# Visual Basic .NET (VB.NET)

## Introduction:

- Microsoft developed VB.NET in 1991 to be an event-driven programming language.
- VB6 is predecessor of VB.NET.
- VB.NET is fully object-oriented programming language.
- VB.NET can be used to develop windows desktop applications, mobile applications, web applications, and so on.
- It can be downloaded from the following link:

  **https://visualstudio.microsoft.com/vs/older-downloads/**

## Data Types

VB.NET has two major data types; Numeric Data Types and Non-Numeric Data Types.

**1- Numeric Data Types**

This class comprises numbers that divided into seven types as shown in the Table 1:

Table 1: Data Types of Numeric

| Type | Storage | Range |
|---|---|---|
| Byte | 1 byte | 0 to 255 |
| Integer | 2 bytes | -32,768 to 32,767 |
| Long | Long | -2,147,483,648 to 2,147,483,648 |
| Single | 4 bytes | -3.402823E+38 to -1.401298E-45 for negative values<br>1.401298E-45 to 3.402823E+38 for positive values. |
| Double | 8 bytes | -1.79769313486232e+308 to - |

| | | |
|---|---|---|
| | | 4.94065645841247E-324<br>for negative values<br>4.94065645841247E-324 to<br>1.79769313486232e+308<br>for positive values. |
| Currency | 8 bytes | -922,337,203,685,477.5808 to<br>922,337,203,685,477.5807 |
| Decimal | 12 bytes | +/- 79,228,162,514,264,337,593,543,950,335 if no<br>decimal is use<br>+/- 7.9228162514264337593543950335 (28 decimal places). |

**2- Non-Numeric Data Types:**

This kind of data type includes the data types which are explained in the Table 2.

Table 2: Data Types of Non-Numeric

| Type | Storage | Range |
|---|---|---|
| String(fixed length) | Length of string | Length of string |
| String(variable length) | Length + 10 bytes | 0 to 2 billion characters |
| Date | 8 bytes | January 1, 100 to December 31, 9999 |
| Boolean | 2 bytes | True or False |
| Object | 4 bytes | Any embedded object |
| Variant(numeric) | 16 bytes | Any value as large as Double |
| Variant(text) | Length+22 bytes | Same as variable-length string |

**3- Suffix for literals:**

Values which are assigned to data called literals. VB uses suffix to determine the data type of the literals for accurate calculations. Table 3 shows number of suffixes.

Table 3: suffixes

| Suffix | Data type |
|---|---|
| & | Long |
| ! | Single |
| # | Double |
| @ | Currency |

**Examples:**

StudentName = “Layla, Jasim”

LastDay = #13-June-2024#

ExpTime = #4:00 pm#

# Variables and Constants:

- VB.NET stores data in variables to be changed during the execution of program. However, it stores data in constant to keep the value of data.
- VB.NET employ **Dim** statement to declare the variable and **Const** to declare the constant.
- The rules of declaration of variables are:
  - ✓ It must be less than 255 characters
  - ✓ No spacing is allowed.
  - ✓ It must not begin with a number.
  - ✓ Period is not permitted.

**Examples:**

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Const ConstantName As Single=K
```

**Scope of variable Declaration:**

There are three keywords are used to declare a variable: **Private**, **Static**, and **Public**. The keyword **Dim** is used to define a local variable instead of **Private**. **Static** is used to preserve the variable value after the function terminated. Finally, **Public** is used to declare a global variable which can be exploited by all procedures.

# Conditional Statements

## 1- IF Statement in VB.NET:

IF statement is a decision statement and the code underneath it directly will be executed when the condition is true.

**Syntax:**

```
If condition then
        the code
End If
```

**Example:**

Write a console application in VB.NET to find a positive and negative number?

```
Module Module1
    Sub Main()
        Dim a As Integer
        a = Console.ReadLine()
        If (a > 0) Then
            Console.WriteLine("a is positve")
        Else
            Console.WriteLine("a is negative")
        End If
        Console.ReadLine()
    End Sub
End Module
```

## Nested If Else:

If . . Else statement is needed when there are more than one decisions.

**Example:**

```
Module Module1
    Sub Main()
        Dim a As Integer
        a = Console.ReadLine()
        If (a = 0) Then
            Console.WriteLine("a is equal to zero")
        Else
              If (a<0) Then
                    Console.WriteLine("a is negative")
              Else
                    Console.WriteLine("a is Positive")
               End If
        End If
        Console.ReadLine()
    End Sub
End Module
```

## The operators of If . . Else statement:

There are many operators can be used with 'If' statement.

| Operator | Description |
|---|---|
| < | Less than operator |
| > | Greater than operator |
| = | Equal to operator |
| <> | Not equal to operator |
| <= | Less than or equal to operator |
| >= | Greater than or equal to operator |

## Logical Operators:

Logical operators are used when there are two conditions should be achieved to execute the true code of 'If' statement.

| Operator | Description |
|---|---|
| And | If both conditions are true, then the result is true |
| Or | If either of the conditions are true, the result will be true |
| Not | If the condition is false, then the result will be true, or vice versa. |

**Example:**

Write a console application in VB.NET to examine the student's mark.

