

# Unit-II

# Working with Arrays and Strings

- ▶ A standard structure for storing data in any programming language is the array.
- ▶ Whereas individual variables can hold single entities, such as one number, one date, or one string, *arrays can hold sets of* data of the same type (a set of numbers, a series of dates, and so on).
- An array has a name, as does a variable, and the values stored in it can be accessed by an index.
- ▶ An array is similar to a variable: it has a name and multiple values. Each value is identified by an index (an integer value) that follows the array's name in parentheses.
- ▶ Each different value is an *element of the array*.

## ▶ Declaring Arrays

- ▶ Unlike simple variables, arrays must be declared with the Dim (or Public, or Private) statement followed by the name of the array and the index of the last element in the array in parentheses—for example,

```
Dim Salaries(15) As Integer
```

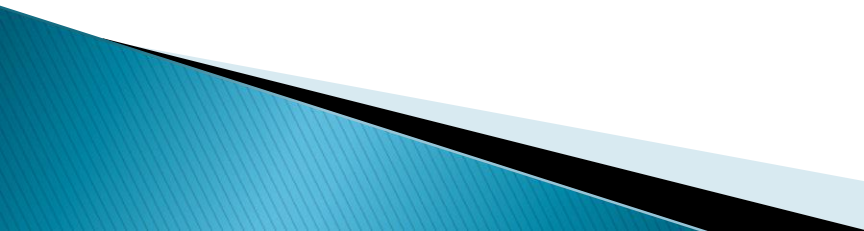
```
Dim Names(15) As String
```

## ▶ Initializing Arrays

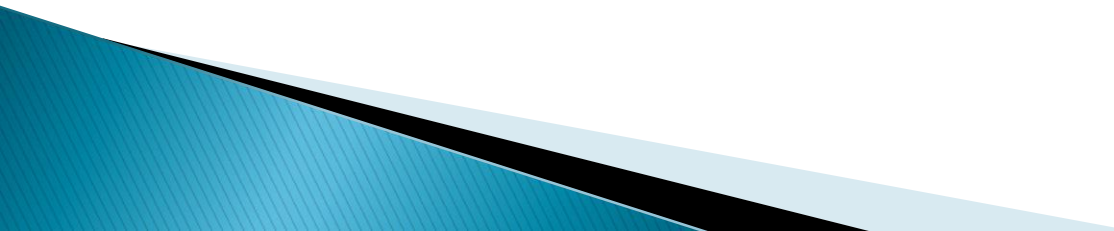
Just as you can initialize variables in the same line where you declare them, you can initialize arrays, too, with the following constructor:

```
Dim names() As String = {"Joe Doe", "Peter Smack"}
```

# IF ELSE in VB.NET

- ▶ The conditional statement **IF ELSE** , is use for examining the conditions that we provided, and making decision based on that contition.
  - ▶ The conditional statement examining the data using comparison operators as well as logical operators.
  - ▶ **If [your condition here]**
  - ▶ **Your code here**
  - ▶ **Else Your code Here**
  - ▶ **End If**
  - ▶ If the contition is TRUE then the control goes to between IF and Else block , that is the program will execute the code between IF and ELSE statements.
  - ▶ If the contition is FLASE then the control goes to between ELSE and END IF block , that is the program will execute the code between ELSE and END IF statements.
- 

# Syntax:–

- ▶ If you want o check more than one condition at the same time , you can use Elself .
  - ▶ If [your condition here]
  - ▶     Your code here
  - ▶ Elself
  - ▶     [your condition here]
  - ▶ Elself [your condition here]
  - ▶ Your code here
  - ▶ Else Your code Here
  - ▶ End If
- 

# FOR NEXT loop in vb.net

- ▶ Whenever you face a situation in programming to repeat a task for several times (more than one times ) or you have to repeat a task till you reach a condition.
- ▶ in these situations you can use loop statements to achieve your desired results.
- ▶ This kind of for loop is useful for iterating over arrays and for other applications in which you know in advance how many times you want the loop to iterate.
- ▶ The **FOR NEXT** Loop , execute the loop body (the source code within For ..Next code block) to a fixed number of times.
- ▶ **For var=[startValue] To [endValue] [Step]**
- ▶ **[loopBody]**
- ▶ **Next [var]**

