Overview of Microsoft .NET Framework

- The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services.
- Using .Net technology it is easy to reduce time and cost for developing and maintaining enterprise business applications.
- .NET framework SDK is free and includes command-line compilers for C++, C#, and VB.NET and various other utilities for development.
- Developers can easily develop windows applications, web applications, mobile applications, etc... Using .NET framework.
- First version of .NET was 1.0 and released in 2002 whereas latest version available in market is .NET 4.6.2, released in 2016.
- .NET framework consists several layers as shown in Fig 1.1 below:

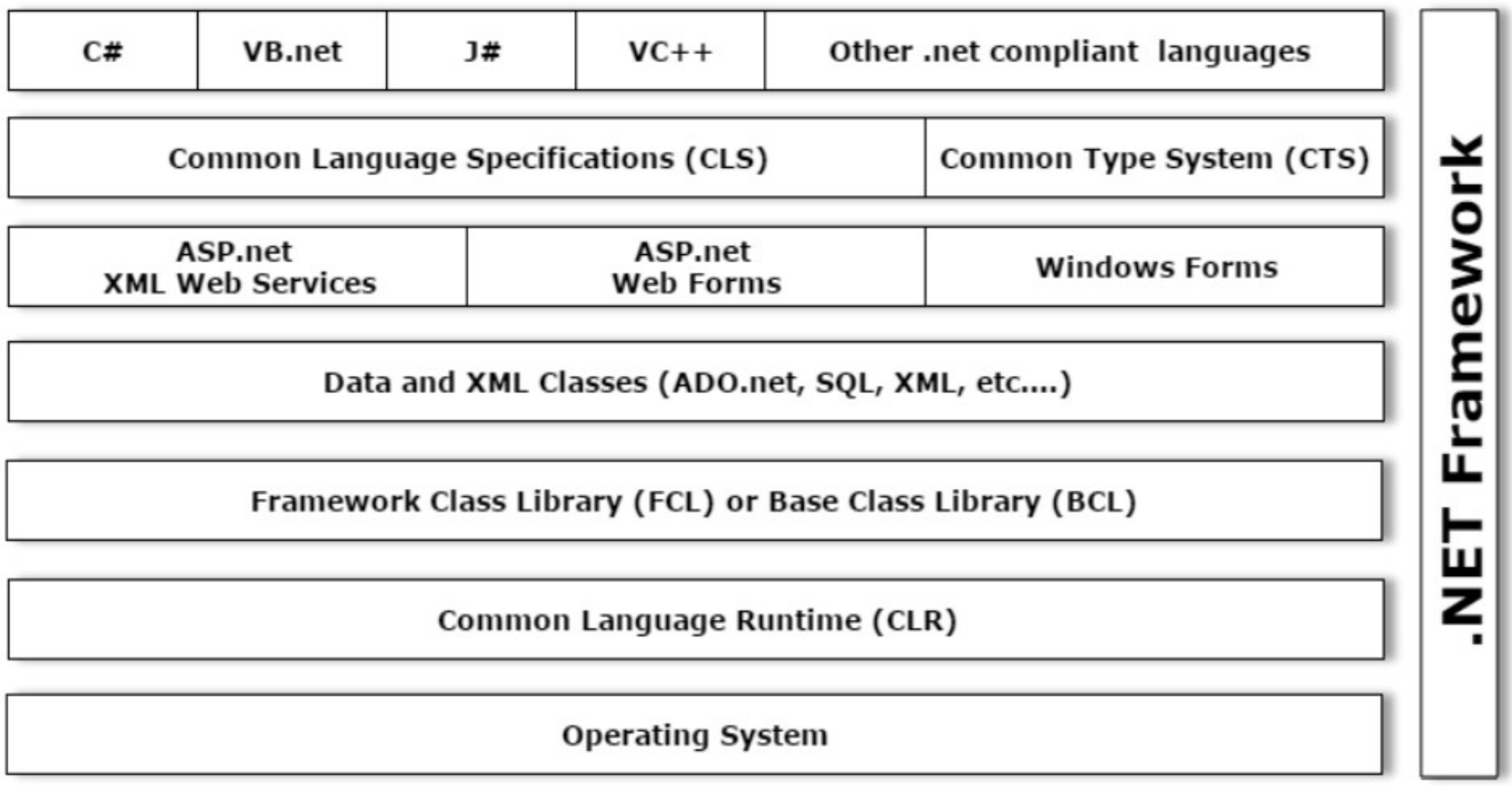


Fig 1.1 .NET Framework Architecture Block Diagram

.NET Framework Components:

Common Language Runtime (CLR):

- The heart of the .Net framework is the **Common Language Runtime (CLR).**
- CLR manages the execution of the whole application.
- In the CLR, code is expressed as the byte code called the **MSIL code (MSIL = Microsoft Intermediate Language).**
- Developers using the CLR, write code in a language such as C# or VB.NET. At Compile time, a .NET Compiler converts such code into MSIL code.
- During run time, the CLR converts the MSIL code into something that can be understood by the operating system.

- At runtime, MSIL's just-in-time compiler converts the MSIL code to the Native code.
- JIT compiler converts the MSIL code in to the Native Code, which is related to the particular systems.
- Rather than converts all the code to the native code JIT compiler will converts the required amount of code to the Native code and saves the time and memory both.
- The CLR manages memory, thread execution, garbage collection, exception handling, common type system, code safety verifications and other system services.

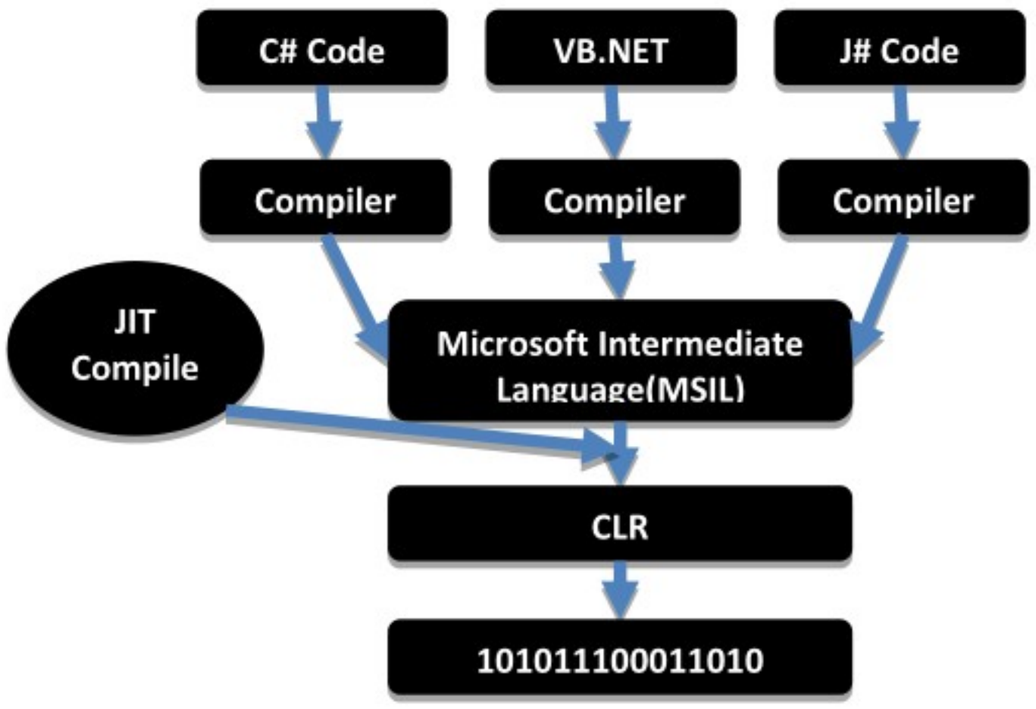


Fig 1.2 Common Language Runtime

Framework Class Library (FCL) or Base Class Library (BCL):