```
Module Module1
    Sub Main()
        Dim a As Integer
        a = Console.ReadLine()
        If (a > 50) And (a < 100) Then
            Console.WriteLine("The Student pass the exam")
        Else
            Console.WriteLine("The Student Fail in the exam")
        End If
        Console.ReadLine()
    End Sub
End Module
```

If . . Else statement is long and allocates wide storage space, therefore, Case statement is a better solution.

## 2- The Case statement

The ‘Case’ statement evaluates one value and it is more readable.

**Syntax:**

```
Select Case var
        Case 1

        Case 2

        Case Else
End Select
```

**Example:**

```
Module Module1

    Sub Main()
        Dim a As Integer
        a = Console.ReadLine()
        Select Case a
            Case Is > 0
                Console.WriteLine("a is positive")
            Case Is < 0
                Console.WriteLine("a is negative")
            Case Else
                Console.WriteLine("a is equal to zero")
        End Select
        Console.ReadLine()
    End Sub
End Module
```

It is useful to use ‘To’ keyword to express a range as shows in the following example.

**Example:**

```
Module Module1

    Sub Main()
        Dim a As Integer
        a = Console.ReadLine()
        Select Case a
            Case 0 To 50
                Console.WriteLine("The student fail in the exam")
            Case 50 To 100
                Console.WriteLine("The pass the exam")
            Case Else
                Console.WriteLine("Please, write a correct mark")
        End Select
        Console.ReadLine()
    End Sub
End Module
```

# Looping Statement

Loop statement is used to execute a statement or collection of statements many times. VB.NET has many types of loop statements as will be explained in the next sections.

## 1- For . . . Next loop statement:

This loop statement is used when we need to implement statement(s) number of times.

**Syntax:**

```
For counter As [data type] = start To end
        Statement(s)
Next
```

Where:

Counter: is variable.

Data type: the data type of the variable.

Start: the start point of the iteration.
End: the end point of the iteration.

**Example:** Write a VB.NET program to print numbers from 1 to 10.

```
Module Module1
   Sub Main()
     For i As Integer = 1 To 10
         Console.WriteLine(i)
       Next
   Console.ReadLine()
   End Sub
End Module
```

## 2- For Each . . . Next statement:

'For Each . . . Next' statement is needed to implement a set of statements for each element in a collection of data, such as an array.

**Syntax:**

```
For Each item As [data type] In group
        Statement(s)
Next
```

Where:

item: is an element in a group.
group: is a collection of data.

**Example:** Write a console program in VB.NET to print data in an array.

```
Module Module1
        Sub Main()
        Dim age() As Integer = {50, 60, 70}
                For Each i As Integer In age
                        Console.WriteLine(i)
                Next
                        Console.ReadLine()
End Sub
End Module
```

## 3- Do while . . . Loop and Do . . . Loop Until statements:

These statements are useful to execute a loop in un-limited number until the condition is False.

**Syntax of Do While . . . Loop:**

```
Do While (condition)
    Statement(s)
Loop
```

**Syntax of Do . . . Loop Until:**

```
Do
Statement(s)
Loop Until (condition)
```

**Example of Do While . . . Loop:**

```
Module Module1
    Sub Main()
        Dim i = 0, a As Integer
        Do While (i < 3)
            a = Console.ReadLine()
            Console.WriteLine(a)
            i += 1
        Loop
        Console.ReadLine()
    End Sub
End Module
```

**Example of Do . . . Loop Until:**

```
Module Module1
    Sub Main()
        Dim i = 0, a As Integer
        Do
            a = Console.ReadLine()
            Console.WriteLine(a)
```

```
            i += 1
        Loop Until (i < 3)
        Console.ReadLine()
    End Sub
End Module
```

## 4- While . . . End statement:

This statement executes the underneath statement(s) when the condition is true.

**Syntax:**

```
While  (condition)
        statement(s)
End While
```

**Example:**

```
Module Module1
    Sub Main()
        Dim i = 0, a As Integer
        While (i < 3)
            a = Console.ReadLine()
            Console.WriteLine(a)
            i += 1
        End While
        Console.ReadLine()
    End Sub
End Module
```

## 5- Nested Loop:

Nested means place a loop statement inside another loop statement.

**Example:**

```
Module Module1
    Sub Main()
        Dim i, j As Integer
```

```
        For i = 0 To 3
            For j = 0 To 3
                Console.Write(i, j)
            Next
        Next
        Console.ReadLine()
    End Sub
End Module
```

## 6- Exit and Continue statements:

These key words are useful to pass the control among the loop statements. Exit transfers the control out of the loop while Continue transfers the control to the next iteration of the loop.

**Syntax:**

Continue Do | While | For

Exit Do | While | For

# Array in VB.NET

Array is a collection of data of the same data type and stored contiguously in memory. Each data in array called element and the first element in the array is stored at the index 0 and the last element at the index n-1, where n is the number of elements.

## Declaration of array:

Dim Array_Name() As [data type]
Examples of declaration and initialization of array:
Dim stdName() As String
Or
Dim stdName(3) As String **“Array of four elements”**
or
Dim stdName() As String = {“House”, “Car”, “Door”}

or

Dim stdMark() As Integer
or
Dim stdMark(3) As Integer **"Array of four elements"**
or
Dim stdMark() As Integer = {1,2,3,4,5}
Or
ReDim stdMark(10) As Integer **" change the size of array 'stdMark' to 11 elements"**

Dim Matrix(5, 5) As Integer **"Two dimension array"**
Or
Dim Matrix = New Integer (,) ((1,2), (3,4), (4,5),(5, 6),(6,7)) As Integer "Two dimension array of (5,5)"

# Storing value in an array:

Values can be stored in an array through direct assignment or looping statements.

**Examples:**

stdMark(2) = 20

stdName(2)= Jasim

**The syntax of using loop statement:**

```
For j As Integer = 0 To 4     "array of 5 elements"
    a = console.ReadLine()
    stdMark(j) = a
Next
```

| j | a | stdMark |
|---|---|---|
| 0 | 15 | (0)=15 |
| 1 | 30 | (1)=30 |
| 2 | 3 | (2)=3 |
| 3 | 4 | (3)=4 |

| | | |
|---|---|---|
| 4 | 8 | (4)=8 |
| 5 | Exit | Exit |

**Example:** write console program in VB.NET to store data in a one dimensional array and write the content of the array?

```
Module Module1
    Sub Main()
        Dim arr(4) As Integer
        For i As Integer = 0 To 4
            Dim a As Integer = Console.ReadLine()
            arr(i) = a
        Next
        For j As Integer = 0 To 4
            Console.WriteLine(arr(j))
        Next
        Console.ReadLine()
    End Sub
End Module
```

**Example:** Write a console program that can store and print the content of two dimension array?

