❖ Classes and Object

- ❖ A class is a group of different data members or objects with the same properties, processes, events of an object, and general relationships to other member functions.
- ❖ we can say that it is like a template or architect that tells what data and function will appear when it is included in a class object. For example, it represents the method and variable that will work on the object of the class.
- ❖ Objects are the basic run-time units of a class. Once a class is defined, we can create any number of objects related to the class to access the defined properties and methods. For example, the Car is the Class name, and the speed, mileage, and wheels are attributes of the Class that can be accessed by the Object.

Creating the Class

```
1. [ Access_Specifier ]  [ MustInherit | NotInheritable ]  Class ClassName
2. ' Data Members or Variable Declaration
3. ' Methods Name
4. ' Statement to be executed
5. End Class
```

Where,

- **Access_Specifier:** It defines the access levels of the class, such as Public, Private or Friend, Protected, Protected Friend, etc. to use the method. (It is an optional parameter).
- **MustInherit:** It is an optional parameter that specifies that the class can only be used as a base class, and the object will not directly access the base class or the abstract class.
- **NotInheritable:** It is also an optional parameter that representing the class not being used as a base class.
- **Implements:** It is used to specify interfaces from which the class inherits (optional).

My_program.vb

```
1. Public Class My_program
2. ' properties, method name, etc
3. ' Statement to be executed
4. End Class
```

In the above syntax, we have created a class with the name 'My_program' using the Class keyword.

The Syntax for creating an object

```
1. Dim Obj_Name As Class_Name = New Class_Name() ' Declaration of object
2. Obj_Name.Method_Name() ' Access a method using the object
```

In the above syntax, we have created an instance **(Obj_Name)** for the class **Class_Name**. By using the object name **'Obj_Name'** to access all the data members and the method name of **Class_Name**.

Let's create a program to find the Area and Parameter of a rectangle using the class and object in VB.NET.

My_program.vb

```
1. Imports System
2. Module My_program
3.     Sub Main()
4.         Dim rect As Rectangle = New Rectangle() 'create an object
5.         Dim rect2 As Rectangle = New Rectangle() 'create an object
6.         Dim area, para As Integer
7.
8.         'rect specification
9.         rect.setLength = (5)
10.        rect.setBreadth= (6)
11.
12.        'rect2 specification
13.        rect2.setLength = (5)
14.        rect2.setBreadth = (10)
15.
16.        'Area of rectangle
17.        area = rect.length * rect.Breadth
18.        'area = rect.GetArea()
19.        Console.WriteLine(" Area of Rectangle is {0}", area)
20.
21.        'Parameter of rectangle
22.        'para = rect.GetParameter()
23.         para = 2 (rect2.length + rect.Breadth)
24.        Console.WriteLine(" Parameter of Rectangle is {0}", para)
25.        Console.WriteLine(" Press any key to exit...")
26.        Console.ReadKey()
27.    End Sub
28.    Public Class Rectangle
29.        Public length As Integer
30.        Public Breadth As Integer
```

```
31.
32.       Public Sub setLength(ByVal len As Integer)
33.          length = len
34.       End Sub
35.
36.       Public Sub setBreadth(ByVal bre As Integer)
37.          Breadth = bre
38.       End Sub
39.       Public Function GetArea() As Integer
40.          Return length * Breadth
41.       End Function
42.
43.       Public Function GetParameter() As Integer
44.          Return 2 * (length + Breadth)
45.       End Function
46.    End Class
47. End Module
```

❖ Method Overloading

- In visual basic, **Method Overloading** means defining multiple methods with the same name but with different parameters.
- By using **Method Overloading**, we can perform different tasks with the same method name by passing different parameters.
- Suppose, if we want to overload a method in visual basic, we need to define another method with the same name but with different signatures.
- In visual basic, the Method Overloading is also called as **compile time polymorphism** or **early binding**.

Visual Basic Method Overloading Example

Following is the example of implementing a method overloading in a visual basic programming language.

```
Module Module1
  Public Class Calculate
    Public Sub AddNumbers(ByVal a As Integer, ByVal b As Integer)
      Console.WriteLine("a + b = {0}", a + b)
    End Sub
    Public Sub AddNumbers(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer)
      Console.WriteLine("a + b + c = {0}", a + b + c)
    End Sub
  End Class
```

```
  Sub Main(ByVal args As String())
    Dim c As Calculate = New Calculate()
    c.AddNumbers(1, 2)
    c.AddNumbers(1, 2, 3)
    Console.WriteLine("Press Enter Key to Exit..")
    Console.ReadLine()
  End Sub
End Module
```

❖ Constructor

In visual basic, **Constructor** is a method and it will invoke automatically whenever an instance of class or **struct** is created. The constructor is useful to initialize and set default values for the data members of the new object.