- **The Framework Class Library (FCL)** is a standard library and Microsoft's .NET Framework implementation of the Standard Libraries as defined in the **Common Language Infrastructure.**
- The .NET Framework class library is a library consisting of **namespaces**, **classes**, **interfaces**, and **Datatypes** included in the .NET Framework.
- The Base Class Library (BCL) is the core of the FCL and provides the most fundamental functionality, which includes classes in namespaces **System, System.Collections, System.IO and System.Text**

Data and XML Classes (ADO.net, SQL, XML, etc…):

- These classes extend the FCL to support data management and XML manipulation.
- ADO.NET extends the base classes and managing the data to the database and maintains the XML data manipulations.
- These all classes support database management with the help of Structured Query Language (SQL) and it also allows you to change the data.
- The .Net framework also supports the classes which will help you to maintain the XML data and perform searching and translations.

Applications (Windows, Web, Web Services, etc…):

- Applications are the interface between users and computers that allow .NET to interact with the outside world.
- With the help of ASP.NET we can develop more powerful and attractive web applications compare to ASP.

- With the help of Windows Applications we can develop the different kind of software and ERP.
- With the help of .NET Framework we can do the Rapid Application Development (RAD) by using the Web forms and Windows Forms.

Common Language Specification (CLS):

- **Common Language Specification (CLS)** is a document that says how computer programs can be turned into bytecode.
- **Common Language Specification (CLS)** is a set of basic language features that .Net Languages needed to develop Applications and Services, which are compatible with the .Net Framework.
- When there is a situation to communicate Objects written in different .Net languages, those objects must expose the features that are common to all the languages.
- **Common Language Specification (CLS)** ensures complete interoperability among applications, regardless of the language used to create the application.
- **Common Language Specification (CLS)** defines a subset of **Common Type System (CTS).**

Common Type System (CTS):

- As .Net framework is language independent and support over 20 different programming languages, many programming language will write data types in their own programming languages.
- For example, an integer variable in **C# is written as Int**, where as in **visual basic it is written as integer**.
- CTS define the basic data types that IL understands. Each .NET compliant language should map its data types to these standard data types.
- This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to/from each other.

.NET Compliant Languages:

- This is the top most layer, it consists of .NET complaint languages such as, C#, VB.NET, J#, VC++, etc…
- This feature of .NET gives the facility to write programs in multiple languages, which allows programmers to use their favorite languages.

Just-In-Time Compiler (JITTER):

- In the .NET Framework, all the Microsoft .NET languages use a Common Language Runtime, which solves the problem of installing separate runtimes for each of the programming languages.
- Before the Microsoft Intermediate Language (MSIL) can be executed, it must be converted by a .NET Framework Just-In-Time (JIT) compiler to native code, which is CPU-specific code that runs on the same computer architecture as the JIT compiler.

IL (Intermediate Language):

- MSIL or IL (Intermediate Language) is machine independent code generated by .NET framework after the compilation of program written in any language by you.
- The MSIL code includes instruction to load, initialize and invoke methods on objects.

Garbage Collection (GC):

- The .Net Framework provides a new mechanism for releasing unreferenced objects from the memory (that is no longer needed in the program), this process is called **Garbage Collection (GC).**
- It provides the following benefits:
  - Enables you to develop your application without having to free memory.
  - Allocates objects on the managed heap efficiently.
- The garbage collector takes care of bulk of the memory management responsibility, freeing up the developer to focus on core issues.
- The garbage collector is optimized to perform the memory free-up at the best time based upon the allocations being made.

Assemblies:

- An assembly is a portable executable file that contains a compiled portion of code.
- All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest.
- Developers or intermediate users can use and/or reuse them to build their applications as rapid as possible.

Getting Started with Visual Basic .net IDE

- **Integrated Development Environment (IDE)** consist of inbuilt compiler, debugger, editors, and automation tools for easy development of code. Visual Basic.net IDE can be accessed by opening a new project.