```
Module Module1
    Sub Main()
        Dim a As Integer
        Dim arr(3, 3) As Integer
        For i As Integer = 0 To 2
            For j As Integer = 0 To 2
                a = Console.ReadLine()
                arr(i, j) = a
            Next
        Next
        For m As Integer = 0 To 2
            For n As Integer = 0 To 2
                Console.WriteLine(arr(m, n))
            Next
        Next
        Console.ReadLine()
    End Sub
End Module
```

| i | j | a | arr(i,j)=a |
|---|---|---|---|
| 0 | 0 | 20 | arr(0,0)=20 |
| 0 | 1 | 15 | Arr(0,1)=15 |
| 0 | 2 | 30 | Arr(0,2)=30 |
| 1 | 0 | 20 | Arr(1,0)=20 |
| 1 | 1 | 35 | Arr(1,1)=35 |
| 1 | 2 | 40 | Arr(1,2)=40 |
| 2 | 0 | 50 | Arr(2,0)=50 |
| 2 | 1 | 60 | Arr(2,1)=60 |
| 2 | 2 | 70 | Arr(2,2)=70 |
| 3 | 3 | Exit loop | Exit loop |

# Functions and Sub-Procedures

## 1- Sub-Procedures:

It is a group of statements used to perform a specific task and return the control to the calling code in the program after implementing the task. Sub-Procedure can be called any number of times to execute the task and it starts with *sub* and end with *End Sub*.

**The structure of Sub-Procedure**

```
Modifiers Sub <sub-procedure> (set of parameters)
        Set of instructions
End Sub
```

## 2- Functions:

It is a set of instructions written in VB programming language starts with *Function* Keyword and ends with *End Function*. The function is used perform a task and return value and control to the calling point in the program.

**The structure of Function**

```
Modifiers Function <Function Name> (set of parameters) As Return Type
        Set of instructions
End Function
```

Where;

Modifiers are set of values that specify the access level of the function and they are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing. There are two types of functions: user defined function and pre-defined function.

**A- Pre-Defined Functions**

There are known as built-in functions and provided by the vender to facilitate the development of applications. Math functions and String functions are examples of the type of functions.

- **Math Functions** are a collection of Mathematical functions provide trigonometric, logarithmic, and other common mathematical methods.

| .NET methods | Description |
|---|---|
| Abs | Returns the absolute value of a number. |
| Acos | Returns the angle whose cosine is the specified number. |
| Asin | Returns the angle whose sine is the specified number. |
| Atan | Returns the angle whose tangent is the specified number. |
| Atan2 | Returns the angle whose tangent is the quotient of two specified numbers. |
| BigMul | Returns the full product of two 32-bit numbers. |
| Ceiling | Returns the smallest integral value that's greater than or equal to the specified Decimal or Double. |
| Cos | Returns the cosine of the specified angle. |
| Cosh | Returns the hyperbolic cosine of the specified angle. |
| DivRem | Returns the quotient of two 32-bit or 64-bit signed integers, and also returns the remainder in an output parameter. |
| Exp | Returns e (the base of natural logarithms) raised to the specified power. |
| Floor | Returns the largest integer that's less than or equal to the specified Decimal or Double number. |
| IEEERemainder | Returns the remainder that results from the division of a specified number by another specified number. |
| Log | Returns the natural (base e) logarithm of a specified number or the logarithm of a specified number in a specified base. |
| Log10 | Returns the base 10 logarithm of a specified number. |
| Max | Returns the larger of two numbers. |
| Min | Returns the smaller of two numbers. |
| Pow | Returns a specified number raised to the specified power. |
| Round | Returns a Decimal or Double value rounded to the nearest integral value or to a specified number of fractional digits. |
| Sign | Returns an Integer value indicating the sign of a number. |
| Sin | Returns the sine of the specified angle. |
| Sinh | Returns the hyperbolic sine of the specified angle. |
| Sqrt | Returns the square root of a specified number. |
| Tan | Returns the tangent of the specified angle. |
| Tanh | Returns the hyperbolic tangent of the specified angle. |
| Truncate | Calculates the integral part of a specified Decimal or Double number. |

**Example:-** find the absolute value to the number 50?

```
Module Module1
    Sub Main()
        Dim x As Double = Math.Abs(50.3)
```

```
        Dim y As Double = Math.Abs(-50.3)
        Console.WriteLine(x)
        Console.WriteLine(y)
        Console.ReadLine()
    End Sub
End Module
```

- **String Functions** are a set of functions developed to process strings in VB.NET.

| .NET methods | Description |
|---|---|
| Asc, AscW | Returns an Integer value representing the character code corresponding to a character. |
| Chr, ChrW | Returns the character associated with the specified character code. |
| Filter | Returns a zero-based array containing a subset of a String array based on specified filter criteria. |
| Format | Returns a string formatted according to instructions contained in a format String expression. |
| FormatCurrency | Returns an expression formatted as a currency value using the currency symbol defined in the system control panel. |
| FormatDateTime | Returns a string expression representing a date/time value. |
| FormatNumber | Returns an expression formatted as a number. |
| FormatPercent | Returns an expression formatted as a percentage (that is, multiplied by 100) with a trailing % character. |
| InStr | Returns an integer specifying the start position of the first occurrence of one string within another. |
| InStrRev | Returns the position of the first occurrence of one string within another, starting from the right side of the string. |
| Join | Returns a string created by joining a number of substrings contained in an array. |
| LCase | Returns a string or character converted to lowercase. |
| Left | Returns a string containing a specified number of characters from the left side of a string. |
| Len | Returns an integer that contains the number of characters in a string. |
| LSet | Returns a left-aligned string containing the specified string adjusted to the specified length. |
| LTrim | Returns a string containing a copy of a specified string with no leading spaces. |
| Mid | Returns a string containing a specified number of characters from a string. |
| Replace | Returns a string in which a specified substring has been replaced with another substring a specified number of times. |
| Right | Returns a string containing a specified number of characters from the right side of a string. |
| RSet | Returns a right-aligned string containing the specified string adjusted to the specified length. |

| | |
|---|---|
| RTrim | Returns a string containing a copy of a specified string with no trailing spaces. |
| Space | Returns a string consisting of the specified number of spaces. |
| Split | Returns a zero-based, one-dimensional array containing a specified number of substrings. |
| StrComp | Returns -1, 0, or 1, based on the result of a string comparison. |
| StrConv | Returns a string converted as specified. |
| StrDup | Returns a string or object consisting of the specified character repeated the specified number of times. |
| StrReverse | Returns a string in which the character order of a specified string is reversed. |
| Trim | Returns a string containing a copy of a specified string with no leading or trailing spaces. |
| UCase | Returns a string or character containing the specified string converted to uppercase. |

**Example:** Write a console program in VB.NET to print the length of string?