# Example

- ▶ Module Module1
- ▶ Sub Main()
- ▶     ' This loop goes from 0 to 5.
- ▶     For value As Integer = 0 To 5
- ▶         ' Exit condition if the value is three.
- ▶         If (value = 3) Then
- ▶             Exit For
- ▶         End If
- ▶         Console.WriteLine(value)
- ▶     Next
- ▶ End Sub
- ▶ End Module
  
- ▶ Output
  
- ▶ 0
- ▶ 1
- ▶ 2

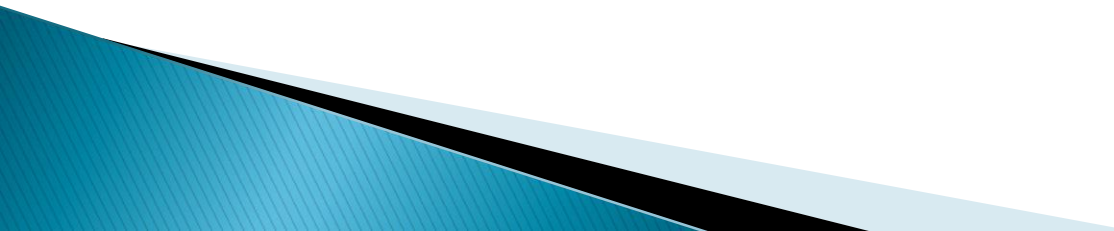
# FOR EACH loop in VB.NET

- ▶ Whenever you face a situation in programming to repeat a task for several times (more than one times ) or you have to repeat a task till you reach a condition, in these situations you can use loop statements to achieve your desired results.
- ▶ FOR NEXT Loop, FOR EACH Loop , WHILE Loop and DO WHILE Loop are the Commonly used loops in Visual Basic.NET.
- ▶ For Each Loop
- ▶ FOR EACH Loop usually using when you are in a situation to execute every single element or item in a group, in these type of situation you can use For Each loop.
- ▶ For Each [Item] In [Group]
- ▶       [loopBody]
- ▶ Next [Item]



# Example

```
Public Class Form1
    Private Sub
        Button1_Click(ByVal sender As System.Object,
            ByVal e As System.EventArgs) Handles
                Button1.Click
            Dim siteName As String
            Dim singleChar As Char
            siteName =
                "DEPARTMENT OF COMPUTER SCIENCE"
            For Each singleChar In siteName
                MsgBox(singleChar)
            Next
        End Sub
    End Class
```



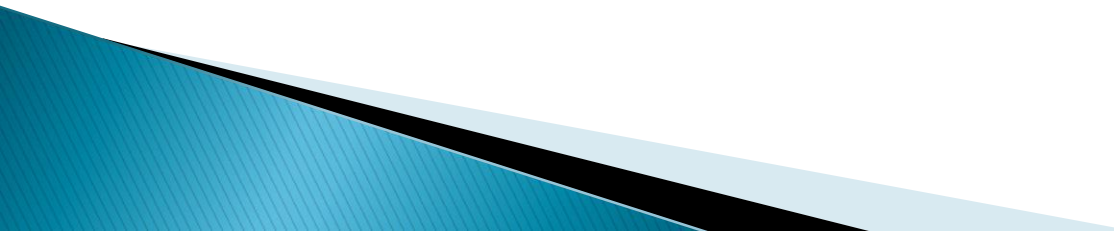
# Example:–

- ▶ Public Class Form1
- ▶ Private Sub Button1\_Click(ByVal sender As System.Object,
- ▶     ByVal e As System.EventArgs) Handles Button1.Click
- ▶     Dim siteName As String
- ▶     Dim singleChar As Char
- ▶     siteName = "Department of computer science"
- ▶     For Each singleChar In siteName
- ▶         MsgBox(singleChar)
- ▶     Next
- ▶ End Sub
- ▶ End Class

