Delegates

A delegate is a type that represents references to methods with a particular parameter list and return type.

When we instantiate a delegate, we can associate its instance with any method with a compatible signature and return type.

We can invoke (or call) the method through the delegate instance.

Delegates have the following properties:

- Delegates are similar to C++ function pointers, but delegates are fully object-oriented, and unlike C++ pointers to member functions, delegates encapsulate both an object instance and a method.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.

Delegate types are derived from the Delegate class in the .NET Framework.

Delegate types are sealed—they cannot be derived from— and it is not possible to derive custom classes from Delegate.

Because the instantiated delegate is an object, it can be passed as a parameter, or assigned to a property.

This allows a method to accept a delegate as a parameter, and call the delegate at some later time. This is known as an asynchronous callback, and is a common method of notifying a caller when a long process has completed.

When a delegate is used in this fashion, the code using the delegate does not need any knowledge of the implementation of the method being used.

There are two types of delegates as

Single delegates

Single delegate can be used to invoke a single method.

Multicast delegates

Multicast delegate can be used to invoke the multiple methods.

The delegate instance can do multicasting (adding new method on existing delegate instance) using the + operator and – operator can be used to remove a method from a delegate instance.

All methods will invoke in sequence as they are assigned.

Application :

Application which demonstrate single delegate and multicast delegate.

```
using System;
delegate int fptr(int no1 , int no2);

delegate int mcast(int no1, int no2);

class Compare
{
   public int Max(int no1, int no2)
   {
      if(no1 > no2)
         return no1;
      else
         return no2;
   }

   public int Min(int no1, int no2)
   {
      if(no1 < no2)
         return no1;
      else
         return no2;
   }
}
   class Marvellous
   {
      static void Main(string[] args)
      {
         // Create single delegate
         Compare obj = new Compare();
```

```
        fptr obj1 = new fptr(obj.Max);
        Console.WriteLine("Maximum element is {0}",obj1(11,21));

        fptr obj2 = new fptr(obj.Min);
        Console.WriteLine("Minimum element is {0}",obj2(11,21));

        // Create multicast delegate
        mcast del = new mcast(obj.Max);

        del += new mcast(obj.Min);

        Console.WriteLine("Minimum element is {0}",del(11,21));

        del -= new mcast(obj.Min);

        Console.WriteLine("Maximum element is {0}",del(11,21));
    }
}
```