```
Module Module1
    Sub Main()
        Dim testString As String = "Hello World"
        Dim testLen As Integer = Len(testString)
        Console.WriteLine(testLen)
        Console.ReadLine()
    End Sub
End Module
```

**B- User-defined functions:** This class of functions is designed by users for a specific purpose.

Functions and Sub procedures includes parameters which are passed to them for processing purpose. Parameters in the calling function and Sub statement known as arguments. Arguments passed to the functions and Sub by two common methods; by value and by reference.

1- **Passing arguments by value *ByVal***: it is the default passing mode. When the function is called, a copy of the original data that is referenced by an argument will be copied in a new location. The new location referenced by the function parameter, therefore, the original data will not be affected after data processing by the function.

2- **Passing arguments by reference *ByRef*** : in this mode, the data will be referenced by the argument and parameter. Consequently, the original data will be changed after data processing.

**Example-1:** Write a Sub procedure to sum two numbers.

```
Module Module1
    Sub Add(ByVal a As Integer, ByVal b As Integer)
        Dim c = a + b
    End Sub

    Sub Main()
        Dim a, b, c As Integer
        a = Console.ReadLine()
        b = Console.ReadLine()
        c = Add(a, b)                    'the procedure does not provide a
            return value
        Console.WriteLine(c)
        Console.ReadLine()
    End Sub
End Module
```

**Example-2**: Write a function to sum two numbers.

```
Module Module1
    Function Add(ByVal a As Integer, ByVal b As Integer)
        Dim c = a + b
        Return c
    End Function

    Sub Main()
        Dim a, b, c As Integer
        a = Console.ReadLine()
        b = Console.ReadLine()
        c = Add(a, b)                REM The function return a value
        Console.WriteLine(c)
        Console.ReadLine()
    End Sub
```

**Example-3:** the example illustrates the difference between ByVal and ByRef.

```
Module Module1
    Function Add(ByRef a As Integer, ByVal b As Integer)
        a += 1
        b += 1
    End Function

    Sub Main()
        Dim a, b As Integer
        a = 3
        b = 5
        Add(a, b)
        Console.WriteLine("a = {0}", a)
        Console.WriteLine("b = {0}", b)
        Console.ReadLine()
    End Sub
End Module
```

**The same example but using Sub procedure.**

```
Module Module1
    Sub Add(ByRef a As Integer, ByVal b As Integer)
        a += 1
        b += 1
    End Sub

    Sub Main()
        Dim a, b As Integer
        a = 3
        b = 5
        Add(a, b)
        Console.WriteLine("a = {0}", a)
        Console.WriteLine("b = {0}", b)
        Console.ReadLine()
    End Sub
End Module
```

# File

**File** is a group of data saved in a disk with a unique name and path. **File stream** is reading and writing data from and to a file. The namespace *System.IO* contains many classes that are employed to process the file content. In this lecture, the common classes for handling files will be explained as follows.

1- **StreamReader**, this function is used to explore the file lines through loop control statement.

**Example:**

```
Imports System.IO
Module Module1

    Sub Main()
Using reader As StreamReader = New StreamReader("E:\Test.txt")

            Do

                Dim line As String = reader.ReadLine

                If line Is Nothing Then
                    Exit Do
                End If
                Console.WriteLine(line)
            Loop
            Console.ReadLine()
        End Using
    End Sub
End Module
```

2- **ReadAllText,** this function read the complete file content without loop statement.

**Example:**

```
Imports System.IO
Module Module1
    Sub Main()
        Dim value As String = File.ReadAllText("E:\Test.txt")
        Console.WriteLine(value.Length)
        Console.WriteLine(value)
        Console.ReadLine()
    End Sub
End Module
```

3- **ReadAllLines,** this function reads the file and store each line in a String array.

**Example:**

```
Imports System.IO
Module Module1
    Sub Main()
        Dim array As String() = File.ReadAllLines("E:\Test.txt")
        For Each line As String In array
                        ' Use the line to find each line length.
            Dim length As Integer = line.Length
            Console.WriteLine(length)
        Next
        Console.ReadLine()
    End Sub
End Module
```

4- **List,** ReadAllLines method stores file lines in a list using ToList rather than an array.

**Eaxmple:**

```
Imports System.IO
Module Module1
    Sub Main()
     Dim list As List(Of String) = File.ReadAllLines("E:\Test.txt").ToList
        Console.WriteLine(list.Count)
       Console.ReadLine()
    End Sub
    End Module
```

5- **WriteAllLines,** this function takes the input as lines in a String array and save them in a text file.

**Example:**

```
Imports System.IO
Module Module1
    Sub Main()
        Dim array As String() = New String() {"door", "dog", "arrow"}
        File.WriteAllLines("E:\Test.txt", array)
        Console.ReadLine()
    End Sub
End Module
```

# Classes and objects

**Class** is a piece of software that describes an object through a set of members: methods and data.

**The common class syntax**

[Access Modifier] *Class Name*

Statements

End Class

Access modifiers are optional and have the values: Public, Private, Protected, Friend, Protected Friend.

| Access Modifier Type | Definition |
|---|---|
| Public | The member can be accessed by the code in the same assembly or another one. |
| Private | The member can be accessed by the code in the same class. |
| Protected | The member can be accessed by the code in the same class or by the derived class. |
| Friend | The member can be accessed by the current assembly. |
| Protected Friend | The member can be accessed by the current assembly and the derived class. |

An **Object** is an instance of a class, and the process of creating an object is called instantiation.

**The syntax of initialization process**

Dim *Object Name* As New *Class Name*

Or

Dim *Object Name* As *Class Name* = New *Class Name* ()

**Example:** Write a console application in VB.NET to calculate the volume of two rectangles.

