زنگ سی‌شارپ – قسمت پنجاه (قسمت پایانی)

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

استفاده از Method Group Conversion

از نسخه‌ی ۲، یک ویژگی به سی‌شارپ اضافه شد که به‌شکل قابل توجهی اختصاص‌دهی method به delegate را ساده می‌کرد. این ویژگی method group conversion نام دارد و به شما اجازه می‌دهد تا به سادگی، نام یک متد را به delegate اختصاص دهید بدون این‌که نیاز داشته باشید از کلمه‌ی کلیدی new استفاده کنید یا constructor مربوط به delegate را فراخوانی کنید.

به مثال زیر دقت کنید:

```
using System;

class Program
{
    delegate string UppercaseDelegate(string input);

    static string UppercaseFirst(string input)
    {
        char[] buffer = input.ToCharArray();
        buffer[0] = char.ToUpper(buffer[0]);
        return new string(buffer);
    }

    static string UppercaseLast(string input)
    {
        char[] buffer = input.ToCharArray();
        buffer[buffer.Length - 1] = char.ToUpper(buffer[buffer.Length - 1]);
        return new string(buffer);
    }

    static string UppercaseAll(string input)
    {
        return input.ToUpper();
    }

    static void WriteOutput(string input, UppercaseDelegate del)
    {
        Console.WriteLine("Your string before: {0}", input);
        Console.WriteLine("Your string after: {0}", del(input));
    }

    static void Main()
    {
        // using method group conversion
```

```
        WriteOutput("perls", UppercaseFirst);
        WriteOutput("perls", UppercaseLast);
        WriteOutput("perls", UppercaseAll);
    }
}

/* Output

Your string before: perls
Your string after: Perls
Your string before: perls
Your string after: perlS
Your string before: perls
Your string after: PERLS

 */
```

این مثال را در قسمت قبل نیز مشاهده کردید با این تفاوت که در این مثال از ویژگی Method group conversion استفاده شده است.

دقت کنید که در مثال‌های قبل، از متدهای static استفاده می‌کردید. در مثال بعد مشاهده می‌کنید که می توانید از متدهای یک شیء استفاده کنید که static نیستند (instance methods).

```
using System;
delegate string StrMod(string str);
class StringOps
{
    // Replaces spaces with hyphens.
    public string ReplaceSpaces(string s)
    {
        Console.WriteLine("Replacing spaces with hyphens.");
        return s.Replace(' ', '-');
    }
}
class DelegateTest
{
    static void Main()
    {
        StringOps so = new StringOps(); // create an instance of StringOps
        // Initialize a delegate.
        StrMod strOp = so.ReplaceSpaces;
        string str;
        // Call methods through delegates.
        str = strOp("This is a test.");
        Console.WriteLine("Resulting string: " + str);
    }
}
```

همان‌طور مشاهده می‌کنید، ابتدا از روی کلاس مربوطه یک شیء ساخته و سپس از طریق آن شیء، متد را صدا زده‌ایم.

دقت کنید که در مثال بالا نیز از method group conversion استفاده شده است.

متدهای بی‌نام (Anonymous Methods)

یک anonymous method راهی برای ساختن یک بلوک کد بدون نام است که به یک delegate instance اختصاص می‌یابد. به مثال زیر توجه کنید:

```
using System;
delegate void CountIt();
class AnonMethDemo
{
    static void Main()
    {
        CountIt count = delegate
        {
            for (int i = 0; i <= 5; i++)
                Console.WriteLine(i);
        }; // notice the semicolon
        count();
    }
}

/* Output

0
1
2
3
4
5

*/
```

در این برنامه ابتدا یک delegate type به اسم CountIt تعریف کرده‌ایم. درون متد اصلی، یک instance از این delegate تعریف کرده و آن را با یک بلوک کد مساوی قرار داده‌ایم که قبل از آن بلوک، کلمه‌ی کلیدی delegate را می‌بینید. این بلوک کد، anonymous method است که بعد از فراخوانی count، اجرا می‌شود. به semicolon انتهای بلوک نیز توجه داشته باشید.

به مثال بعد توجه کنید:

```
using System;
// Notice that CountIt now has a parameter.
delegate void CountIt(int end);
class AnonMethDemo2
{
    static void Main()
    {
        // Here, the ending value for the count
        // is passed to the anonymous method.
        CountIt count = delegate(int end)
        {
            for (int i = 0; i <= end; i++)
```

```
                Console.WriteLine(i);
        };
        count(3);
        Console.WriteLine();
        count(5);
    }
}

/* Output

0
1
2
3
0
1
2
3
4
5

*/
```

همان‌طور که می‌بینید، CountIt در مثال بالا شامل یک integer argument است. پارامتر نیز بعد از کلمه‌ی کلیدی delegate مشخص شده است.

سی‌شارپ دو نوع از anonymous function را تعریف می‌کند که عبارتند از anonymous method و lamba expression. تا این‌جا با anonymous method آشنا شدید. در ادامه با lamba expression آشنا خواهید شد.