In case, if we create a class without having any constructor, the compiler will automatically create a one default constructor for that class. So, there is always one constructor that will exist in every class.

In visual basic, a class can contain more than one constructor with a different type of arguments and the constructors will never return anything, so we don't need to use any return type, not even **void** while defining the constructor method in the class.

Constructor Syntax

As discussed, the constructor is a method and it won't contain any return type. If we want to create a constructor in visual basic, we need to create a method with `New` keyword.

Following is the syntax of creating a constructor in visual basic programming language.

In visual basic, **Constructor** is a method and it will invoke automatically whenever an instance of class or **struct** is created. The constructor is useful to initialize and set default values for the data members of the new object.

In case, if we create a class without having any constructor, the compiler will automatically create a one default constructor for that class. So, there is always one constructor that will exist in every class.

In visual basic, a class can contain more than one constructor with a different type of arguments and the constructors will never return anything, so we don't need to use any return type, not even **void** while defining the constructor method in the class.

Following is the syntax of creating a constructor in visual basic programming language.

```
Public Class User
  ' Constructor
  Public Sub New()
    ' Your Custom Code
  End Sub
End Class
```

If you observe the above syntax, we created a class called "**User**" and a method with `New` keyword. Here the method **New()** will become a constructor of our class.

Constructor Types

In visual basic, we have a different type of constructors available, those are

- Default Constructor
- Parameterized Constructor
- Copy Constructor
- Private Constructor

Now, we will learn about each constructor in detail with examples in a visual basic programming language.

Default Constructor

In visual basic, if we create a constructor without having any parameters, we will call it as **default constructor** and the instance of the class will be initialized without any parameter values.

Following is the example of defining the default constructor in visual basic programming language.

```
Module Module1
  Class User
    Public name, location As String
    ' Default Constructor
    Public Sub New()
      name = "Suresh Dasari"
      location = "Hyderabad"
    End Sub
  End Class
  Sub Main()
    ' The constructor will be called automatically once the instance of the class created
    Dim user As User = New User()
    Console.WriteLine(user.name)
    Console.WriteLine(user.location)
    Console.WriteLine("Press Enter Key to Exit..")
    Console.ReadLine()
  End Sub
End Module
```

Parameterized Constructor

In visual basic, if we create a constructor with at least one parameter, we will call it a **parameterized constructor** and every time the instance of the class must be initialized with parameter values.

Following is the example of defining the parameterized constructor in a visual basic programming language.

```
Module Module1
```

```
  Class User
    Public name, location As String
    ' Parameterized Constructor
    Public Sub New(ByVal a As String, ByVal b As String)
      name = a
      location = b
    End Sub
  End Class
  Sub Main()
    ' The constructor will be called automatically once the instance of the class created
    Dim user As User = New User("Suresh Dasari", "Hyderabad")
    Console.WriteLine(user.name)
    Console.WriteLine(user.location)
    Console.WriteLine("Press Enter Key to Exit..")
    Console.ReadLine()
  End Sub
End Module
```

Constructor Overloading

In visual basic, we can **overload** the constructor by creating another constructor with the same method name but with different parameters.

Following is the example of implementing a constructor overloading in a visual basic programming language.

```
Module Module1
  Class User
    Public name, location As String
    ' Default Constructor
    Public Sub New()
      name = "Suresh Dasari"
      location = "Hyderabad"
    End Sub
    ' Parameterized Constructor
    Public Sub New(ByVal a As String, ByVal b As String)
      name = a
      location = b
    End Sub
  End Class
  Sub Main()
    ' Default Constructor will be called
    Dim user As User = New User()
    ' Parameterized Constructor will be called
    Dim user1 As User = New User("Rohini Alavala", "Guntur")
    Console.WriteLine(user.name & ", " & user.location)
    Console.WriteLine(user1.name & ", " & user1.location)
    Console.WriteLine(vbLf & "Press Enter Key to Exit..")
```

```
    Console.ReadLine()
  End Sub
End Module
```

❖ Destructor

n visual basic, **Destructor** is a special method of a class and it is useful in class to destroy the object or instances of classes. The destructor in visual basic will invoke automatically whenever the class instances become unreachable.

