از شیء firstPerson فراخوانی شده و شیء جدیدی که به‌وجود آورده است را به secondPerson متصل می‌کند. به این ترتیب یک شیء از جنس Person ساخته و به secondPerson reference متصل شد که دارای فیلدهای نام و نام‌خانوادگی مخصوص به خودش است.

Return کردن یک آرایه

از آن‌جا که آرایه‌ها در سی‌شارپ object هستند، یک متد همچنین می‌تواند یک آرایه را نیز return کند. برای مثال به برنامه‌ی بالا یک متد دیگر به اسم CreateFriends() اضافه کردیم که در آرایه‌ای از جنس string یک سری اسم (اسامی دوستان) را ذخیره می‌کند و در نهایت توسط متد GetFriends() این آرایه را return می‌کنیم:

```
using System;
class ReturnObExample
{
    static void Main()
    {
        Person firstPerson = new Person("Catherine", "Gilbert");
        Console.Write("Fist Person = ");
        firstPerson.Show();

        Person secondPerson = firstPerson.CreateNewPerson("Damon", "Salvatore");

        Console.Write("Second Person = ");
        secondPerson.Show();

        Console.WriteLine();

        firstPerson.CreatFriends("Stefan", "Damon", "Elena");
        string[] personFriends = firstPerson.GetFriends();

        Console.Write("Catherine's Friends: ");
        for (int i = 0; i < personFriends.Length; i++)
        {
            Console.Write(personFriends[i] + ", ");
        }
        Console.WriteLine();
    }
}
```

```
class Person
{
    string Name, Family;
    string[] Friends;
    public string[] GetFriends()
    {
        return Friends;
    }

    public Person(string name, string family)
    {
        Name = name;
        Family = family;
    }

    public Person CreateNewPerson(string name, string family)
    {
        Person ob = new Person(name, family);
        ob.Name = name;
        ob.Family = family;
        return ob;
    }

    public void Show()
    {
        Console.WriteLine("Name: {0}, Family: {1}", Name, Family);
    }

    public void CreatFriends(params string[] buddies)
    {
        Friends = new string[buddies.Length];

        for (int i = 0; i < buddies.Length; i++)
            Friends[i] = buddies[i];
    }
}
```

توجه کنید که متد GetFriends() چگونه آرایه‌ای از جنس string را بازمی‌گرداند. شما می‌توانید آرایه‌ای با ابعاد بیشتر را نیز return کنید.

Method Overloading

در سی‌شارپ دو یا بیشتر از دو متد می‌توانند نام یکسانی داشته باشند، به‌شرطی که تعریف پارامترهای آن‌ها متفاوت باشد. در این‌جور موارد گفته می‌شود که متدها overload شده‌اند و درکل به این پروسه method overloading گفته می‌شود. Method overloading یکی از جنبه‌های اجرای polymorphism (چند ریختی) است.

```
class Program
{

    static void Main()
    {
        MethodA();
        MethodA("");
```

```
    }

    static void MethodA()
    {
    }

    static void MethodA(string a)
    {
    }
}
```

همان‌طور که می‌بینید در مثال بالا دو متد هم‌نام به اسم MethodA() داریم که یکی بدون پارامتر است و دیگری یک پارامتر از جنس string دارد. درکل برای overload کردن یک متد کافی است که ورژن‌های مختلفی از آن متد را تعریف کنید. برای این کار باید یک محدودیت را در هنگام overload کردن رعایت کنید: نوع یا تعداد پارامترهای هر متد overload شده باید با بقیه متفاوت باشد. همچنین کافی نیست که فقط return-type یک متد overload شده با دیگری متفاوت باشد بلکه نوع یا تعداد پارامترهای استفاده شده در آن باید با بقیه متدهای overload شده فرق کند.

