C# - Delegate

A function can have one or more parameters of different data types, but what if you want to pass a function itself as a parameter? How does C# handle the callback functions or event handler? The answer is - **delegate**.

A delegate is like a pointer to a function. It is a reference type data type and it holds the reference of a method. All the delegates are implicitly derived from `System.Delegate` class.

A delegate can be declared using **delegate** keyword followed by a function signature as shown below.

Delegate Syntax:

<access modifier> delegate <return type> delegate_name>(<parameters>)

The following example declares a Print delegate.

Example: Declare delegate

```
public delegate void Print(int value);
```

The Print delegate shown above, can be used to point to any method that has same return type & parameters declared with Print. Consider the following example that declares and uses Print delegate.

Example: C# delegate

```
class Program
{
    // declare delegate
    public delegate void Print(int value);

    static void Main(string[] args)
    {
        // Print delegate points to PrintNumber
        Print printDel = PrintNumber;

        // or
        // Print printDel = new Print(PrintNumber);

        printDel(100000);
        printDel(200);

        // Print delegate points to PrintMoney
        printDel = PrintMoney;

        printDel(10000);
```

```
        printDel(200);
    }

    public static void PrintNumber(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}",num);
    }

    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

Output:

Number: 10,000
Number: 200
Money: $ 10,000.00
Money: $ 200.00

In the above example, we have declared Print delegate that accepts *int* type parameter and returns void. In the Main() method, a variable of Print type is declared and assigned a PrintNumber method name. Now, invoking Print delegate will in-turn invoke PrintNumber method. In the same way, if the Print delegate variable is assigned to the PrintMoney method, then it will invoke the PrintMoney method.

Optionaly, a delegate object can be created using the `new` operator and specify a method name, as shown below:

```
Print printDel = new Print(PrintNumber);
```

Invoking Delegate

The delegate can be invoked like a method because it is a reference to a method. Invoking a delegate will in-turn invoke a method which id refered to. The delegate can be invoked by two ways: using () operator or using the Invoke() method of delegate as shown below.

Example: Invoking a delegate

```
Print printDel = PrintNumber;
printDel.Invoke(10000);

//or
printDel(10000);
```

Pass Delegate as a Parameter

A method can have a parameter of a delegate type and can invoke the delegate parameter.

Example: Delegate Parameter

```
public static void PrintHelper(Print delegateFunc, int numToPrint)
{
    delegateFunc(numToPrint);
}
```

In the above example, PrintHelper method has a delegate parameter of Print type and invokes it like a function: `delegateFunc(numToPrint)`.

The following example shows how to use PrintHelper method that includes delegate type parameter.

Example: Delegate parameter

```
class Program
{
    public delegate void Print(int value);


    static void Main(string[] args)
    {
        PrintHelper(PrintNumber, 10000);
        PrintHelper(PrintMoney, 10000);
    }

    public static void PrintHelper(Print delegateFunc, int numToPrint)
    {
        delegateFunc(numToPrint);
    }

    public static void PrintNumber(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}",num);
    }

    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

Output:

Number: 10,000
Money: $ 10,000.00

Multicast Delegate

The delegate can points to multiple methods. A delegate that points multiple methods is called a multicast delegate. The "+" operator adds a function to the delegate object and the "-" operator removes an existing function from a delegate object.

Example: Multicast delegate

```
public delegate void Print(int value);

static void Main(string[] args)
{
    Print printDel = PrintNumber;
    printDel += PrintHexadecimal;
    printDel += PrintMoney;

    printDel(1000);

    printDel -=PrintHexadecimal;
    printDel(2000);
}

public static void PrintNumber(int num)
{
    Console.WriteLine("Number: {0,-12:N0}",num);
}

public static void PrintMoney(int money)
{
    Console.WriteLine("Money: {0:C}", money);
}

public static void PrintHexadecimal(int dec)
{
    Console.WriteLine("Hexadecimal: {0:X}", dec);
}
```

Output:

Number: 1,000
Hexadecimal: 3EB
Money: $ 1,000.00

Number: 2,000
Money: $2,000.00

As you can see in the above example, Print delegates becomes a multicast delegate because it points to three methods - PrintNumber, PrintMoney & PrintHexadecimal. So invoking printDel will invoke all the methods sequentially.

Delegate is also used with Event, Anonymous method, Func delegate, Action delegate.

Points to Remember :

1. Delegate is a function pointer. It is reference type data type.
2. Syntax: public delegate void <function name>(<parameters>)
3. A method that is going to assign to delegate must have same signature as delegate.
4. Delegates can be invoke like a normal function or Invoke() method.
5. Multiple methods can be assigned to the delegate using "+" operator. It is called multicast delegate.

C# - Event

In general terms, an event is something special that is going to happen. For example, Microsoft launches events for developers, to make them aware about the features of new or existing products. Microsoft notifies the developers about the event by email or other advertisement options. So in this case, Microsoft is a publisher who launches (raises) an **event** and **notifies** the developers about it and developers are the **subscribers** of the event and attend (**handle**) the event.

Events in C# follow a similar concept. An event has a publisher, subscriber, notification and a handler. Generally, UI controls use events extensively. For example, the button control in a Windows form has multiple events such as click, mouseover, etc. A custom class can also have an event to notify other subscriber classes about something that has happened or is going to happen. Let's see how you can define an event and notify other classes that have event handlers.