Lambda Expression

Lambda expression راهی دیگر برای ساخت anonymous function است. از این‌رو، lambda expression می‌تواند به delegate اختصاص داده شود. به‌دلیل این‌که lambda expression راحت‌تر از anonymous method معادل است، پیشنهاد می‌شود که تقریبا در همه‌ی موارد از lambda expression استفاده شود.

تمامی lambda expression ها از lambda operator استفاده می‌کنند که عبارت است از: =>

این operator یک lambda را به دو قسمت تقسیم می‌کند. در سمت چپ، پارامترهای ورودی و در سمت راست، بدنه‌ی lambda مشخص می‌شود.

به مثال زیر توجه کنید:

```
using System;

delegate int Incr(int v);
delegate bool IsEven(int v);

class SimpleLambdaDemo
{
    static void Main()
    {
        // A lambda expression that increases its parameter by 2.
        Incr incr = count => count + 2;

        Console.WriteLine("Use incr lambda expression: ");
        int x = -10;
        while (x <= 0)
        {
            Console.Write(x + " ");
            x = incr(x); // increase x by 2
        }
        Console.WriteLine("\n");

        // a lambda expression that returns true if its parameter
        // is even and false otherwise.
        IsEven isEven = n => n % 2 == 0;

        Console.WriteLine("Use isEven lambda expression: ");
        for (int i = 1; i <= 10; i++)
            if (isEven(i)) Console.WriteLine(i + " is even.");
    }
}

/* Output

se incr lambda expression:
-10 -8 -6 -4 -2 0
Use isEven lambda expression:
2 is even.
4 is even.
6 is even.
8 is even.
10 is even.

*/
```

در برنامه‌ی بالا به این عبارات دقت کنید:

```
Incr incr = count => count + 2;

IsEven isEven = n => n % 2 == 0;
```

عبارت اول به incr یک lambda expression را اختصاص می‌دهد که مقدار فرستاده شده به count را با ۲ جمع کرده و نتیجه را return می‌کند. این عبارت می‌تواند به Incr delegate اختصاص یابد زیرا با تعریف Incr تطابق دارد. در عبارت دوم، اگر حاصل lambda یک عدد زوج باشد، مقدار true و اگر حاصل عددی فرد باشد، مقدار false را return خواهیم کرد.

به مثال زیر توجه کنید:

```
using System;

delegate int IntOp(int end);

class StatementLambdaDemo
{
    static void Main()
    {
        IntOp fact = n =>
        {
            int r = 1;
            for (int i = 1; i <= n; i++)
                r = i * r;
            return r;
        };

        Console.WriteLine("The factorial of 3 is " + fact(3));
        Console.WriteLine("The factorial of 5 is " + fact(5));
    }
}

/* Output

The factorial of 3 is 6
The factorial of 5 is 120

*/
```

تفاوت مثال بالا با مثال قبل، این است که در این مثال، lambda شامل بدنه است.

Events

Event یکی دیگر از ویژگی‌های مهم سی‌شارپ است که بر اساس delegate می‌باشد. یک object می‌تواند برای یک event تعدادی event handler را register کند و هنگامی‌که یک event اتفاق می‌افتد، تمامی handler های register شده، فراخوانی می‌شوند. Event handler ها باید مطابق با delegate باشند.

Event ها اعضای کلاس هستند و توسط کلمه‌ی کلیدی event تعریف می‌شوند. فرم آن به شکل زیر است:

```
event event-delegate event-name;
```

در این‌جا، event-delegate نام آن delegate است که این event را support می‌کند و event-name نام همین event object است که تعریف کرده‌ایم.

به مثال زیر توجه کنید:

```
using System;
delegate void MyEventHandler();
class MyEvent
{
    public event MyEventHandler SomeEvent;
    public void OnSomeEvent()
    {
        if (SomeEvent != null)
            SomeEvent();
    }
}
class EventDemo
{
    static void Handler()
    {
        Console.WriteLine("Event Occurred!");
    }
    static void Main()
    {
        MyEvent evt = new MyEvent();
        evt.SomeEvent += Handler;
        evt.OnSomeEvent();
    }
}

/* Output

Event Occurred!

*/
```

این برنامه با تعریف delegate type برای event handler شروع می‌شود:

```
delegate void MyEventHandler();
```

تمام event ها از طریق delegate فعال می‌شوند. از این‌رو، event delegate type، نوع بازگشتی و signature را برای event مشخص می‌کند.

سپس، کلاس MyEvent تعریف شده که درون آن، یک event به نام SomeEvent تعریف شده است:

```
public event MyEventHandler SomeEvent;
```

همچنین، درون کلاس MyEvent یک متد به نام OnSomeEvent() وجود دارد که برای fire کردن event استفاده می‌شود. یعنی این همان متدی است که وقتی event اتفاق می‌افتد، فراخوانی می‌شود. این متد، event handler را از طریق SomeEvent delegate فراخوانی می‌کند:

```
if (SomeEvent != null)
            SomeEvent();
```

دقت کنید که handler درصورتی فراخوانی می‌شود که SomeEvent برابر با null نباشد. درون کلاس EventDemo یک متد به اسم Handler() وجود داردکه درون متد Main() به‌عنوان handler برای event شیء تعریف شده، register می‌شود:

```
MyEvent evt = new MyEvent();
evt.SomeEvent += Handler;
```

دقت کنید که handler از طریق += افزوده شده است. event ها فقط از += و -= پیشتیبانی می‌کنند. در نهایت می‌بینید که event به‌صورت زیر fire شده است:

```
evt.OnSomeEvent();
```

فراخوانی OnSomeEvent() موجب می‌شود تا تمامی event handler های register شده، فراخوانی شوند. در این مورد تنها یک handler را register کرده بودیم اما می‌توانید تعداد بیشتری را نیز register کنید.