به مثال زیر توجه کنید:

```
using System;
class MethodOverloading
{
    public int Addition(int a, int b)
    {
        return a + b;
    }
    public int Addition(int a, int b, int c)
    {
        return a + b + c;
    }
    public float Addition(float a, float b)
    {
        return a + b;
    }
    public float Addition(float a, float b, float c)
    {
        return a + b + c;
    }
}

//Now you can use those Addition method four types
class hub
{
    public static void Main()
    {
        MethodOverloading methOverload = new MethodOverloading();

        Console.WriteLine("Addition of two integers: "
            + methOverload.Addition(2, 5));

        Console.WriteLine("Addition of two double type values: "
            + methOverload.Addition(0.40f, 0.50f));
```

```
        Console.WriteLine("Addition of three integers: "
            + methOverload.Addition(2, 5, 5));

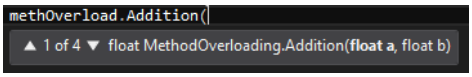
        Console.WriteLine("Addition of three double type values: "
            + methOverload.Addition(0.40f, 0.50f, 0.60f));
    }
}
```

در مثال بالا ما چندین ورژن از متد Addition() را داریم که هر کدام از نظر نوع و تعداد پارامترهای استفاده شده در آن‌ها با بقیه متفاوت هستند و همچنین return-type آن‌ها هم می‌تواند متفاوت باشد. توجه کنید که overload کردن به شکل زیر کاملاً نادرست است:

```
// One OvlDemo(int) is OK.
public void OvlDemo(int a)
{
    Console.WriteLine("One parameter: " + a);
}

/* Error! Two OvlDemo(int)s are not OK even though
return types differ. */
public int OvlDemo(int a)
{
    Console.WriteLine(“One parameter: “ + a);
    return a * a;
}
```

توجه به این نکته ضروری است که overload کردن ربطی به return-type ندارد و تنها پارامترها و جنس پارامترها اهمیت دارند. در مثال بالا با توجه به این‌که return-type متفاوت است اما به‌دلیل یکی بودن پارامترها overload اتفاق نمی‌افتد و برنامه کامپایل نمی‌شود. در ویژوال استودیو هنگامی‌که یک متد را صدا می‌زنید که چندین overload دارد با چنین چیزی مواجه می‌شوید:



در این‌جا IntelliSense ویژوال استودیو به شما نشان می‌دهد که این متد چهار overload دارد و شما می‌توانید مقادیر مختلفی را به‌عنوان argument به این متد بدهید. کافی است که کلیدهای بالا و پایین را فشار دهید تا overload های این متد را مشاهده کنید.

همان‌طور که ذکر شد، method overloading یکی از جنبه‌های اجرای polymorphism است و باعث اجرای الگوی "one interface, multiple methods" می‌شود. مثلاً در زبانی مثل C که method overloading را ساپورت نمی‌کند هر متد باید یک اسم منحصربه‌فرد داشته باشد درحالی‌که شما مکرراً نیاز دارید تا توسط یک متد روی data type های مختلف کاری

را انجام دهید. همان برنامه بالا که از متد ()Addition استفاده شده بود را در نظر بگیرید. اگر قرار بود این‌کار را توسط زبان C انجام دهیم می‌بایست چهار function مختلف با اسم‌های مختلف تعریف می‌کردیم درحالی‌که این چهار function همه‌گی یک کار یکسان را انجام می‌دهند. مسلماً این‌کار اندکی قضیه را پیچیده‌تر می‌کند زیرا با توجه به این‌که همه‌ی این function ها یک کار را انجام می‌دهند و مفهوم یکسانی دارند شما مجبورید ۴ اسم مختلف را به‌خاطر بسپارید. مثلاً متد ()WriteLine از کلاس Console نوزده overload دارد. تصور کنید در این موارد که تعداد overload ها زیاد است آن‌گاه تعریف کردن متدهای جداگانه با اسامی مجزا عملاً کار احمقانه‌ای است.