The following are the properties of destructor in a visual basic programming language.

- In visual basic, destructors can be used only in classes and a class can contain only one destructor.
- The destructor in class can be represented by using **Finalize()** method.
- The destructor in visual basic won't accept any parameters and access modifiers.
- The destructor will invoke automatically, whenever an instance of a class is no longer needed.
- The destructor automatically invoked by garbage collector whenever the class objects that are no longer needed in an application.

Destructor Syntax

Following is the syntax of defining the destructor in visual basic programming language.

```
Class User
  ' Destructor
  Protected Overrides Sub Finalize()
    ' Your Code
  End Sub
End Class
```

If you observe the above syntax, we created a destructor using **Finalize()** method.

Destructor Example

Following is the example of using destructor in a visual basic programming language to destruct the unused objects of the class.

```
Module Module1
  Class User
    Public Sub New()
      Console.WriteLine("An Instance of class created")
    End Sub
    Protected Overrides Sub Finalize()
      Console.WriteLine("An Instance of class destroyed")
    End Sub
  End Class
  Sub Main()
```

```
        Details()
        GC.Collect()
        Console.ReadLine()
    End Sub
    Public Sub Details()
        Dim user As User = New User()
    End Sub
End Module
```

If you observe the above example, we created a class with default constructor and **destructor**. Here, we created an instance of class "**User**" in **Details()** method and whenever the **Details** function execution is done, then the garbage collector (**GC**) automatically will invoke the destructor in **User** class to clear the object of class.

❖ Inheritance

In visual basic, **Inheritance** is one of the primary concepts of object-oriented programming (OOP) and it is useful to inherit the properties from one class (base) to another (child) class.

The inheritance will enable us to create a new class by inheriting the properties from other classes to reuse, extend and modify the behavior of other class members based on our requirements.

In visual basic inheritance, the class whose members are inherited is called a **base** (**parent**) class and the class that inherits the members of **base** (**parent**) class is called a **derived** (**child**) class.

Inheritance Syntax

Following is the syntax of implementing an inheritance to define derived class that inherits the properties of base class in a visual basic programming language.

```
<access_modifier> Class <base_class_name>
    // Base class Implementation
End Class

<access_modifier> Class <derived_class_name>
Inherits base_class_name
    // Derived class implementation
End Class
```

If you observe the above syntax, we are inheriting the properties of **base** class into **child** class to improve the code reusability.

Following is the simple example of implementing inheritance in a visual basic programming language.

```
Public Class X
    Public Sub GetDetails()
        ' Method implementation
    End Sub
End Class

Public Class Y
    Inherits X
```

```
    ' your class implementation
End Class

Class Program
    Public Shared Sub Main(ByVal args As String())
        Dim y As Y = New Y()
        y.GetDetails()
    End Sub
End Class
```

If you observe the above example, we defined a class "**X**" with a method called "**GetDetails**" and the class "**Y**" is inheriting from the class "**X**". After that, we are calling a "**GetDetails**" method by using an instance of derived class "**Y**".

In visual basic, it's not possible to inherit the base class constructors in the derived class and the accessibility of other members of the base class also depends on the access modifiers which we used to define those members in the base class.

Inheritance Example

Following is the example of implementing an **inheritance** by defining two classes in a visual basic programming language.

```
Public Class User
    Public Name As String
    Private Location As String
    Public Sub New()
        Console.WriteLine("Base Class Constructor")
    End Sub
    Public Sub GetUserInfo(ByVal loc As String)
        Location = loc
        Console.WriteLine("Name: {0}", Name)
        Console.WriteLine("Location: {0}", Location)
    End Sub
End Class

Public Class Details
    Inherits User
    Public Age As Integer
    Public Sub New()
        Console.WriteLine("Child Class Constructor")
    End Sub
    Public Sub GetAge()
        Console.WriteLine("Age: {0}", Age)
    End Sub
End Class

Class Program
    Public Shared Sub Main(ByVal args As String())
```

```
        Dim d As Details = New Details()
        d.Name = "Suresh Dasari"
        ' Compile Time Error
        ' d.Location = "Hyderabad";
        d.Age = 32
        d.GetUserInfo("Hyderabad")
        d.GetAge()
        Console.WriteLine(vbLf & "Press Any Key to Exit..")
        Console.ReadLine()
    End Sub
End Class
```

Multi-Level Inheritance

Generally, visual basic supports only **single inheritance** that means a class can only inherit from one base class. However, in visual basic the inheritance is transitive and it allows you to define a hierarchical inheritance for a set of types and it is called a multi-level inheritance.