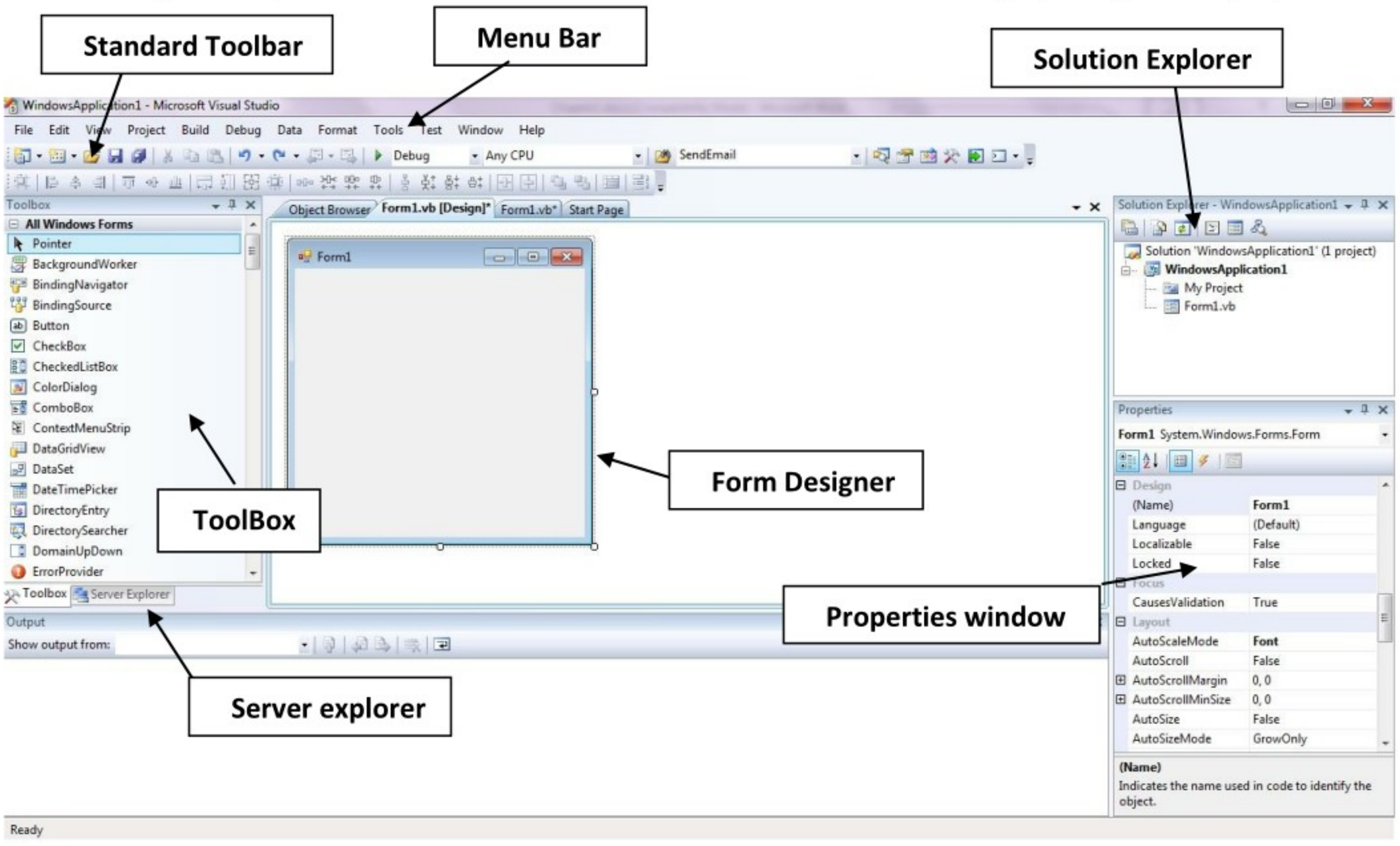


Fig 1.3 Visual Basic Integrated Development Environment (IDE)

- Visual Basic.net **Integrated Development Environment (IDE)** consist of Solution Explorer, Toolbox, Form, Properties Window, and Menu Bar which is shown in Fig 1.3.
- In Visual Studio windows related to a project are combined together and placed at certain locations on the screen. This type of IDE is known as Multiple Document Interface or MDI.