# While End While loop

- ▶ Whenever you face a situation in programming to repeat a task for several times (more than one times ) or you have to repeat a task till you reach a condition, in these situations you can use loop statements to achieve your desired results.
- ▶ While ..End While
- ▶ While .. End While Loop execute the code body (the source code within While and End while statements ) until it meets the specified condition.
- ▶ The expression is evaluated each time the loop is encountered.
- ▶ If the evaluation result is true, the loop body statements are executed.
- ▶ While [condition]
- ▶       [loop body]
- ▶ End While

# Enum in vb.net

- ▶ When you are in a situation to have a number of constants that are logically related to each other, you can define them together these constants in an enumerator list.
  - ▶ An enumerated type is declared using the enum keyword.  
Syntax:
  - ▶ An enumeration has a name, an underlying data type, and a set of members.
  - ▶ Each member represents a constant.
  - ▶ It is useful when you have a set of values that are functionally significant and fixed.
  - ▶ Retrieve and check the Enum value.
- 

# Example

- ▶ How to get int value from enum

Enum Days

Sunday = 1

Tuesday = 2

wednesday = 3

End Enum

Private day As Integer = CInt(Days.Tuesday)

# VB.Net – Select Case Statement

- ▶ A **Select Case** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.
- ▶ Syntax:
- ▶ The syntax for a Select Case statement in VB.Net is as follows:  
Select [ Case ] expression  
[ Case expressionlist  
    [ statements ] ]  
[ Case Else  
    [ elsestatements ] ]  
End Select

# Example

```
Dim grade As Char
```

```
grade = "B"
```

```
Select grade
```

```
Case "A"
```

```
    msgbox("Excellent!")
```

```
Case "B", "C"
```

```
    msgbox("Well done")
```

```
Case "D"
```

```
    msgbox("You passed")
```

```
Case "F"
```

```
    msgbox("Better try again")
```

```
Case Else
```

```
    msgbox("Invalid grade")
```

```
End Select
```



# Function and Procedure

- ▶ A procedure is a group of statements that together perform a task when called.
- ▶ After the procedure is executed, the control returns to the statement calling the procedure.
- ▶ VB.NET has two types of procedures:
  - 1) Functions
  - 2) Sub procedures or Subs
- ▶ Function returns a value, whereas Subs do not return a value.
- ▶ Defining a Function

The Function statement is used to declare the name, parameter and the body of a function.



# Continue...

- ▶ The syntax for the Function statement is:

[Modifiers] Function FunctionName [(ParameterList)] As ReturnType  
[Statements]

End Function

Where,

- Modifiers: specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected.
- FunctionName: indicates the name of the function
- ParameterList: specifies the list of the parameters
- ReturnType: specifies the data type of the variable the function returns

# Example

- ▶ Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
- ▶ ' local variable declaration \*/
- ▶ Dim result As Integer
- ▶ If (num1 > num2) Then
- ▶     result = num1
- ▶ Else
- ▶     result = num2
- ▶ End If
- ▶ FindMax = result
- ▶ End Function

# Function Returning a Value

In VB.Net, a function can return a value to the calling code in two ways:

- ▶ By using the return statement
- ▶ By assigning the value to the function name