```
Module Module1

    Class Rect
        Public width As Double   ' width of a rectangle
        Public height As Double   ' Height of a rectangle
    End Class

    Sub Main()
        Dim Rect1 As Rect = New Rect()       ' Declare Rectangle1 of type Rectangle
        Dim Rect2 As Rect = New Rect()       ' Declare Rectangle2 of type Rectangle
        Dim volume As Double = 0.0         ' Store the volume of a Rectangle here

        ' Rectangle 1 specification
```

```
        Rect1.width = 5.0
        Rect1.height = 6.0

        ' Rectangle 2 specification
        Rect2.width = 10.0
        Rect2.height = 12.0

        'volume of Rectangle 1
        volume = Rect1.width * Rect1.height
        Console.WriteLine("Volume of Rectangle1 : {0}", volume)

        'volume of Rectangle 2
        volume = Rect2.width * Rect2.height
        Console.WriteLine("Volume of Rectangle2 : {0}", volume)
        Console.ReadLine()
    End Sub
End Module
```

**Another solution**

```
    Module Module1

        Class Rect
            Public width As Double   ' wedth of a rectangle
            Public height As Double   ' Height of a rectangle

            Public Function GetVolume() As Integer
                Dim Volume = width * height
                Return Volume
            End Function

        End Class

        Sub Main()
            Dim Rect1 As Rect = New Rect()       ' Declare Rectangle1 of type Rectangle
            Dim Rect2 As Rect = New Rect()       ' Declare Rectangle2 of type Rectangle
            Dim volume As Double = 0.0        ' Store the volume of a Rectangle here

            ' Rectangle 1 specification
            Rect1.width = 5.0
            Rect1.height = 6.0
            volume = Rect1.GetVolume()
```

```
        ' Rectangle 2 specification
        Rect2.width = 10.0
        Rect2.height = 12.0


        'volume of Rectangle 1
        Console.WriteLine("Volume of Rectangle1 : {0}", Rect1.GetVolume())

        'volume of Rectangle 2
        Console.WriteLine("Volume of Rectangle2 : {0}", Rect2.GetVolume())

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

## Constructors and Destructors

*Constructor* is a unique member of *Sub* a class that is executed whenever a new object is created of that class. The constructor is called *New*, and it does not have a return type. It does not have parameters, but in some circumstances, assigning values to the constructor is helpful. *Destructor* is called *Finalise* is a member *Sub* of the class and does not have parameters and return a type. It is executed when the execution of the application is ending to release the resources, files, and so on.

**Example:**

```
Class Rect
    Public width As Double   ' width of a rectangle
    Public height As Double   ' Height of a rectangle

    Public Sub New()       'Constructor
    End Sub

   Protected Overrides Sub Finalize()      'Disrtructor must be Protected
   End Sub

    Public Function GetVolume() As Integer
        Dim Volume = width * height
        Return Volume
```

```
    End Function
End Class

Sub Main()
    Dim Rect1 As Rect = New Rect()       ' Declare Rectangle1 of type Rectangle
    Dim Rect2 As Rect = New Rect()       ' Declare Rectangle2 of type Rectangle
    Dim volume As Double = 0.0         ' Store the volume of a Rectangle here

    ' Rectangle 1 specification
    Rect1.width = 5.0
    Rect1.height = 6.0
    volume = Rect1.GetVolume()

    ' Rectangle 2 specification
    Rect2.width = 10.0
    Rect2.height = 12.0

    'volume of Rectangle 1
    Console.WriteLine("Volume of Rectangle1 : {0}", Rect1.GetVolume())

    'volume of Rectangle 2
    Console.WriteLine("Volume of Rectangle2 : {0}", Rect2.GetVolume())

    Console.ReadLine()
End Sub
```

## Base Class (Superclass) and Derived Class (Subclass):

Base class is the class which created by the developer and derived class is an instance of the Base class.

### Syntax

<access modifier> Class <class name>

Instructions

End Class

Class <derived Class> : Inherits <base class>

Instructions

End Class

## Inheritance

Is the process of assigning the members of the base class to the derived class. Therefore, the base class should be created before the derived class. As can be seen in the previous examples, that classes Rect1 and Rect2 inherited the members of the base class Rect which known as *MyBase* in VB.NET.

### Example

```
Module Module1
    Dim x, y, z As Integer

    Class input
        Sub read()
            Console.WriteLine("Enter two values")
            x = Console.ReadLine()
            y = Console.ReadLine()
        End Sub
    End Class

    Class output
        Inherits input
        Sub result()
            z = x + y
            Console.WriteLine("Addition result are = " & z)
        End Sub
    End Class

    Sub Main()
        Dim obj As New output
        obj.read()
        obj.result()
        Console.ReadLine()
    End Sub
End Module
```

# Basic Controls of VB.NET

Object in Visual Basic can be created using VB form and controls in toolbox. Every control in toolbox consists of three elements:

1- **Properties** which explain the project properties, such as fore color caption. The properties and it can be updated during design time using properties window or at run time through statements in the program code.

Object. Property = Value

Where,

Object is the name of the object.
Property is the feature of the object that can be changed.
Value is the new value of the property.

For example:

Form1.Text = “Hello World”

2- **Method** is a procedure (function or Sub) that is used to perform a specific task. there are two types of methods:

- A method which is provided with the control of the toolbox.
- A user defined method.