An event is nothing but an encapsulated delegate. As we have learned in the previous section, a delegate is a reference type data type. You can declare the delegate as shown below:

Example: Delegate

```
public delegate void someEvent();

public someEvent someEvent;
```

Now, to declare an event, use the **event** keyword before declaring a variable of delegate type, as below:

Example: Event Declaration

```
public delegate void SomeEvent();

public event SomeEvent someEvent;
```

Thus, a delegate becomes an event using the **event** keyword.

Now, let's see a practical example of an event. Consider the following PrintHelper class that prints integers in different formats like number, money, decimal, temperature and hexadecimal. It includes a beforePrintEvent to notify the subscriber of the BeforePrint event before it going to print the number.

Example: Event

```
public class PrintHelper
{
    // declare delegate
    public delegate void BeforePrint();

    //declare event of type delegate
    public event BeforePrint beforePrintEvent;

    public PrintHelper()
    {

    }

    public void PrintNumber(int num)
    {
        //call delegate method before going to print
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Decimal: {0:G}", dec);
```

```
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Money: {0:C}", money);
    }

    public void PrintTemperature(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Temperature: {0,4:N1} F", num);
    }
    public void PrintHexadecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Hexadecimal: {0:X}", dec);
    }
}
```

The delegate can also be invoked using the Invoke() method, e.g., beforePrintEvent.Invoke().

PrintHelper is a publisher class that publishes the beforePrint event. Notice that in each print method, it first checks to see if beforePrintEvent is not null and then it calls *beforePrintEvent()*. beforePrintEvent is an object of type BeforePrint delegate, so it would be null if no class is subscribed to the event and that is why it is necessary to check for null before calling a delegate.

Now, let's create a subscriber. Consider the following simple Number class for example.

Example: Event subscriber

```
class Number
{
    private PrintHelper _printHelper;

    public Number(int val)
    {
        _value = val;

        _printHelper = new PrintHelper();
                //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }
```

```
//beforePrintevent handler
    void printHelper_beforePrintEvent()
    {
        Console.WriteLine("BeforPrintEventHandler: PrintHelper is going to print a
value");
    }

    private int _value;

    public int Value
    {
        get { return _value; }
        set { _value = value; }
    }

    public void PrintMoney()
    {
        _printHelper.PrintMoney(_value);
    }

    public void PrintNumber()
    {
        _printHelper.PrintNumber(_value);
    }
}
```

All the subscribers must provided a handler function, which is going to be called when a publisher raises an event. In the above example, the Number class creates an instance of PrintHelper and subscribes to the beforePrintEvent with the "+=" operator and gives the name of the function which will handle the event (it will be get called when publish fires an event). *printHelper_beforePrintEvent* is the event handler that has the same signature as the BeforePrint delegate in the PrintHelper class.

So now, create an instance of Number class and call print methods:

Example: Event

```
Number myNumber = new Number(100000);
myNumber.PrintMoney();
myNumber.PrintNumber();
```

Output:

BeforePrintEventHandler: PrintHelper is going to print value
Money: $ 1,00,000.00
BeforePrintEventHandler: PrintHelper is going to print value
Number: 1,00,000

The following image illustrates an event model:

Event Arguments

Events can also pass data as an argument to their subscribed handler. An event passes arguments to the handler as per the delegate signature. In the following example, PrintHelper declares the BeforePrint delegate that accepts a string argument. So now, you can pass a string when you raise an event from PrintNumber or any other Print method.

Example: Event Arguments

```
public class PrintHelper
{
    public delegate void BeforePrint(string message);
    public event BeforePrint beforePrintEvent;


    public void PrintNumber(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent.Invoke("PrintNumber");

        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintDecimal");

        Console.WriteLine("Decimal: {0:G}", dec);
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintMoney");

        Console.WriteLine("Money: {0:C}", money);
    }

    public void PrintTemperature(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintTemerature");

        Console.WriteLine("Temperature: {0,4:N1} F", num);
    }

    public void PrintHexadecimal(int dec)
    {
```

```
        if (beforePrintEvent != null)
            beforePrintEvent("PrintHexadecimal");

        Console.WriteLine("Hexadecimal: {0:X}", dec);
    }
}
```

Now, the subscriber class should have an event handler that has a string parameter.

In the following example, Number class has a printHelper_beforePrintEvent function with string parameter.

Example: Event

```
class Number
{
    private PrintHelper _printHelper;

    public Number(int val)
    {
        _value = val;

        _printHelper = new PrintHelper();
        //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }
    //beforePrintevent handler
    void printHelper_beforePrintEvent(string message)
    {
        Console.WriteLine("BeforePrintEvent fires from {0}",message);
    }

    private int _value;

    public int Value
    {
        get { return _value; }
        set { _value = value; }
    }

    public void PrintMoney()
    {
        _printHelper.PrintMoney(_value);
    }

    public void PrintNumber()
    {
        _printHelper.PrintNumber(_value);
    }

}
```

Output:

```
BeforePrintEvent fires from PrintMoney.
Money: $ 1,00,000.00
BeforePrintEvent fires from PrintNumber.
Number: 1,00,000
```

Points to Remember :

1. Use event keyword with delegate type to declare an event.
2. Check event is null or not before raising an event.
3. Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.
4. Function that handles the event is called event handler. Event handler must have same signature as declared by event delegate.
5. Events can have arguments which will be passed to handler function.
6. Events can also be declared static, virtual, sealed and abstract.
7. An Interface can include event as a member.
8. Events will not be raised if there is no subscriber
9. Event handlers are invoked synchronously if there are multiple subscribers
10. The .NET framework uses an `EventHandler` delegate and an `EventArgs` base class.