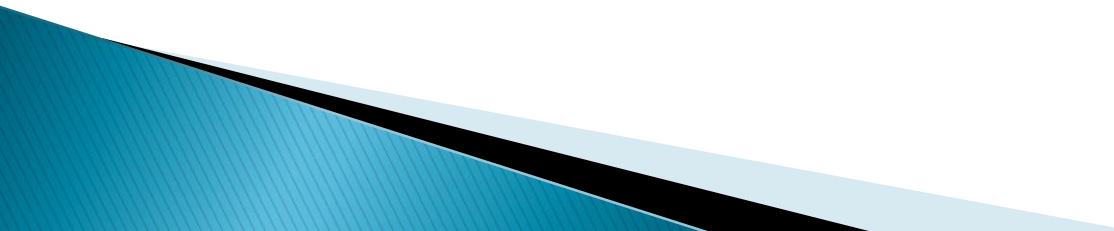
# Example

- ▶ Module myfunctions
- ▶     Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
- ▶         ' local variable declaration \*/
- ▶         Dim result As Integer
- ▶         If (num1 > num2) Then
- ▶             result = num1
- ▶         Else
- ▶             result = num2
- ▶         End If
- ▶         FindMax = result
- ▶     End Function
- ▶     Sub Main()
- ▶         Dim a As Integer = 100
- ▶         Dim b As Integer = 200
- ▶         Dim res As Integer
- ▶         res = FindMax(a, b)
- ▶         Console.WriteLine("Max value is : {0}", res)
- ▶         Console.ReadLine()
- ▶     End Sub
- ▶ End Module

# Sub procedures or Subs

- ▶ Sub procedures are procedures that do not return any value.
- ▶ The **Sub** statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is:
- ▶ Syntax:–  
[Modifiers] Sub SubName [(ParameterList)]  
    [Statements]  
End Sub

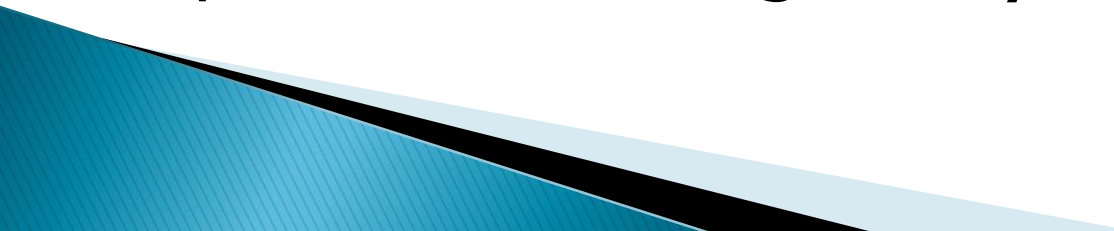
# Cont...

- ▶ Where,  
***Modifiers***: specify the access level of the procedure; possible values are: Public, Private, Protected, Friend.  
***SubName***: indicates the name of the Sub  
***ParameterList***: specifies the list of the parameters
- 

# Example

- ▶ Module mysub
- ▶ Sub CalculatePay(ByRef hours As Double, ByRef wage As Decimal)
- ▶ 'local variable declaration
- ▶ Dim pay As Double
- ▶ pay = hours \* wage
- ▶ Console.WriteLine("Total Pay: {0:C}", pay)
- ▶ End Sub
- ▶ Sub Main()
- ▶ 'calling the CalculatePay Sub Procedure
- ▶ CalculatePay(25, 10)
- ▶ CalculatePay(40, 20)
- ▶ CalculatePay(30, 27.5)
- ▶ Console.ReadLine()
- ▶ End Sub
- ▶ End Module
- ▶ When the above cod

# Passing Parameters by Value

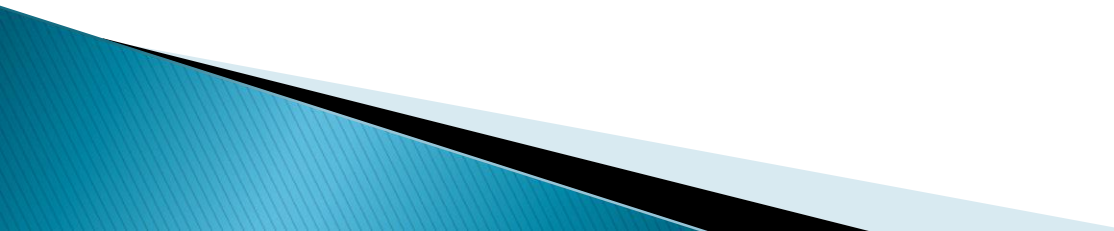
- ▶ This is the default mechanism for passing parameters to a method.
  - ▶ In this mechanism, when a method is called, a new storage location is created for each value parameter.
  - ▶ The values of the actual parameters are copied into them.
  - ▶ So, the changes made to the parameter inside the method have no effect on the argument.
  - ▶ In VB.Net, We can declare the reference parameters using the **ByVal** keyword.
- 



