OOPs in VB.Net

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

Class Definition

A class definition starts with the keyword **Class** followed by the class name; and the class body, ended by the End Class statement. Following is the general form of a class definition −

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ]
[ Partial ] _
Class name [ ( Of typelist ) ]
   [ Inherits classname ]
   [ Implements interfacenames ]
   [ statements ]
End Class
```

Where,

- ***attributelist*** is a list of attributes that apply to the class. Optional.
- ***accessmodifier*** defines the access levels of the class, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- ***Shadows*** indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- ***MustInherit*** specifies that the class can be used only as a base class and that you cannot create an object directly from it, i.e., an abstract class. Optional.
- ***NotInheritable*** specifies that the class cannot be used as a base class.
- ***Partial*** indicates a partial definition of the class.
- ***Inherits*** specifies the base class it is inheriting from.
- ***Implements*** specifies the interfaces the class is inheriting from.

The following example demonstrates a Box class, with three data members, length, breadth and height −

```
Module mybox
```

```
  Class Box
    Public length As Double   ' Length of a box
    Public breadth As Double  ' Breadth of a box
    Public height As Double   ' Height of a box
  End Class
  Sub Main()
    Dim Box1 As Box = New Box()       ' Declare Box1 of type Box
    Dim Box2 As Box = New Box()       ' Declare Box2 of type Box
    Dim volume As Double = 0.0        ' Store the volume of a box here

    ' box 1 specification
    Box1.height = 5.0
    Box1.length = 6.0
    Box1.breadth = 7.0

    ' box 2 specification
    Box2.height = 10.0
    Box2.length = 12.0
    Box2.breadth = 13.0

    'volume of box 1
    volume = Box1.height * Box1.length * Box1.breadth
    Console.WriteLine("Volume of Box1 : {0}", volume)

    'volume of box 2
    volume = Box2.height * Box2.length * Box2.breadth
    Console.WriteLine("Volume of Box2 : {0}", volume)
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result –

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Member Functions and Encapsulation

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member and has access to all the members of a class for that object.

Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

Let us put above concepts to set and get the value of different class members in a class −

```
Module mybox
  Class Box
    Public length As Double    ' Length of a box
    Public breadth As Double   ' Breadth of a box
    Public height As Double    ' Height of a box
    Public Sub setLength(ByVal len As Double)
      length = len
    End Sub

    Public Sub setBreadth(ByVal bre As Double)
      breadth = bre
    End Sub

    Public Sub setHeight(ByVal hei As Double)
      height = hei
    End Sub

    Public Function getVolume() As Double
      Return length * breadth * height
    End Function
  End Class
  Sub Main()
    Dim Box1 As Box = New Box()       ' Declare Box1 of type Box
    Dim Box2 As Box = New Box()       ' Declare Box2 of type Box
    Dim volume As Double = 0.0        ' Store the volume of a box here

    ' box 1 specification
    Box1.setLength(6.0)
    Box1.setBreadth(7.0)
    Box1.setHeight(5.0)

    'box 2 specification
    Box2.setLength(12.0)
    Box2.setBreadth(13.0)
    Box2.setHeight(10.0)

    ' volume of box 1
```

```
    volume = Box1.getVolume()
    Console.WriteLine("Volume of Box1 : {0}", volume)

    'volume of box 2
    volume = Box2.getVolume()
    Console.WriteLine("Volume of Box2 : {0}", volume)
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result –

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Constructors and Destructors

A class **constructor** is a special member Sub of a class that is executed whenever we create new objects of that class. A constructor has the name **New** and it does not have any return type.

Following program explains the concept of constructor −

```
Class Line
  Private length As Double    ' Length of a line
  Public Sub New()   'constructor
    Console.WriteLine("Object is being created")
  End Sub

  Public Sub setLength(ByVal len As Double)
    length = len
  End Sub

  Public Function getLength() As Double
    Return length
  End Function
  Shared Sub Main()
    Dim line As Line = New Line()
    'set line length
    line.setLength(6.0)
    Console.WriteLine("Length of line : {0}", line.getLength())
    Console.ReadKey()
  End Sub
End Class
```

When the above code is compiled and executed, it produces the following result –

```
Object is being created
Length of line : 6
```

A default constructor does not have any parameter, but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**. This technique helps you to assign initial value to an object at the time of its creation as shown in the following example −

```
Class Line
   Private length As Double    ' Length of a line
   Public Sub New(ByVal len As Double)   'parameterised constructor
      Console.WriteLine("Object is being created, length = {0}", len)
      length = len
   End Sub
   Public Sub setLength(ByVal len As Double)
      length = len
   End Sub

   Public Function getLength() As Double
      Return length
   End Function
   Shared Sub Main()
      Dim line As Line = New Line(10.0)
      Console.WriteLine("Length of line set by constructor : {0}",
line.getLength())
      'set line length
      line.setLength(6.0)
      Console.WriteLine("Length of line set by setLength : {0}",
line.getLength())
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result –

```
Object is being created, length = 10
Length of line set by constructor : 10
Length of line set by setLength : 6
```

A **destructor** is a special member Sub of a class that is executed whenever an object of its class goes out of scope.