For example, suppose if class **C** is derived from class **B**, and class **B** is derived from class **A**, then class **C** will inherit the members declared in both class **B** and class **A**.

```
Public Class A
    ' Implementation
End Class

Public Class B
    Inherits A
    ' Implementation
End Class

Public Class C
    Inherits B
    ' Implementation
End Class
```

Example:2 ……

```
Public Class A
    Public Name As String
    Public Sub GetName()
        Console.WriteLine("Name: {0}", Name)
    End Sub
End Class

Public Class B
    Inherits A
    Public Location As String
    Public Sub GetLocation()
```

```
    Console.WriteLine("Location: {0}", Location)
  End Sub
End Class

Public Class C
  Inherits B
  Public Age As Integer
  Public Sub GetAge()
    Console.WriteLine("Age: {0}", Age)
  End Sub
End Class

ClassProgram
  Public Shared Sub Main(ByVal args As String())
    Dim c As C = New C()
    c.Name = "Suresh Dasari"
    c.Location = "Hyderabad"
    c.Age = 32
    c.GetName()
    c.GetLocation()
    c.GetAge()
    Console.WriteLine(vbLf & "Press Any Key to Exit..")
    Console.ReadLine()
  End Sub
End Class
```

❖ Method Overriding

In visual basic, **Method Overriding** means override a base class method in the derived class by creating a method with the same name and signatures to perform a different task. The Method Overriding in visual basic can be achieved by using **Overridable** & **Overrides** keywords along with the inheritance principle.

Suppose, if we want to change the behavior of the base class method in a derived class, we need to use **method overriding**. The base class method which we want to override in the derived class that needs to be defined with an **Overridable** keyword and we need to use **Overrides** keyword in derived class while defining the method with the same name and parameters then only we can override the base class method in the derived class.

In visual basic, the Method Overriding is also called as **run time polymorphism** or **late binding**. Following is the code snippet of implementing a **method overriding** in a visual basic programming language.

```
' Base Class
Public Class Users
  Public Overridable Sub GetInfo()
    Console.WriteLine("Base Class")
  End Sub
End Class
```

```
'Derived Class
Public Class Details
  Inherits Users
  Public Overrides Sub GetInfo()
    Console.WriteLine("Derived Class")
  End Sub
End Class
```

If you observe the above code snippet, we created two classes ("**Users**", "**Details**") and the derived class (**Details**) is inheriting the properties from the base class (**Users**) and we are overriding the base class method **GetInfo** in the derived class by creating a method with same name and parameters, this is called a **method overriding** in visual basic.

Here, we defined the **GetInfo** method with an **Overridable** keyword in the base class to allow derived class to override that method using the **Overrides** keyword.

As discussed, only the methods with **Overridable** keyword in the base class are allowed to override in the derived class using **Overrides** keyword.

Method Overriding Example

Following is the example of implementing a method overriding in a visual basic programming language.

```
Module Module1
  Public Class BClass
    Public Overridable Sub GetInfo()
      Console.WriteLine("Learn C# Tutorial")
    End Sub
  End Class
  Public Class DClass
    Inherits BClass
    Public Overrides Sub GetInfo()
      Console.WriteLine("Welcome ")
    End Sub
  End Class
  Sub Main(ByVal args As String())
    Dim d As DClass = New DClass()
    d.GetInfo()
    Dim b As BClass = New BClass()
    b.GetInfo()
    Console.WriteLine("Press Enter Key to Exit..")
    Console.ReadLine()
  End Sub
End Module
```

S.NO	METHOD OVERLOADING	METHOD OVERRIDING
1.	Method overloading is a compile time polymorphism.	Method overriding is a run time polymorphism.
2.	It help to rise the readability of the program.	While it is used to grant the specific implementation of the method which is already provided by its parent class or super class.
3.	It is occur within the class.	While it is performed in two classes with inheritance relationship.
4.	Method overloading may or may not require inheritance.	While method overriding always needs inheritance.
5.	In this, methods must have same name and different signature.	While in this, methods must have same name and same signature.
6.	In method overloading, return type can or can not be be same, but we must have to change the parameter.	While in this, return type must be same or co-variant.