Overload Constructors

Constructor ها نیز می‌توانند همانند متدها overload شوند. این‌کار باعث می‌شود بتوانید به طرق مختلفی object های خود را بسازید. به مثال زیر توجه کنید:

```
// Demonstrate an overloaded constructor.
using System;
class MyClass
{
    public int x;

    public MyClass()
    {
        Console.WriteLine("Inside MyClass().");
        x = 0;
    }

    public MyClass(int i)
    {
        Console.WriteLine("Inside MyClass(int).");
        x = i;
    }
    public MyClass(double d)
    {
        Console.WriteLine("Inside MyClass(double).");
        x = (int)d;
    }

    public MyClass(int i, int j)
    {
        Console.WriteLine("Inside MyClass(int, int).");
        x = i * j;
    }
}
class OverloadConsDemo
{
    static void Main()
    {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass(88);
        MyClass t3 = new MyClass(17.23);
        MyClass t4 = new MyClass(2, 4);
```

```
        Console.WriteLine("t1.x: " + t1.x);
        Console.WriteLine("t2.x: " + t2.x);
        Console.WriteLine("t3.x: " + t3.x);
        Console.WriteLine("t4.x: " + t4.x);
    }
}
```

خروجی:

```
Inside MyClass().
Inside MyClass(int).
Inside MyClass(double).
Inside MyClass(int, int).
t1.x: 0
t2.x: 88
t3.x: 17
t4.x: 8
```

در این‌جا MyClass() چهار overload دارد که هر کدام باعث می‌شوند object به شکل متفاوتی ساخته شود. توسط کلمه‌کلیدی new و argument هایی که برای ساخت شیء به‌کار می‌برید، constructor مربوط به آن فراخوانی می‌شود و شیء را به‌وجود می‌آورد. با overload کردن constructor های کلاس، شما این امکان را به‌وجود می‌آورید که object ها به طرق مختلفی بتوانند ساخته شوند.

به مثال بعدی توجه کنید:

```
using System;
class Employee
{
    public int IdNumber;
    public double Salary;

    public Employee()
    {
        IdNumber = 999;
        Salary = 0;
    }
    public Employee(int empId)
    {
        IdNumber = empId;
        Salary = 0;
    }
    public Employee(int empId, double sal)
    {
        IdNumber = empId;
        Salary = sal;
    }
    public Employee(char code)
    {
        IdNumber = 111;
        Salary = 100000;
    }
}
public class CreateSomeEmployees
```

```
{
    public static void Main()
    {
        Employee aWorker = new Employee();
        Employee anotherWorker = new Employee(234);
        Employee theBoss = new Employee('A');

        Console.WriteLine("{0} --- {1}", aWorker.IdNumber,
        aWorker.Salary);

        Console.WriteLine("{0} --- {1}", anotherWorker.IdNumber,
        anotherWorker.Salary);

        Console.WriteLine("{0} --- {1}", theBoss.IdNumber,
        theBoss.Salary);
    }
}
```

در این برنامه چهار overload از Employee() وجود دارد که هرکدام به‌نحوی باعث ساخت شیء می‌شوند.

درخواست یک overloaded constructor از طریق this

هنگامی‌که با constructor های overload شده کار می‌کنید، یک constructor می‌تواند، constructor دیگری را درخواست کند. این‌کار از طریق کلمه‌کلیدی this انجام می‌شود و فرم کلی آن به‌شکل زیر است:

```
constructor-name(parameter-list1) : this(parameter-list2) {
    // ... body of constructor, which may be empty
}
```

در این‌جا ابتدا با توجه به parameter-list2 یکی از constructor های overload شده اجرا می‌شود و سپس اگر کدی درون constructor اصلی (constructor اولیه) وجود داشته باشد، اجرا می‌شود.