For Example:

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        MessageBox.Show("Hello World")
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        Me.Text = "Hello World"
    End Sub
End Class
```

3- **Event** is a signal informs the application that something has occurred. For instance, when a user presses a keyboard key, the Form raises a click event and calls a method to handle the event.

The following table is a list of control classes which are defined in the System.Wndows.Forms namespace.

| Sr. No. | Control Class | Discription |
|---|---|---|
| 1 | Forms | The container for all the controls that make up the user interface. |
| 2 | TextBox | It represents a Windows text box control. |
| 3 | Label | It represents a standard Windows label. |
| 4 | Button | It represents a Windows button control. |
| 5 | ListBox | It represents a Windows control to display a list of items. |
| 6 | ComboBox | It represents a Windows combo box control. |
| 7 | RadioButton | It enables the user to select a single option from a group of choices when paired with other RadioButton controls. |
| 8 | CheckBox | It represents a Windows CheckBox. |
| 9 | PictureBox | It represents a Windows picture box control for displaying an image. |
| 10 | ProgressBar | It represents a Windows progress bar control. |
| 11 | ScrollBar | It Implements the basic functionality of a scroll bar control. |
| 12 | DateTimePicker | It represents a Windows control that allows the user to select a date and a time and to display the date and time with a specified format. |
| 13 | TreeView | It displays a hierarchical collection of labeled items, each represented by a TreeNode. |
| 14 | ListView | It represents a Windows list view control, which displays a collection of items that can be displayed using one of four different views. |

**Example1:** Write an application to sum the value in TextBox1 with the value in TextBox2 and the Print the result in message box after pressing on a Button.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    MsgBox("The sum is  = " & Val(TextBox1.Text) + Val(TextBox2.Text))
End Sub
```

**Example2:** Write an application to sum the value in TextBox1 with the value in TextBox2 and the Print the result in TextBox3 after pressing on a Button.

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    Dim num1 As Single = Val(TextBox1.Text)
    Dim num2 As Single = Val(TextBox2.Text)
    Dim Sum As Single = num1 + num2
    TextBox3.Text = Sum.ToString
     End Sub
```

**Example3:** Write an application to sum the value in TextBox1 with the value in TextBox2 and the Print the result in a Label after pressing on a Button.

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    Dim num1 As Single = Val(TextBox1.Text)
    Dim num2 As Single = Val(TextBox2.Text)
    Dim Sum As Single = num1 + num2
    Label1.Text = Sum.ToString
```

**Listbox** control is used to display a group of items and allow users to add items during in the listbox control during design time or run time.
The common properties of listbox as can be seen in the following table:

| Property name | Explanations |
|---|---|
| Items | Gets the items of the ListBox. |
| SelectedItem | Gets or sets the currently selected item in the ListBox. |
| SelectedItems | Gets a collection containing the currently selected items in the ListBox. |
| selectedValue | Gets or sets the value of the member property specified by the ValueMember property. (Inherited from ListControl) |

ListBox control has a number of methods and the following table contains a set of common procedures:

| Method name | Explanations |
|---|---|
| Add | The Add() method is used to add items to an item collection. |
| Remove | It is used to remove an item from an item collection. However, we can remove items using |

| | |
|---|---|
| | the item name. |
| Clear | It is used to remove all items from the item collection at the same time. |
| Contains | It is used to check whether the particular item exists in the ListBox or not. |

**Example:** A VB.NET project that enables users to add and delete an item from a listbox control.

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        ListBox1.Items.Add(TextBox2.Text())
        TextBox2.Clear()
    End Sub

    Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles ListBox1.SelectedIndexChanged
        MessageBox.Show(ListBox1.SelectedIndex.ToString())
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        ListBox1.Items.Remove(ListBox1.SelectedItem)
        MessageBox.Show("The selected item has been deleted successfully")
    End Sub
End Class
```

**ComboBox control** is a combination of textbox and drop-down list which permit users to select an item among list of items or add a new item to the existing list. ComboBox class has number of properties and the following table has the common of them.

| Property name | Explanations |
|---|---|
| Items | Gets an object representing the collection of the items contained in this ComboBox. |
| SelectedItem | Gets or sets currently selected item in the ComboBox |

| | |
|---|---|
| SelectedIndex | Gets or sets the index specifying the currently selected item. |
| SelectedValue | Gets or sets the value of the member property specified by the ValueMember property. |

ComboBox control has also set of methods as explained in the following table the common of these methods.

| Method name | Explanations |
|---|---|
| SelectAll | Selects all the text in the editable area of the combo box. |
| FindString | Finds the first item in the combo box that starts with the string specified as an argument. |
| FindStringExact | Finds the first item in the combo box that exactly matches the specified string. |

**Example** of add item to combobox.

```
Public Class Form1
   Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
      ComboBox1.Items.Add(TextBox1.Text)
      TextBox1.Clear()
   End Sub
End Class
```

**Example** of removing an item from a list of items in comboBox.

```
Public Class Form1
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
      If ComboBox1.SelectedIndex <> -1 Then
         ComboBox1.Items.RemoveAt(ComboBox1.SelectedIndex)
      End If
   End Sub
```

**Example** of clearing combobox

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button3.Click
    ComboBox1.Items.Clear()
End Sub
```