Menu System / Toolbars

- Menu bar consist of the commands that are used for constructing a software code. These commands are listed as menus and sub menus
- Menu bar also includes a simple Toolbar that lists the commonly used commands as buttons. This Toolbar can be customized, so that the buttons can be added as required.

The new Project dialog box

- Creating a new Visual Basic project is a simple task.
- You can either click **File→New→Project or File or Website command** or **the New Project link** from the Start Page. In both cases, Visual Studio shows the New Project window (See Fig 1.4)

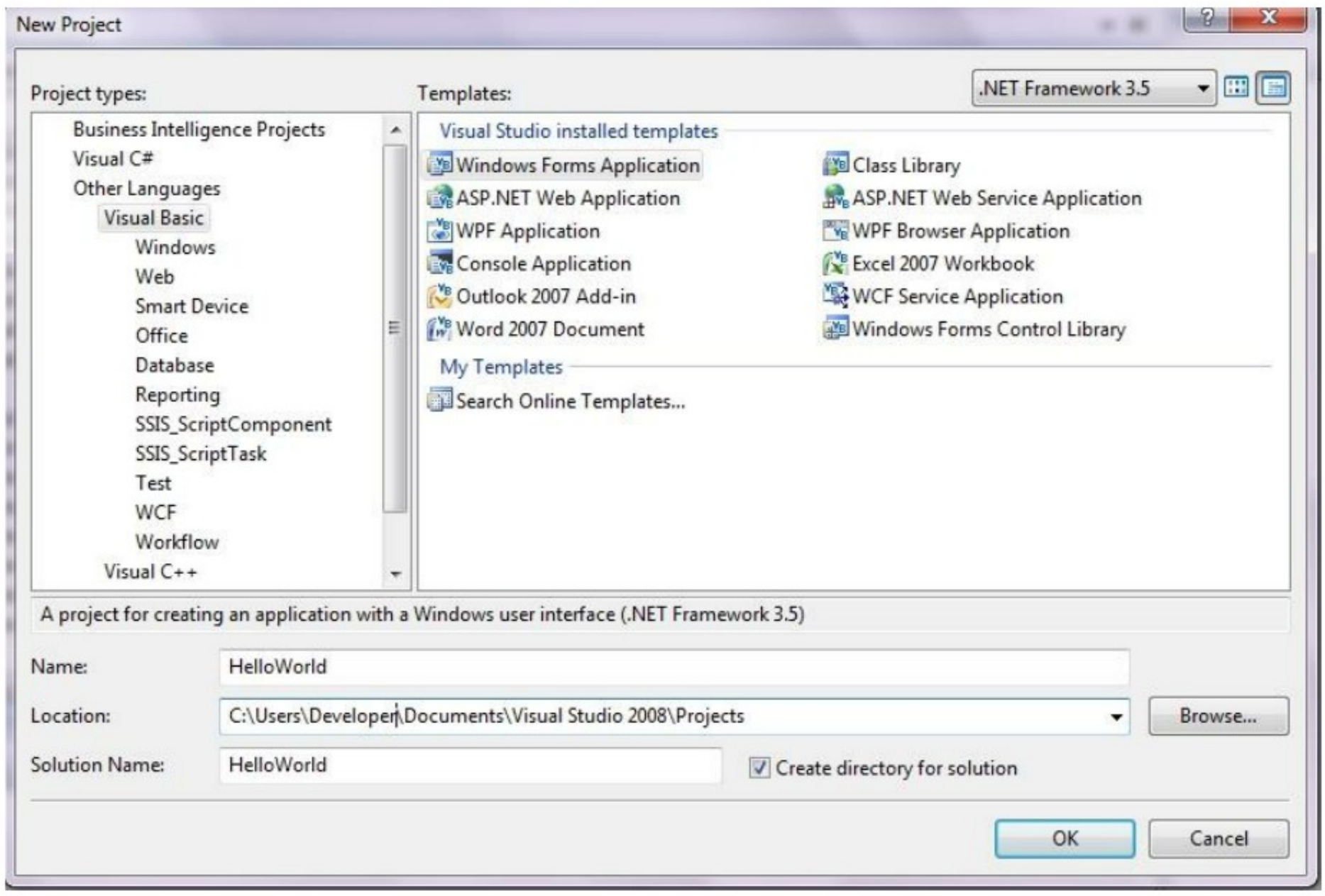


Fig 1.4 New Project window

- In Visual Studio, you can choose what version of the .NET Framework your application targets. This can be useful if you plan to develop applications with a high compatibility level and without new language features.
- After that enter name for the new project and click OK to create the project.

Graphical designers

- VB.Net programmers have made extensive use of forms to build user interfaces. Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag and drop controls from the Visual Studio Toolbox window.

Code designers

- The code editor in Visual Studio is now based on Windows Presentation Foundation. This innovation introduces new features to the Visual Basic code editor.
- We can focus on the zoom functionality in the code editor, the gradient selection, and IntelliSense.

IntelliSense

- IntelliSense is one of the most important feature in the coding experience with Visual Studio.
- IntelliSense is represented by a pop-up window that appears in the code editor each time you begin typing a keyword or an identifier and shows options for autocompleting words.

The Object Explorer/Browser

- The Object Browser is a special tool window that enables you to browse the .NET Framework class library.
- You can get a hierarchical view of the Base Class Library and of all the types defined in your solution, including types defined in referenced external assemblies.

