C SHARP DELEGATES AND PLUG-IN METHODS WITH DELEGATES

Delegate is a reference type that holds the reference of a class method. Any method which has the same signature as delegate can be assigned to delegate. Delegate is like function pointer in C++. There are three aspects to a delegate: Declaration, Instantiation and Invocation.

Declaration:

A delegate is declared by using the keyword delegate, otherwise it resembles a method declaration.

```
1.  delegate int delegateAdd (int x,int y);
```

Instantiation:

To create a delegate instance, we need to assign a method (which has same signature as delegate) to delegate.

```
1.  static int Add (int x,int y)
2. {
3.  return x+y;
4.  }
5. ..
6. //create delegate instance
7. delegateAdd objAdd= new delegateAdd (Add);
8. //short hand for above statement
9. delegateAdd objAdd=Add;
```

Invocation:

Invoking a delegate is like as invoking a regular method.

```
1.  // Invoke delegate to call method
2. int result = objAdd.Invoke (3,6);
3. //short hand for above statement
4. int result = objAdd (3,6);
```

Plug-in Methods with Delegates

A method can be assigned to a delegate instance dynamically. This is useful for writing plug-in methods.In below example, The SquareData method has a delegate parameter, for specifying a plug-in SquareData.

```
1.  public delegate int delegateSquare(int x);
2. class Util
3. {
4.  public static void SquareData(int[] arr, delegateSquare obj)
5.  {
6.  for (int i = 0; i < arr.Length; i++)
7.  arr[i] = obj(arr[i]);
8.  }
9. }
10.   class demo
11.  {
12.   static void Main()
13.   {
14.   int[] arr = { 1, 2, 3 };
15.   Util.SquareData(arr, Square); // Dynamically hook in Square
16.   foreach (int i in arr)
17.   Console.Write(i + " "); // 1 4 9
18.   }
19.   static int Square(int x)
20.   {
21.   return x * x;
22.   }
23.  }
```

Types of Delegates

Single cast Delegate

A single cast delegate holds the reference of only single method. Above created delegate are single cast delegate.

Multi cast Delegate

A delegate which holds the reference of more than one method is called multi cast delegate. A multicast delegate only contains the reference of methods which return type is void. The + and += operators are used to combine delegate instances.

```
1.  MyDelegate d = Method1;
2. d = d + Method2;
3. //short hand for above statement
4. d += Method2;
```

Now, Invoking d will call both methods - Method1 and Method2. Methods are invoked in the order in which they are added.
The - and -= operators are used to remove a method from the delegate instances.

```
1.  d -= Method1;
```

Now, Invoking d will invoke only Method2.

Multicast delegate example

```
1.  delegate void Delegate_Multicast(int x, int y);
2. class demo
3. {
4.  static void Method1(int x, int y)
5.  {
6.  int z=x+y;
7.  Console.WriteLine("Method1 is called");
8.  Console.WriteLine("\n Sum is : {0}",z);
9.  }
10.   static void Method2(int x, int y)
11.   {
12.   int z=x+y;
13.   Console.WriteLine("\n Method2 is called");
14.   Console.WriteLine("\n Sum is : {0}",z);
15.   }
16.   public static void Main()
```

```
17.  Delegate_Multicast dmulti = Method1;
18.  dmulti += Method2;
19.  dmulti(1, 2); // Method1 and Method2 are called
20.  dmulti -= Method1;
21.  dmulti(2, 3); // Only Method2 is called
22.  }
23. }
24. Output:
25. Method1 is called
26. Sum is : 3
27. Method2 is called
28. Sum is : 5
```

Note

1. Delegates are immutable in nature, so when you call += or -= , a new delegate instance is created and it assign to the existing delegate instance.
2. All delegate are implicitly derived from System.MulticastDelegate, class which is inherit from System.Delegate class.
3. Delegate types are all incompatible with each other, even if their signatures are the same.

```
1.  delegate void D1();
2. delegate void D2();
3. ...
4. D1 d1 = Method1;
5. D2 d2 = d1; // Compile-time error
6. D2 d2 = new D2 (d1); // correct
```

4. Delegate instances are considered equal if they have the reference of same method.

```
1. delegate void D();
2. ...
3. D d1 = Method1;
4. D d2 = Method1;
5. Console.WriteLine (d1 == d2); // True
```

5. Multicast delegates are considered equal if they reference the same methods in the same order.
6. Delegates are used in event handling.

Source : http://www.dotnet-tricks.com/Tutorial/csharp/OITO200612-C-Sharp-Delegates-and-Plug-in-Methods-with-Delegates.html