**Example** of retrieving information from comboBox control.

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles ComboBox1.SelectedIndexChanged
    MessageBox.Show(ComboBox1.Text)
    Label1.Text = ComboBox1.Text
    TextBox1.Text = ComboBox1.Text
End Sub
```

**PictureBox** control is used to display images on the form either at design time or run time. This control has number of properties as explained in the following table the common of them.

| Property name | Explanations |
|---|---|
| Image | The image property is used to display the image on the PictureBox of a Windows form. |
| ImageLocation | It is used to set or get the path or URL of the image displayed on the picture box of the window form. |
| BackColor | It is used to set the background color for the PictureBox in the window form. |
| BackgroundImage | It is used to set the background image of a window form by setting or getting value in the picture box. |

The ordinary methods of PictureBox class are demonstrated in the following table.

| Method name | Explanations |
|---|---|
| Load | The Load() method is used to load the specified image from the control using the ImageLocation property. |
| GetStyle | The GetStyle() method is used to get values for the specified bit style in the PictureBox control. |
| ToString | Returns the string that represents the current picture box. |

Pictures can be loaded during design time through the properties window, and they can be loaded during run time.

**Example** of loading image at runtime.

The **first** method is mentioned the path to the required image.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        PictureBox1.Image = Image.FromFile("C:\Users\Samer\Pictures\Nature
Pictures\2.jpg")
    End Sub
```

The **second** method employs the OpenFileDialog (OFD) to specify the path to the needed image through a dialog window. It is important to determine the types of image files in Filter property in the property window (*.JPEG | *.JPG | *.BMP |*.GIF).

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        OFD1.ShowDialog()
        PictureBox1.Load(OFG1.FileName)
    End Sub
```

**Windows media player control** is used to play different kinds of media files, for example; mp3, mpeg, avi, wave and so on. This control can be added to the toolbox window through the following steps:

1- If windows media player control not exists, then press wright mouse button on the combobox control and select “choose items” from the drop down list.
2- Choose COM Components tab from the window.
3- Tick the check box of the windows media player control from the list of components and press “OK’ to the control to the toolbox.

**Example** of using windows media player control in VB.NET.

1- Drag and drop the control form the toolbox to the form.

2- Drag and drop OpenFileDialog (OFD) control to specify the path to the media file through dialog window.

3- Add new button to the form and change the text property of the button to the "Load" word. After that, write the following instruction to the "Load" button to a media file from the memory.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    If (OpenFileDialog1.ShowDialog = DialogResult.OK) Then
        AxWindowsMediaPlayer1.URL = OpenFileDialog1.FileName
    End If
End Sub
```

4- Add another button to the form and change the text property of the button to "Play" word and write the following instructions to the button to play the selected file.

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
    AxWindowsMediaPlayer1.Ctlcontrols.play()
End Sub
```

5- Add more button to stop the play of the media file and change the text property to the word "Stop" and assign the following code to the button.

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button3.Click
    AxWindowsMediaPlayer1.Ctlcontrols.stop()
End Sub
```

6- Add the last button to the form and change its text property to the word "Pause" to pause the play of the file and assign the following code to the last button.

```
Private Sub Button4_Click(sender As Object, e As EventArgs) Handles
Button4.Click
    AxWindowsMediaPlayer1.Ctlcontrols.pause()
End Sub
```

**Timer** control is a hidden control during the execution of the application. There are many applications that use timer, such as: clock, animation, control of watching, and etc.

**Example** of display clock:

1- Add Timer control to the form.

2- Add label control

3- Assign the following code to the label:

```
Private Sub Label1_Click(sender As Object, e As EventArgs) Handles
Label1.Click
    Label1.Text = TimeOfDay
End Sub
```

**Example** of Stopwath application using time control.

1- Add Timer control to the form and assign the following codes.

```
  Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
    TextBox1.Text = TextBox1.Text + 1
  End Sub
```

2- Change the Interval property from 100 to 1000 which is equal 1 millisecond.

3- Add TextBox to the form and change the text property to “1”.

4- Add button and change the text property to “Start” and assign the following instruction to the button.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Timer1.Start()
End Sub
```

5- Add another button and also change the text property to “Stop” and assign the following code.

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
    Timer1.Stop()
End Sub
```

**ProgressBar** control is used when the application performs some tasks such as: printing, copying, installing, and so on.

**Example** of using ProgressBar in VB.NET.

1- Insert Timer control to the form and add the following code.

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
    If ProgressBar1.Value <= ProgressBar1.Maximum - 1 Then
       ProgressBar1.Value += 1
    End If
End Sub
```

2- Insert button to the form and write the following instruction

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    Timer1.Enabled = True
End Sub
```

**ScrollBar** control is used to show a large amount of information in the form. There are two types of scrollbar control: Horizontal ScrollBar (HScrollBar) and Vertical ScrollBar (VScrollBar).

**DateTimePicker** class allows you to choose date and time through by editing the information in the control.

**MenuStrip** control is similar to a container for menu structure. Each item in the MenuStrip control has a specific function such as the word “Exit”.

**Example:-** use the example in windows media player to play media.

1- Insert OpenFileDialog class to the form.
2- Insert MenuStrip control to the form and use the items of the MenuStrip to Load, Play, Stop, and Pause the media.
3- Add the following codes to each corresponding item in the Menu.

```
Public Class Form1
    Private Sub LoadToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles LoadToolStripMenuItem.Click
        If (OpenFileDialog1.ShowDialog = DialogResult.OK) Then
            AxWindowsMediaPlayer1.URL = OpenFileDialog1.FileName
        End If
    End Sub

    Private Sub StopToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles StopToolStripMenuItem.Click
        AxWindowsMediaPlayer1.Ctlcontrols.play()
    End Sub

    Private Sub PauseToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles PauseToolStripMenuItem.Click
        AxWindowsMediaPlayer1.Ctlcontrols.stop()
    End Sub

    Private Sub PlayToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles PlayToolStripMenuItem.Click
        AxWindowsMediaPlayer1.Ctlcontrols.pause()
    End Sub

    Private Sub ExitToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles ExitToolStripMenuItem.Click
        End
    End Sub
End Class
```

# Connect MS Access database with VB.NET

To process of data In MS Access database using VB.NET project needs two necessary steps: First, “import system.data.OleDb” which is the name space that includes the needed methods for the connection. Second, “connection string”, which is a sequence of words that describe the path to the database.

