Object Oriented Concept

- All languages those supports the concepts of OOP they are considered as **OOP programming language.**
- All OOP based languages supports the following concepts.
  - Class and Object
  - Abstraction and Encapsulation
  - Constructor and Destructor
  - Inheritance
  - Polymorphism

- **Class**
  - Class is a group of **methods** and **variables** where methods are known as **Member function** and variables are known as **Data Member**.
  - In vb.net class is created using **Class Keyword** and completed with **End Class**.
  - Once class is created you can created any number of objects.

Syntax:

```
[accessmodifier][Shadows][MustInherit|NotInheritable]
Class <Classname>
  [Inherits classname]
  [Implements interfacenames]
        [statements]
End Class
```

where

- **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.
- **Inherits** specifies the base class it is inheriting from.
- **Implements** specify the interfaces the class is inheriting from.

Example

- Add new class file to the project and give proper name (Class1.vb). The extension of the class is **.vb**

- Write down the following code in Class1.vb File

```
Public Class Class1 'class name
    Dim a, b As Integer
    'Class Method
    Sub getdata(ByVal i As Integer, ByVal j As Integer)
        a = i
        b = j
    End Sub
    Sub disp() 'class method
        MsgBox("a==" & a & "b==" & b)
    End Sub
    Function sum() As Integer 'function
        Return a + b
    End Function
End Class
```

- Add windows form name frmclass.vb and write down the following code in btnclass click event.

```
Private Sub btnclass_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnclass.Click
        Dim obj As New Class1 'object is created
        obj.getdata(10, 20) 'method is called
        obj.disp()
        MsgBox("the sum of two no is " & obj.sum())
    End Sub
```

Object

- It is runtime entity.
- It is member of class which is able to access methods and variables of class which has been declared in Public Mode.
- In vb.net ,it can be created as follows:

Syntax:

<object> As New <Class Name>

- Example:

```
Private Sub btnclass_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnclass.Click
    Dim obj As New Class1 'object is created
    obj.getdata(10, 20) 'method is called
    obj.disp()
    MsgBox("the sum of two no is " & obj.sum())
End Sub
```

- **Abstraction**
  - It refers to the act of representing essentials features without including the background details or explanations.
  - It defines way to abstract or hide your data and members from outside class.
  - Classes use the concept of Abstraction.
  - When we define a classes then different accessibility mode define the access level of variables and methods. They are Public, Private, Protected.

Example:

```
Public Class Class1 'class name
  Dim a, b As Integer
  'Class Method
  Sub getdata(ByVal i As Integer, ByVal j As Integer)
    a = i
    b = j
  End Sub
  Sub disp() 'class method
    MsgBox("a==" & a & "b==" & b)
  End Sub
  Function sum() As Integer 'function
    Return a + b
  End Function
End Class
```

Encapsulation

- It is a process to bind data and methods in a unit called class. When we define class that process itself define the concept of Encapsulation.
- It can protect your data from accidental corruption.
- Rather than defining the data in the form of public, we can declare those fields as private.

- **Example**

```
Public Class Class1 'class name
  Dim a, b As Integer
  'Class Method
  Sub getdata(ByVal i As Integer, ByVal j As Integer)
    a = i
    b = j
  End Sub
  Sub disp() 'class method
    MsgBox("a==" & a & "b==" & b)
  End Sub
  Function sum() As Integer 'function
    Return a + b
  End Function
End Class
```

- **Constructor**
  - It is a special member function which is used to initialize the object of its class.
  - It is automatically called when object is created.
  - In vb.net, The name of constructor is NEW()
  - It can be overloaded.
  - There are 2 types of constructor such as constructor with arguments and without arguments.
- **Example**

Add Class file and give name ClsConstuctor

```
Public Class ClsConstuctor
  Dim i As Integer
  Dim n As Integer
  'Constructor without arguments
  Sub New()
    MsgBox("I am Default Constuctor")
  End Sub
  'Constructor with arguments
  Sub New(ByVal i As Integer)
    n = i
    MsgBox("I am paramiterized Constuctor")
    MsgBox(n)
  End Sub
  Protected Overrides Sub finalize()
    n = 0
```

```
        MsgBox("I m Desctuctor")
    End Sub
End Class
```