به مثال زیر توجه کنید:

```
// Demonstrate invoking a constructor through this.
using System;
class AlphaBeta
{
    public int Alpha, Beta;
    public AlphaBeta() : this(0, 0)
    {
        Console.WriteLine("Inside AlphaBeta()");
    }
    public AlphaBeta(AlphaBeta obj) : this(obj.Alpha, obj.Beta)
    {
        Console.WriteLine("Inside AlphaBeta(obj)");
    }
    public AlphaBeta(int i, int j)
    {
        Console.WriteLine("Inside AlphaBeta(int, int)");
        Alpha = i;
        Beta = j;
```

```
    }
}
class OverloadConsDemo
{
    static void Main()
    {
        AlphaBeta ob1 = new AlphaBeta();
        AlphaBeta ob2 = new AlphaBeta(8, 9);
        AlphaBeta ob3 = new AlphaBeta(ob2);

        Console.WriteLine();
        Console.WriteLine("ob1.x, ob1.y: " + ob1.Alpha + ", " + ob1.Beta);
        Console.WriteLine("ob2.x, ob2.y: " + ob2.Alpha + ", " + ob2.Beta);
        Console.WriteLine("ob3.x, ob3.y: " + ob3.Alpha + ", " + ob3.Beta);
    }
}
```

خروجی:

```
Inside AlphaBeta(int, int)
Inside AlphaBeta ()
Inside AlphaBeta (int, int)
Inside AlphaBeta (int, int)
Inside AlphaBeta (obj)

ob1.x, ob1.y: 0, 0
ob2.x, ob2.y: 8, 9
ob3.x, ob3.y: 8, 9
```

به این نکته توجه کنید، پارامترهایی که بعد از this می‌آیند مشخص می‌کنند که کدامین constructor باید در ابتدا اجرا شود. مثلاً برای AlphaBeta() که خودش پارامتری ندارد، this(0, 0) دو عدد integer دارد و مشخص می‌کند که ابتدا باید آن constructor ای اجرا شود که دو پارامتر int دارد. بنابراین ابتدا محتویات AlphaBeta(int i, int j) اجرا شده و سپس محتویات AlphaBeta() اجرا می‌شود. در مورد AlphaBeta(AlphaBeta obj) نیز مراحل به‌ترتیب قبل است. This(obj.Alpha, obj.Beta) دو عدد int گرفته است بنابراین ابتدا محتویات AlphaBeta(int i, int j) اجرا شده و سپس محتویات AlphaBeta(AlphaBeta obj) اجرا می‌شود. یکی از مزایای این کار این است که از کدنویسی اضافی جلوگیری می‌کند. با این‌کار شما دیگر در هر constructor نیاز ندارید که برای مقداردهی به Alpha و Beta کد تکراری بنویسید.

به مثال دیگری در این زمینه توجه کنید:

```
using System;
class Mouse
{
    public Mouse()
        : this(-1, "")
    {
        // Uses constructor initializer.
    }

    public Mouse(int weight, string name)
```

```
    {
        // Constructor implementation.
        Console.WriteLine("Constructor weight = {0}, name = {1}",
            weight,
            name);
    }
}

class Program
{
    static void Main()
    {
        // Test the two constructors for Mouse type.
        Mouse mouse1 = new Mouse();
        Mouse mouse2 = new Mouse(10, "Sam");
    }
}
```

تمرین شماره ۱۴: در این تمرین می‌بایست یک جعبه‌ی موسیقی درست کنید. این جعبه موسیقی، باید چندین هنرمند داشته باشد و هر هنرمند می‌تواند چندین آلبوم و تک آهنگ داشته باشد. همچنین می‌بایست قابلیت Play و Stop کردن را به این جعبه موسیقی بدهید و بتوانید آلبوم‌ها و آهنگ‌ها را حذف و اضافه کنید.

راهنمایی:

```
using System;
using System.Media;
class MyClass
{
    static void Main()
    {
        SoundPlayer myPlayer = new SoundPlayer(@"c:\mywavfile.wav");
        myPlayer.Play();
        myPlayer.Stop();
    }
}
```

این تمرین را تا آن‌جا که می‌توانید شی‌گرا بنویسید. در قسمت بعد حل تمرین روی سایت قرار می‌گیرد.

توجه: زین پس هر یک از دوستان که جواب تمرین‌های زنگ سی‌شارپ را برای ما ایمیل کند، به عنوان تشویقی یک نکته‌ی سی‌شارپ به‌صورت شخصی برای وی فرستاده می‌شود.

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.