# Example

```
Module paramByval
    Sub swap(ByVal x As Integer, ByVal y As Integer)
        Dim temp As Integer
        temp = x ' save the value of x
        x = y    ' put y into x
        y = temp 'put temp into y
    End Sub
    Sub Main()
        ' local variable definition
        Dim a As Integer = 100
        Dim b As Integer = 200
        Console.WriteLine("Before swap, value of a : {0}", a)
        Console.WriteLine("Before swap, value of b : {0}", b)
        ' calling a function to swap the values '
        swap(a, b)
        Console.WriteLine("After swap, value of a : {0}", a)
        Console.WriteLine("After swap, value of b : {0}", b)
        Console.ReadLine()
    End Sub
End Module
```

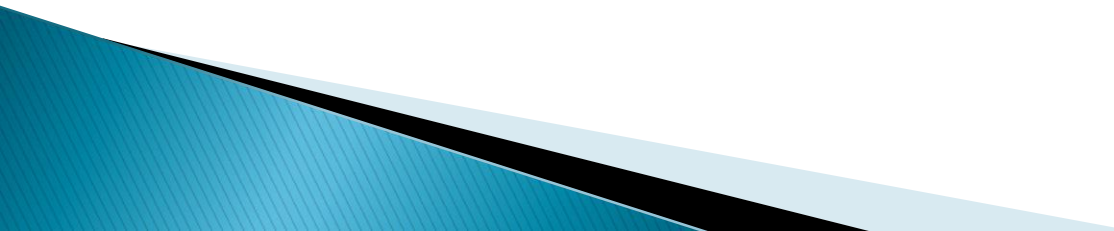
# Passing Parameters by Reference

- ▶ A reference parameter is a reference to a memory location of a variable.
  - ▶ When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
  - ▶ The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
  - ▶ In VB.Net, we can declare the reference parameters using the **ByRef** keyword.
- 

# Example

```
▶ Module paramByref
▶   Sub swap(ByRef x As Integer, ByRef y As Integer)
▶     Dim temp As Integer
▶     temp = x ' save the value of x
▶     x = y   ' put y into x
▶     y = temp 'put temp into y
▶   End Sub
▶   Sub Main()
▶     ' local variable definition
▶     Dim a As Integer = 100
▶     Dim b As Integer = 200
▶     Console.WriteLine("Before swap, value of a : {0}", a)
▶     Console.WriteLine("Before swap, value of b : {0}", b)
▶     ' calling a function to swap the values '
▶     swap(a, b)
▶     Console.WriteLine("After swap, value of a : {0}", a)
▶     Console.WriteLine("After swap, value of b : {0}", b)
▶     Console.ReadLine()
▶   End Sub
▶ End Module
```

# OOP Concepts

- ▶ Class
  - ▶ Object
  - ▶ Encapsulation
  - ▶ Inheritance
  - ▶ Interface
  - ▶ Polymorphism
- 

# Class

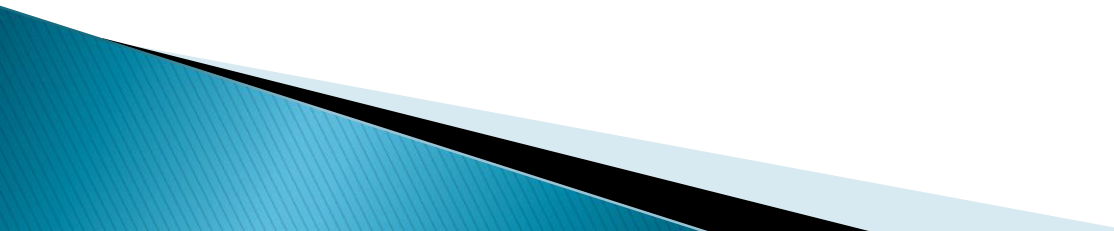
- ▶ When you define a class, you define a blueprint for a data type.
- ▶ This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
- ▶ Objects are instances of a class.
- ▶ The methods and variables that constitute a class are called members of the class.