The Toolbox

- Toolbox in Visual Basic.net consist of Controls, Containers, Menu Options, Crystal Report Controls, Data Controls, Dialogs, Components, Printing controls that are used in a form to design the interfaces of an application.

The Solution Explorer

- Solution Explorer is a special tool window that enables managing solutions, projects, and files in the projects or solution.
- It provides a complete view of what files compose your projects and enables adding or removing files and organizing files into subfolders.

Properties Window

- The Properties window allows user to view and modify the properties of the form and of the controls that it contains.
- You often need to set properties for your code, for .NET objects you use in your code, and for your files. To make things easier, Visual Studio provides a tool window named Properties window, which is a graphical tool for setting items' properties.

The Dynamic help window

- Visual Studio ships with the MSDN Library, which is the place where you can find documentation for Visual Basic version and the .NET Framework versions.

- There are basically two ways to access the MSDN Library: offline and online.
- You can also quickly find information on particular objects using built-in tools, such as the Object Browser.

The Server explorer

- Server Explorer is the server management console for Visual Studio.
- Use this window to open data connections and to log on to servers and explore their databases and system services.
- To access Server Explorer, choose Server Explorer on the View menu.

The Command window

- The .NET Framework allow you to access tools from the command prompt, so if you wish to use these utilities from any window command window, you will need to register these paths with the operating system.

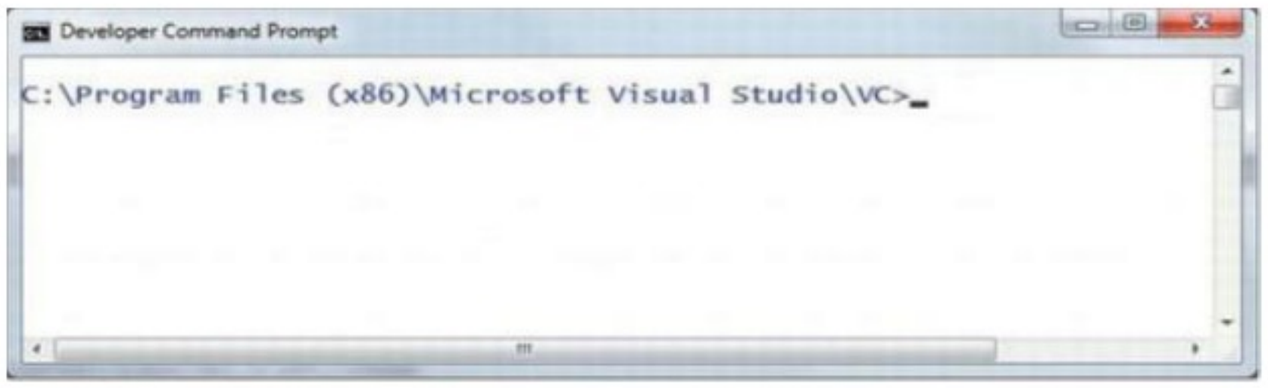


Fig 1.5 Command Prompt

Visual basic language concept:

Variables:

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in VB.Net has a specific type, which determines the size and layout of the variable's memory, the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The basic value types provided in VB.Net can be categorized as:

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char
Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

- There are some rules to choose variable name by the programmer.
  - Variable name **should not be a keyword**.
  - Variable name **cannot start with a digits**.
  - Upper case and lower case are distinct. Variable **Height** is not the same as **height** or **HEIGHT**.
  - **White space** is not allowed with variable Name.
  - Variable name can be any length.

Variable Declaration in VB.NET:

- The **Dim** statement is used for variable declaration and storage allocation for one or more variables.

  Syntax:

  Dim variablename As Datatype

  Example:

  Dim pi As Double

  - In above example, we declared a one variable "pi" and data type of that variable is Double.

Variable Initialization in VB.NET:

- The **Dim** statement is used for variable declaration and storage allocation for one or more variables.

  Syntax:

  Dim variablename As Datatype

  Example:

  Dim pi As Double = 3.14

  - In above example, we declared a one variable "pi" and data type of that variable is Double and assign it a value 3.14.

Accepting Values from User:

- The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

  Dim var as Integer

  var = Console.ReadLine()

Constant:

- The constants refer **to fixed values** that the program may not alter during its execution. These fixed values are also called **literals**.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.
- In VB.Net, constants are declared using the **Const** statement. The Const statement is used at module, class, structure, procedure, or block level for use in place of literal values.

  Syntax:

  Const Dim variablename As Datatype

  Example:

  Const Dim PI = 3.14149

  OR

  Const Dim pi As Double = 3.1415

DataTypes:

- Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

DataType	Storage Allocation	Value Range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point, 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.7976931348623157OE+308 through -4.94065645841246544E-324, for **negative values** 4.94065645841246544E-324 through 1.7976931348623157OE+308, for **positive values**
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	2 bytes	0 through 65,535 (unsigned)

Operators:

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Arithmetic Operators:

- Following table shows all the arithmetic operators supported by VB.Net. Assume variable A holds 2 and variable B holds 7, then:

Operator	Description	Example
^	Raises one operand to the power of another.	B^A will give 49
+	Adds two operands.	A + B will give 9
-	Subtracts second operand from the first.	A - B will give -5
*	Multiplies both operands.	A * B will give 14
/	Divides one operand by another and returns a floating point result.	B / A will give 3.5
\	Divides one operand by another and returns an integer result.	B \ A will give 3
MOD	Modulus Operator and remainder of after an integer division.	B MOD A will give 1

Comparison Operator:

- Following table shows all the comparison operators supported by VB.Net. Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A == B) is not true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A <> B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Logical/Bitwise Operators:

- Following table shows all the logical operators supported by VB.Net. Assume variable A holds Boolean value True and variable B holds Boolean value False, then:

Operator	Description	Example
And	It is the logical as well as bitwise AND operator. If both the operands are true, then condition becomes true.	(A And B) is False.

Or	It is the logical as well as bitwise OR operator. If any of the two operands is true, then condition becomes true.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand.	Not(A And B) is True.
Xor	It is the logical as well as bitwise Logical Exclusive OR operator. It returns True if both expressions are True or both expressions are False; otherwise, it returns False.	A Xor B is False.
AndAlso	It is the logical AND operator. It works only on Boolean data.	(A AndAlso B) is False.
OrElse	It is the logical OR operator. It works only on Boolean data.	(A OrElse B) is True.
IsFalse	It determines whether an expression is False.	
IsTrue	It determines whether an expression is True.	

