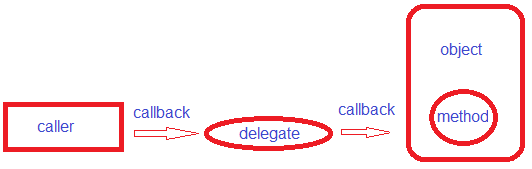
**What is .Net Delegates**

**Delegates**

The .NET Framework defines a special type (Delegate) that provides the functionality of a function pointer. A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature.

In the .NET environment, a delegate is a type that defines a method signature and it can pass a function as a parameter. In simple words we can say delegate is a .NET object which points to a method that matches its specific signature. A delegate is a form of type-safe function pointer used by the Common Language Infrastructure.



There are three steps for defining and using delegates:

### Declaration

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

### Instantiation

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

### Invocation

Invoking a delegate is like as invoking a regular method.

'1. Declaration

Public Delegate Function MyDelagate(a As Integer, b As Integer) As Integer

'delegates having same signature as method

Public Class Example

' methods to be assigned and called by delegate

Public Function Sum(a As Integer, b As Integer) As Integer

Return a + b

End Function

Public Function Difference(a As Integer, b As Integer) As Integer

Return a - b

End Function

End Class

Class Program

Private Shared Sub Main()

Dim obj As New Example()

' 2. Instantiation : As a single cast delegate

Dim sum As New MyDelagate(AddressOf obj.Sum)

Dim diff As New MyDelagate(AddressOf obj.Difference)

' 3.Invocation

Console.WriteLine("Sum of two integer is = " + sum(10, 20))

Console.WriteLine("Difference of two integer is = " + diff(20, 10))

End Sub

End Class

' 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**

### Single cast delegate

A single cast delegate holds the reference of only single method. In previous example, created delegate is a 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. Multicast delegates are considered equal if they reference the same methods in the same order.

'1. Declaration

Public Delegate Sub MyDelagate(a As Integer, b As Integer)

Public Class Example

' methods to be assigned and called by delegate

Public Sub Sum(a As Integer, b As Integer)

Console.WriteLine("Sum of integers is = " + (a + b))

End Sub

Public Sub Difference(a As Integer, b As Integer)

Console.WriteLine("Difference of integer is = " + (a - b))

End Sub

End Class

Class Program

Private Shared Sub Main()

Dim obj As New Example()

' 2. Instantiation

Dim A As MyDelagate = AddressOf obj.Sum

Dim B As MyDelagate = AddressOf obj.Difference

Dim C As MyDelagate = [delegate].MyDelagate.Combine(A, B)

' 3. Invocation

C(50, 20)

Console.ReadKey()

End Sub

End Class

' Out put

' Sum of integers is = 70

' Difference of integer is = 30

'