## **Class Definition:–**

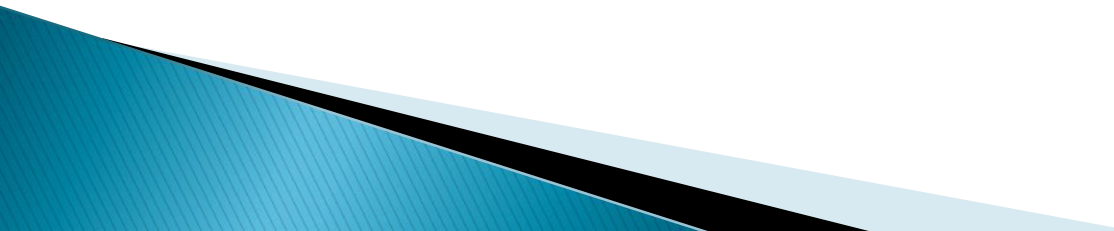
- ▶ A class definition starts with the keyword **Class** followed by the class name; and the class body, ended by the End Class statement.

# Cont...

Following is the general form of a class definition:

- ▶ Class name [ ( Of typelist ) ]
  - ▶ [ Inherits classname ]
  - ▶ [ Implements interfacenames ]
  - ▶ [ statements ]
  - ▶ End Class
- 

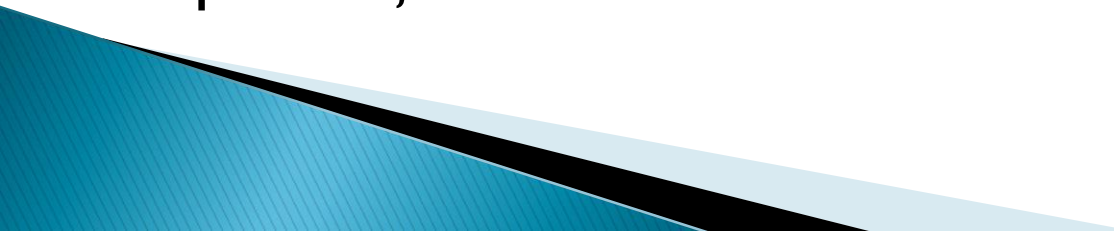
# Cont...

- ▶ 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 and Private. 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*** specifies the interfaces the class is inheriting from.
- 

```
Module Module1
    Class Box
        Public length As Double
        Public breadth As Double
        Public height As Double
    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
```



# Encapsulation

- ▶ Encapsulation is a procedure of covering up of the data & functions into a single unit called as class
  - ▶ An encapsulated object is often called an abstract data type.
  - ▶ Abstraction is the act of representing essential features without including the background details or explanations.
  - ▶ Encapsulation 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:-


```
Module Module1
    Public Class A 'class A starts
        'private member variables
        Private name As String
        Private roll_no As Integer
        Private section As String

        Sub New(ByVal n As String, ByVal r As Integer, ByVal s As String) 'constructor with
3 arguments
            name = n
            roll_no = r
            section = s
        End Sub

        Public Function myFunction() 'member method to show values of
member variables
            Console.WriteLine("Name: " & name)
            Console.WriteLine("Roll No. : " & roll_no)
            Console.WriteLine("Section: " & section)
            Return Nothing
        End Function
    End Class 'class A ends

    Sub Main()
        Dim obj As A = New A("John", 5, "A") 'creating object of A class
        obj.myFunction() 'calling member method by object
        Console.ReadKey()
    End Sub
End Module
```