در مثال زیر نحوه‌ی استفاده از lambda expression با event را می‌بینید:

```
using System;
delegate void MyEventHandler(int n);
class MyEvent
{
    public event MyEventHandler SomeEvent;
    public void OnSomeEvent(int n)
    {
        if (SomeEvent != null)
            SomeEvent(n);
    }
}
class LambdaEventDemo
{
    static void Main()
    {
        MyEvent evt = new MyEvent();

        // Use a lambda expression as an event handler.
        evt.SomeEvent += (n) =>
        Console.WriteLine("Event received. Value is " + n);

        // Raise (fire) the event twice.
        evt.OnSomeEvent(1);
        evt.OnSomeEvent(2);
    }
}

/* Output

Event received. Value is 1
Event received. Value is 2

*/
```

استفاده از List در سی‌شارپ

یک آرایه به‌صورت پویا اندازه‌اش تغییر نمی‌کند اما List به‌صورت پویا resize می‌شود. با استفاده از List دیگر نیاز ندارید تا اندازه‌ی آن را ابتدای کار مشخص کنید و در واقع با استفاده از List آرایه‌ای می‌سازید که اندازه‌ی محدود نیست.

فرم کلی List به شکل زیر است:

```
List<type> list-name = new List<type>();
```

در این‌جا، کلمه‌ی کلیدی List بیان‌کننده‌ی این امر است که قصد ساخت یک collection از نوع List را داریم. type مشخص‌کننده‌ی جنس List است که می‌تواند int و string و ... باشد.

در مثال زیر ابتدا یک List از جنس int تعریف می‌کنیم (بدون مشخص کردن اندازه) و سپس مقادیری را به آن می‌افزاییم. مقادیر به همان ترتیبی که افزوده شده‌اند در List ذخیره می‌شوند:

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main()
    {
        List<int> list = new List<int>();
        list.Add(2);
        list.Add(3);
        list.Add(5);
        list.Add(7);

        foreach (int item in list) {
            Console.WriteLine(item);
        }

        // Or
        Console.WriteLine();

        // Count property on the List type, is equal to Length on arrays.
        for (int i = 0; i < list.Count; i++) {
            Console.WriteLine(list[i]);
        }

    }
}

/* Output

2
3
5
7

2
```

```
3
5
7

*/
```

در این مثال، ابتدا یک List از جنس int تعریف کرده و سپس با فراخوانی متد ()Add مقادیر مختفی را در list ذخیره کرده‌ایم. در ادامه می‌بینید که با استفاده از حلقه foreach می‌توانید به‌سادگی مقادیر درون list را نمایش دهید. هچنین مشاهده می‌کنید که توسط حلقه‌ی for نیز می‌توانید به مقادیر درون list دسترسی داشته باشید. نکته مورد توجه در حلقه‌ی for برای استفاده از list، Count property است که معادل length property در هنگام استفاده از آرایه می‌باشد.

به مثال بعد توجه کنید:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
       List<bool> list = new List<bool>();
       list.Add(true);
       list.Add(false);
       list.Add(true);
       Console.WriteLine(list.Count); // 3

       list.Clear();
       Console.WriteLine(list.Count); // 0
    }
}

/* Output

0
3

*/
```

همان‌طور که در مثال بالا مشاهده می‌کنید، list.Count تعداد عناصر ذخیره شده در List است. بعد از فراخوانی ()Clear تمامی عناصر ذخیره شده در List از بین می‌روند.

به مثال بعد توجه کنید:

```
using System;
using System.Collections.Generic;

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

```
        int[] arr = new int[3]; // New array with 3 elements
        arr[0] = 2;
        arr[1] = 3;
        arr[2] = 5;
        List<int> list = new List<int>(arr); // Copy to List
        Console.WriteLine(list.Count);       // 3 elements in List
    }
}


/* Output

3

*/
```

همان‌طور که می‌بینید، می‌توانید یک آرایه‌ی از پیش آماده شده را مستقیماً درون یک List کپی کنید.

به مثال زیر توجه کنید:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<int> list = new List<int>(new int[] { 19, 23, 29 });

        // Finds first element greater than 20
        int result = list.Find(item => item > 20);

        Console.WriteLine(result);
    }
}


/* Output

23

*/
```

یکی دیگر از متدهای List متد ()Find است که غالباً از lambda expression استفاده می‌کند.

List شامل متدهای بسیاری است که برای مطالعه‌ی آن‌ها می‌توانید این لینک را ببینید.

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.