A **destructor** has the name **Finalize** and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories, etc.

Destructors cannot be inherited or overloaded.

Following example explains the concept of destructor −

```
Class Line
   Private length As Double    ' Length of a line
   Public Sub New()   'parameterised constructor
      Console.WriteLine("Object is being created")
   End Sub

   Protected Overrides Sub Finalize()  ' destructor
      Console.WriteLine("Object is being deleted")
   End Sub

   Public Sub setLength(ByVal len As Double)
      length = len
   End Sub

   Public Function getLength() As Double
      Return length
   End Function

   Shared Sub Main()
      Dim line As Line = New Line()
      'set line length
      line.setLength(6.0)
      Console.WriteLine("Length of line : {0}", line.getLength())
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result −

```
Object is being created
Length of line : 6
Object is being deleted
```

Shared Members of a VB.Net Class

We can define class members as static using the Shared keyword. When we declare a member of a class as Shared, it means no matter how many objects of the class are created, there is only one copy of the member.

The keyword **Shared** implies that only one instance of the member exists for a class. Shared variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it.

Shared variables can be initialized outside the member function or class definition. You can also initialize Shared variables inside the class definition.

You can also declare a member function as Shared. Such functions can access only Shared variables. The Shared functions exist even before the object is created.

The following example demonstrates the use of shared members −

```
Class StaticVar
   Public Shared num As Integer
   Public Sub count()
      num = num + 1
   End Sub
   Public Shared Function getNum() As Integer
      Return num
   End Function
   Shared Sub Main()
      Dim s As StaticVar = New StaticVar()
      s.count()
      s.count()
      s.count()
      Console.WriteLine("Value of variable num: {0}", StaticVar.getNum())
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result −

Value of variable num: 3

Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

Base & Derived Classes

A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.

The syntax used in VB.Net for creating derived classes is as follows −

<access-specifier> Class <base_class>
...
End Class
Class <derived_class>: Inherits <base_class>
...
End Class

Consider a base class Shape and its derived class Rectangle −

```
' Base class
Class Shape
   Protected width As Integer
   Protected height As Integer
   Public Sub setWidth(ByVal w As Integer)
      width = w
   End Sub
   Public Sub setHeight(ByVal h As Integer)
      height = h
   End Sub
End Class
' Derived class
Class Rectangle : Inherits Shape
   Public Function getArea() As Integer
      Return (width * height)
   End Function
End Class
Class RectangleTester
   Shared Sub Main()
      Dim rect As Rectangle = New Rectangle()
      rect.setWidth(5)
      rect.setHeight(7)
      ' Print the area of the object.
      Console.WriteLine("Total area: {0}", rect.getArea())
```

```
    Console.ReadKey()
  End Sub
End Class
```

When the above code is compiled and executed, it produces the following result –

```
Total area: 35

Base Class Initialization
```

The derived class inherits the base class member variables and member methods. Therefore, the super class object should be created before the subclass is created. The super class or the base class is implicitly known as **MyBase** in VB.Net

The following program demonstrates this −

Live Demo

```
' Base class
Class Rectangle
  Protected width As Double
  Protected length As Double
  Public Sub New(ByVal l As Double, ByVal w As Double)
    length = l
    width = w
  End Sub
  Public Function GetArea() As Double
    Return (width * length)
  End Function
  Public Overridable Sub Display()
    Console.WriteLine("Length: {0}", length)
    Console.WriteLine("Width: {0}", width)
    Console.WriteLine("Area: {0}", GetArea())
  End Sub
  'end class Rectangle
End Class

'Derived class
Class Tabletop : Inherits Rectangle
  Private cost As Double
  Public Sub New(ByVal l As Double, ByVal w As Double)
    MyBase.New(l, w)
  End Sub
  Public Function GetCost() As Double
    Dim cost As Double
```

```
    cost = GetArea() * 70
    Return cost
  End Function
  Public Overrides Sub Display()
    MyBase.Display()
    Console.WriteLine("Cost: {0}", GetCost())
  End Sub
  'end class Tabletop
End Class
Class RectangleTester
  Shared Sub Main()
    Dim t As Tabletop = New Tabletop(4.5, 7.5)
    t.Display()
    Console.ReadKey()
  End Sub
End Class
```

When the above code is compiled and executed, it produces the following result –

```
Length: 4.5
Width: 7.5
Area: 33.75
Cost: 2362.5
```