Add windows form and write down the following code in btnconstructor click event.

```
Private Sub btnConstructor_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnConstructor.Click
    Dim x As New ClsConstuctor
    Dim y As New ClsConstuctor(10)
End Sub
```

- **How to call Base Class Constructor**
  - It can be called using **MyBase** keyword.
  - It cannot be used to access private members of class.
  - It refers to the immediate base class and its inherited members
- **Example:**

```
Public Class ClsBase
    Private x, y As Integer
    Sub New()
        x = 0
        y = 0
    End Sub
    Sub New (ByVal a As Integer, ByVal b As Integer)
        x = a
        y = b
    End Sub
End Class

Public Class clsderive
    Inherits ClsBase
    Private z As Integer
    Sub New()
        MyBase.New()
        z = 0
    End Sub
    Sub New(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer)
        MyBase.New(a, b)
        z = c
        MsgBox(a & b & c)
    End Sub
End Class
```

Add windows form and write down the following code in **btnconstructor** click event.

```
Private Sub btnConstructor_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnConstructor.Click
Dim obj As New clsderive
Dim obj1 As New clsderive(4, 6, 7)
 End Sub
```

- **Destructor**
  - It is a special function which is called automatically when a class is destroyed.
  - A destructor is also known as finalizer. It is the last method run by class
  - Within a destructor we can place code to clean up the object after it is used ,which might include decrement counters releasing resources
  - It cannot be overloaded.
- Example:

```
Protected Overrides Sub finalize()
    n = 0
    MsgBox("I m Desctuctor")
End Sub
```

- **Inheritance**
  - It is a process of creating a new class called derives class from the existing class called Base Class.
  - The existing class is known as parent, base, super class.
  - The new class is known as child, derive, sub class.
  - It is also known as code reusability.
  - It can be created by using inherits keyword.
  - A derive class obtain all of the methods, properties, and events of the base class.
  - There are 3 modifier related with inheritance.
    - Inherits
    - NotInheritable
    - MustInherit(Abstract Class)
- **Syntax**

```
Class <Class Name>
    [inherits] Base Class Name
    Statements
End Class
```

- **Example**

  Add class file name clsbase and write down the following code

```
Public Class ClsBase 'Base Class
   Protected a As Integer = 40
   Sub disp()
      MsgBox("I am Base Class")
   End Sub
End Class

Public Class clsderive 'Derive Class
   Inherits ClsBase
   Dim b As Integer = 20
   Sub show()
      MsgBox("value from Base Class=" & a)
      MsgBox("value from Derive Class=" & b)
      MsgBox("I am Derived Class Method")
   End Sub
End Class
```

Add windows form and write down the following code in btninheritance click event

```
Private Sub btninheritance_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btninheritance.Click
      Dim x As New clsderive
      x.disp() 'Base Class Method
      x.show() 'Base + derive Class'
End Sub
```

- **NotInheritable**
  - It is known as seal class. If we write NotInheritable keyword in class then no other class can be derived from this class.
  - It gives surety that your class is not become parent of any class.
- **Example**

  Create class clsA as NotInheritable and try to derive it in clsB

```
Public NotInheritable Class clsA

End Class

Public Class ClsB
   Inherits clsA 'ClsB cannot inherite from class ClsA because 'ClsA is declared as
NotInheritable

End Class
```

- **MustInherit (Abstract Class)**
  - MustInherit Keyword is used to create Abstract Class.
  - It specifies that class is intended as a base class.
  - It provides Skelton to the derived class. it has no actual code.
  - It is not used to create an object.
  - If our class contains at least one MustOverride method then our class should be declare as MustInherit class.
  - Methods written in Abstract class as MustOverride are automatically appear in the derived class.
  - Mustoverride methods must be declared in MustInherite Class.
  - No other statements are allowed and especially there is No End Sub or End Function.
- Example

