Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 06

Example

A Common Code developer wants to create a generic method for filtering an array of integers, but with the ability to specify the algorithm for filtration to the Application developer

Solution

The Common Code Developer:

```
public delegate bool IntFilter(int i);
public class Common
{
    public static int[] FilterArray(int[] ints, IntFilter filter)
    {
        ArrayList aList = new ArrayList();
        foreach(int i in ints)
        {
            if(filter(i))
            {aList.Add(i);}
        }
        return((int[])aList.ToArray(typeof(int)));
    }
}
```

Solution

The Application Code Developer (I)

```
Class MyApplication
{
        public static bool IsOdd(int i)
        {
                return ((i % 2)==1);
        }
        public static void Main()
        {
                int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
                int[] fnum = Common.FilterArray(nums, MyApplication.IsOdd);
                foreach(int i in fnum)
                        Console.WriteLine(i);
        }
}
```

Solution

The Application Code Developer, Anonymous method “C#2.0” (II)

```
Class MyApplication
{
        public static void Main()
        {
                int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
                int[] fnum = Common.FilterArray(nums, delegate(int i)
                                        {return((i%2)==1);});
                foreach(int i in fnum)
                        Console.WriteLine(i);
        }
}
```

Solution

The Application Code Developer, Lambda Expression “C#3.0” (III)

```
Class MyApplication
{
        public static void Main()
        {
                int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
                int[] fnum = Common.FilterArray(nums, j=>((j % 2)==1));
                foreach(int i in fnum)
                        Console.WriteLine(i);
        }
}
```

Events

An event in C# is a way for a class (Sender) to provide notifications to clients (Listener) of that class when some interesting thing happens to an object. The most familiar use for events is in graphical user interfaces

The ideal case that we have 2 classes:

1)Sender, which must contains the events. The event itself didn't know what function will be executed, he knows what is the Signature of the function to be executed and its return type (delegate). In the event declaration, it must specify the delegate. Also, the Sender Class must contains the code that fire the event.

2)Listener, contains the relation of the events with the method that will be executed when the event fire. This is done through the delegate. It also contains the implementation of the methods to be executed when the event fire.

Example:

Write a program that read a number from the user, this number is between 1 and 100. If the number is Less than 1, it print a message on the screen "Less than 1" and if it is greater than 100, it prints "Out of Range"

Events

1) The Sender

```
public delegate void ValueHandler(string msg);
public class Sender
{
        public int I;
        //The Sender class can send these 2 events
        public event ValueHandler Low;
        public event ValueHandler High;
        public void ReadData()
        {
                I = int.Parse(Console.ReadLine());
                If(I < 1)
                {
                        if(Low != null)
                        {
                                Low("Out of Range, Number is too small");
                        }
                }
```

Events

1) The Sender

```
                else
                {
                        if(I > 100)
                        {
                                if(High !=null)
                                {
                                        High("Out of Range, Number is too large");
                                }
                        }
                }
        }//End Read Data
}//End Sender Class
```

Events

2) The Listening to the incoming Event

```
public class EventApp
{       public static int Main()
        {       Sender s = new Sender();
                //Hooks into events
                s.Low  += new ValueHandler(OnLow);
                s.High += new ValueHandler(OnHigh);
                //Call the Read Data
                s.ReadData();
        }
        public void OnLow(string str)
        {
                Console.WriteLine("{0}", str);
        }
        public void OnHigh(string str)
        {
                Console.WriteLine("{0}", str);
        }
}
```