**Example:**

Let as consider the following instructions:

```
Imports System.Data.OleDb

Module conn
    Public con As New OleDbConnection   ' con is the connection name
    Public ConStr As String = "Provider=Microsoft.ACE.OLEDB.16.0; Data
      Source= C:\Users\Samer\Documents\Database7.accdb;"       ' connection string
      is the path to the database
End Module
```

The name space “Imports System.Data.OleDb” involve an application called **Object Linking and Embedded Database** OleDb tailored by Microsoft. It is an interface between the applications and the database by receiving the user request and directs it to the appropriate dataset. The variable **con** is the connection name and **conStr** describes the path to the database “Database7.accdb”.

After establishing the connection, VB application needs a specific class called **DataGridView** to display the database content in a tabular form. More classes are needed such as **DataTable** and **DataAdapter**. The former to The first class is class. **Datatable** control is an in-memory data representation that has rows and columns. **DataAdapter** control is a bridge between the data source and data set in memory.

The following example is an application to show the content of table “Students” which belongs to the database “Database7”.

**Example:**

```
Imports System.Data.OleDb
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        Try
            con.ConnectionString = ConStr

            If con.State = ConnectionState.Closed Then
                con.Open()
                Load_Data()     'Function to fetch data from database to Grid
View
            End If
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
        con.Close()
    End Sub

    Sub Load_Data()
        Dim da1 As New OleDbDataAdapter
        Dim dt1 As New DataTable

        dt1.Clear()
        da1 = New OleDbDataAdapter("Select * from Students", con)
        da1.Fill(dt1)
        DataGridView1.DataSource = dt1

    End Sub
End Class
```

**Example of a Module which uses to connect MS Access with VB.NET form:-**

```
Imports System.Data.OleDb
Module connect
    Public con As New
OleDb.OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\Users\Samer\Documents\School_Info.accdb;")
    Public dt As New DataTable
    Public da As New OleDbDataAdapter
End Module
```

**Examples of processing data in MS Access using a VB.NET form.**

```
Imports System.Data.OleDb

Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        Try
            If con.State = ConnectionState.Closed Then
                con.Open()
                Load_Data()
                DGV1.Columns(0).HeaderText = "التسلسل"
                DGV1.Columns(1).HeaderText = "الطالب اسم"
                DGV1.Columns(2).HeaderText = "الطالب عمر"
                DGV1.Columns(3).HeaderText = "المرحلة"

            End If
        Catch ex As Exception
        End Try
        con.Close()
    End Sub

    Sub Load_Data()
        Dim da As New OleDbDataAdapter
        Dim dt As New DataTable

        dt.Clear()
        da = New OleDbDataAdapter("Select * from Student_Info", con)
```

```
        da.Fill(dt)
        DGV1.DataSource = dt

    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        Add_Data()
        clear_data()
        Load_Data()
        MsgBox("The recored is added")
    End Sub

    Private Sub Add_Data()
        Try
            Dim cmd As OleDbCommand = con.CreateCommand
            If con.State = 1 Then con.Close()
            con.Open()
            cmd.CommandText = "insert into Student_Info ([ST_Name], [ST_Age],
[ST_Level]) values('" & TextBox1.Text & "','" & TextBox2.Text & "','" &
TextBox3.Text & "')"
            cmd.Parameters.AddWithValue("@ST_Name", Trim(TextBox1.Text))
            cmd.Parameters.AddWithValue("@ST_Age", Trim(TextBox2.Text))
            cmd.Parameters.AddWithValue("@ST_Level", Trim(TextBox3.Text))
            cmd.ExecuteNonQuery()
            con.Close()

        Catch ex As Exception
            con.Close()
        End Try

    End Sub

    Sub clear_data()
        TextBox1.Text = ""
        TextBox2.Text = ""
        TextBox3.Text = ""
        TextBox4.Text = ""
    End Sub
```

```
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        Delete_Data()
        clear_data()
        Load_Data()
    End Sub

    Private Sub Delete_Data()
        Dim dt As New DataTable
        Dim da As New OleDbDataAdapter
        dt.Clear()
        da = New OleDbDataAdapter("delete * from Student_Info where id = " &
DGV1.CurrentRow.Cells(0).Value, con)
        da.Fill(dt)
        Load_Data()
    End Sub

    Private Sub Edit_Data()
        Dim SaveInto As New OleDbCommand

        Try
            SaveInto.Connection = con
            SaveInto.CommandType = CommandType.Text
            SaveInto.CommandText = "UPDATE Student_Info SET
ST_Name=@ST_Name, ST_Age=@ST_Age, ST_Level=@ST_Level where ID =
@ID"
            SaveInto.Parameters.Clear()

            SaveInto.Parameters.AddWithValue("@ST_Name", TextBox1.Text)
            SaveInto.Parameters.AddWithValue("@ST_Age", TextBox2.Text)
            SaveInto.Parameters.AddWithValue("@ST_Level", TextBox3.Text)
            SaveInto.Parameters.AddWithValue("@ID", TextBox4.Text)

            If con.State = ConnectionState.Open Then con.Close()
            con.Open()
            SaveInto.ExecuteNonQuery()
            con.Close()

        Catch ex As Exception
            con.Close()
```

```
        End Try
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button3.Click
        Edit_Data()
        clear_data()
        Load_Data()
    End Sub

    Private Sub DGV1_DoubleClick(sender As Object, e As EventArgs) Handles
DGV1.DoubleClick
        from_DGV1()
    End Sub

    Private Sub from_DGV1()
        Dim iRowIndex As Integer
        For i As Integer = 0 To DGV1.SelectedCells.Count - 1
            iRowIndex = DGV1.SelectedCells.Item(i).RowIndex
            TextBox1.Text = DGV1.Rows(iRowIndex).Cells(1).Value
            TextBox2.Text = DGV1.Rows(iRowIndex).Cells(2).Value
            TextBox3.Text = DGV1.Rows(iRowIndex).Cells(3).Value
            TextBox4.Text = DGV1.Rows(iRowIndex).Cells(0).Value
        Next
    End Sub

End Class
```