Add Class file and give name clsAbs1

```
Public MustInherit Class clsAbs1
   MustOverride Sub add(ByVal a As Integer, ByVal b As Integer)
   MustOverride Sub div(ByVal a As Integer, ByVal b As Integer)
End Class
```

Add another Class file and give name clstemp

```
Public Class clstemp
   Inherits clsAbs1

   Public Overrides Sub add(ByVal a As Integer, ByVal b As Integer)
            MsgBox(a + b)

   End Sub

   Public Overrides Sub div(ByVal a As Integer, ByVal b As Integer)
               MsgBox(a / b)

   End Sub
End Class
```

Add windows form and write down the following code in btnabstactclass_Click event

```
Private Sub btnabstactclass_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnabstactclass.Click
    Dim obj As New clstemp
```

```
        obj.add(10, 20)
        obj.div(20, 10)
    End Sub
```

- **Polymorphism**

  - It means "one name, multiple forms
  - There are two types of it **(1) Design time (2)Run time**
  - **Design time** polymorphism is achieved by **function overloading**
  - **Run time** polymorphism is achieved by **function overriding**
  - Function overloading means we can declare more than one functions with the **same name but arguments are different.**

Example

Add Class file and write this code (it is the example of function overloading)

```
Public Class Clsmath

  Function add(ByVal a As Integer)
    Return MsgBox(a)
  End Function

  Function add(ByVal a As Integer, ByVal b As Integer)
    Return MsgBox(a + b)
  End Function

  Function add(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer)
    Return MsgBox(a + b + c)
  End Function
End Class
```

Write this code on btnpoly_click event

```
Private Sub btnpoly_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnpoly.Click
    Dim obj As New Clsmath
    obj.add(5)
    obj.add(5, 5)
    obj.add(10, 20, 30)
End Sub
```

Run time polymorphism (function overriding)

- Function overriding means when we declare more than one function with same name **but in different class** and the relationship between class is **parent -child**
- The overriding concept achieve only in inheritance.
- Derive class inherits methods from its base class
- If an inherited property or method needs to behave differently in the derived class.
- **Overridable keyword** is used to mark a function as override. We can redefine overridable methods in derived class
- **The overrides keyword** is used to mark that a function is overriding some base class function.
- The **overrides** keyword overrides an overridable property or method defined in the base class

Example

```
Public Class Clsoverridable
  Public Class base
    Overridable Sub disp()
      MsgBox("Base class")
    End Sub
  End Class
  Public Class clsderived
    Inherits base
    Public Overrides Sub disp()
      MyBase.disp()
      MsgBox("Derived Class")
    End Sub
  End Class
End Class
```

Write this code in btnoverriding_click Event

```
Private Sub btnoverriding_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnoverriding.Click
    Dim obj As New Clsoverridable.clsderived
    obj.disp()
  End Sub
```

- **Interface**

Why interface used?
We cannot inherit more than one class in vb.net. If we wants to inherit then? Use interface
Interface

- Interfaces like classes define a set of properties, methods and events. But unlike classes, an interface does not contain any implementation code
- Interface represents **"Has a" relationship**
- Classes which implements interface they write the coding in the interface's method
- We can implements multiple interface
- If we want to use interface then we need to write" **Implements**" keyword

Syntax

```
Public Interface iface1
        ….logic
End Interface
```

Example

Add interface in your project

```
Public Interface Interface1
  Sub hi()
End Interface
```

Add another interface in your project

```
Public Interface Interface2
  Sub hello()
End Interface
```

Add Class File in your project

```
Public Class ClssInterface
  Implements Interface1, Interface2

  Public Sub hi() Implements Interface1.hi
    MsgBox("Hi i m interface 1")
  End Sub

  Public Sub hello() Implements Interface2.hello
    MsgBox("Hello i m interface 2")
  End Sub
End Class
```

Write down this code in btninterface_click Event

```
Private Sub btninterface_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btninterface.Click
    Dim obj As New ClssInterface
    obj.hi()
    obj.hello()
  End Sub
```