UNDERSTANDING DELEGATES IN C#

A 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. It is very similar to the function pointer but with a difference that delegates are a type-safe. We can say that it is the object-oriented implementation of function pointers.

There are three steps for defining and using delegates:

1. Declaration

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

2. Instantiation

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

3. Invocation

Invoking a delegate is like as invoking a regular method.

```
//1. Declaration
public delegate int MyDelagate(int a, int b); //delegates having same signature as method

public class Example
{
 // methods to be assigned and called by delegate
 public int Sum(int a, int b)
 {
 return a + b;
  }

  public int Difference(int a, int b)
  {
  return a - b;
  }
  }
  class Program
```

```
{
static void Main()
{
Example obj = new Example();

// 2. Instantiation : As a single cast delegate
MyDelagate sum = new MyDelagate(obj.Sum);
MyDelagate diff = new MyDelagate(obj.Difference);

// 3.Invocation
Console.WriteLine("Sum of two integer is = " + sum(10, 20));
Console.WriteLine("Difference of two integer is = " + diff(20, 10));
}
}

/* Out Put

Sum of two integer is = 30
Difference of two integer is = 10
*/
```

Key points about delegates

1. Delegates are like C++ function pointers but are type safe.
2. Delegates allow methods to be passed as parameters.
3. Delegates are used in event handling for defining callback methods.
4. Delegates can be chained together i.e. these allow defining a set of methods that executed as a single unit.
5. Once a delegate is created, the method it is associated will never changes because delegates are immutable in nature.
6. Delegates provide a way to execute methods at run-time.
7. All delegates are implicitly derived from System.MulticastDelegate, class which is inheriting from System.Delegate class.
8. Delegate types are incompatible with each other, even if their signatures are the same. These are considered equal if they have the reference of same method.

Types of delegates

1. Single cast delegate

A single cast delegate holds the reference of only single method. In previous example, created delegate is a single cast delegate.

2. 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. Multicast delegates are considered equal if they reference the same methods in the same order.

```
//1. Declaration
public delegate void MyDelagate(int a, int b);
public class Example
{
 // methods to be assigned and called by delegate
 public void Sum(int a, int b)
 {
 Console.WriteLine("Sum of integers is = " + (a + b));
 }

  public void Difference(int a, int b)
  {
  Console.WriteLine("Difference of integer is = " + (a - b));
  }
 }
 class Program
 {
  static void Main()
  {
  Example obj = new Example();
  // 2. Instantiation
  MyDelagate multicastdel = new MyDelagate(obj.Sum);
  multicastdel += new MyDelagate(obj.Difference);

  // 3. Invocation
  multicastdel (50, 20);
  }
 }

 /* Out put

  Sum of integers is = 70
  Difference of integer is = 30

 */
```

Source : http://www.dotnet-tricks.com/Tutorial/csharp/QD39230314-Understanding-Delegates-in-C#.html