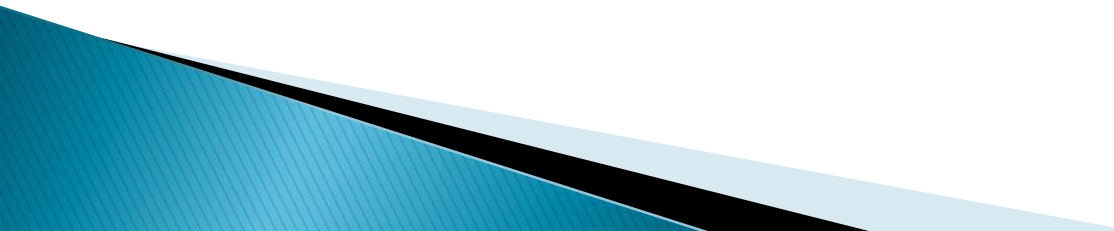
# Inheritance

- ▶ **Inheritance** is one of the feature of Object-Oriented Programming (OOPs).
  - ▶ Inheritance allows a class to use the properties and methods of another class.
  - ▶ In other words, the derived class inherits the behaviours from the base class.
  - ▶ The derived class is also called subclass and the base class is also known as super-class.
  - ▶ The derived class can add its own additional variables and methods.
  - ▶ These additional variable and methods differentiates the derived class from the base class.
  - ▶ In Vb.net the keyword **Inherits** is used in the derived class to specify its base class.
- 

# Example:–

```
Module Module2
    Public Class s1
        Public a As Integer = 5
        Public Function val() As Integer
            Return a
        End Function
    End Class
    Public Class s2
        Inherits s1
        Public c As Integer = 20
        Public Function add() As Integer
            Return c + val()
        End Function
    End Class
    Sub Main()
        Dim res As New s2
        System.Console.WriteLine("Final Value is::")
        System.Console.WriteLine(res.add())
        Console.Read()
    End Sub
End Module
```

# Interface

- ▶ *Interfaces in VB.net* are used to define the class members using a keyword **Interface**, without actually specifying how it should be implemented in a Class.
  - ▶ Interfaces are examples for multiple Inheritance.
  - ▶ Interfaces are implemented in the classes using the keyword **Implements** that is used before any Dim statement in a class.
- 

# Example:-

Module Module1

Public Interface Interface1

Function Add(ByVal x As Integer) As Integer

End Interface

Public Class first

Implements Interface1

Public Function Add(ByVal x As Integer) As Integer Implements Interface1.Add

Return x + x

End Function

End Class

Public Class second

Implements Interface1

Public Function Add(ByVal x As Integer) As Integer Implements Interface1.Add

Return x + x + x

End Function

End Class

Sub Main()

Dim obj1 As New first

Dim obj2 As New second

Dim res1, res2 As Integer

res1 = obj1.Add(10)

Console.WriteLine("Implementing x+x in first class::" & res1)

res2 = obj2.Add(50)

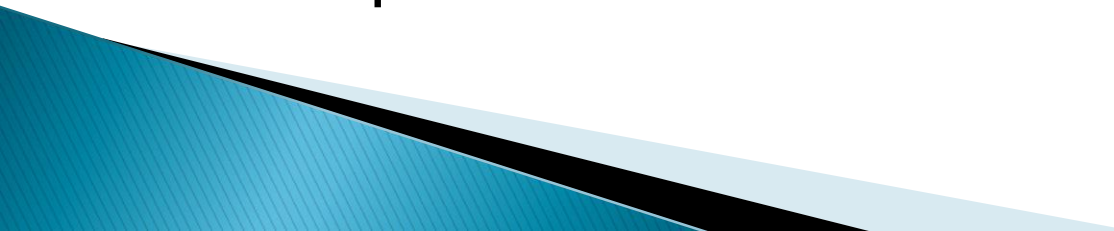
Console.WriteLine("Implementing x+x+x in second class::" & res2)

Console.Read()

End Sub

End Module

# Polymorphism

- ▶ Polymorphism means "The ability to take on different form".
  - ▶ It is also called as Overloading and Overriding with interface which means the use of same thing for different purposes.
  - ▶ Using Polymorphism we can create as many functions we want with one function name but with different argument list.
  - ▶ The function performs different operations based on the argument list in the function call.
  - ▶ The exact function to be invoked will be determined by checking the type and number of arguments in the function.
  - ▶ Example :- Interface
- 

END

Unit – II