Assignment Operator:

- There are following assignment operators supported by VB.Net:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand.	C -= A is equivalent to C = C - A
***=**	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division).	C /= A is equivalent to C = C / A
\=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (Integer division).	C \= A is equivalent to C = C \A
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	C^=A is equivalent to C = C ^ A

Bitshift Operator:

- The bit shift operators perform the shift operations on binary values.
- Bitwise operators work on bits and perform bit-by-bit operations. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Assume if A= 60 and B=13, now in binary format they will be as follows:

A = 0011 1100 A = 0011 1100 A = 0011 1100

A = 0011 1100 B = 0000 1101 B = 0000 1101 B = 0000 1101

---------- ---------- ---------- ----------

~A = 1100 0011 A&B = 0000 1100 A|B = 0011 1101 A^B = 0011 0001

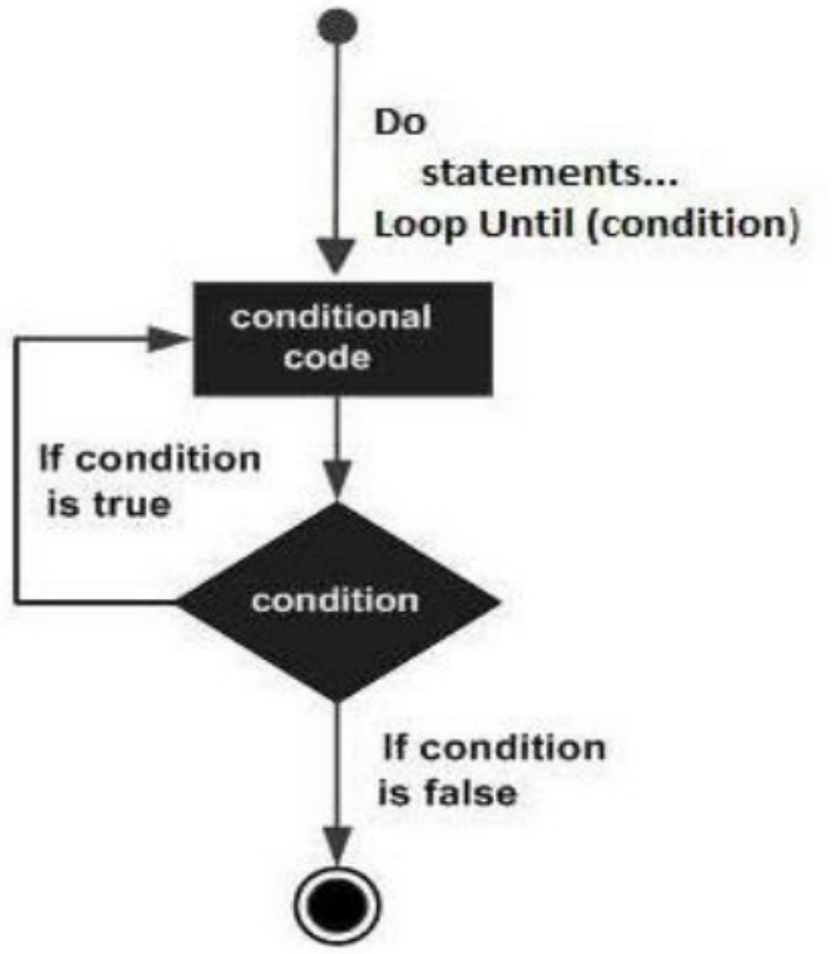
Operator	Description	Example
And	Bitwise AND Operator copies a bit to the result if it exists in both operands.	(A AND B) will give 12, which is 0000 1100
Or	Binary OR Operator copies a bit if it exists in either operand.	(A Or B) will give 61, which is 0011 1101
Xor	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A Xor B) will give 49, which is 0011 0001
Not	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(Not A) will give -61, which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

Loops:

- There may be a situation when you need to execute a block of code several number of times.
- **A loop statement allows us to execute a statement or group of statements multiple times** and following is the general form of a loop statement in most of the programming languages:

Do Loop:

- It repeats the enclosed block of statements while a Boolean condition is true or until the condition becomes true.
- It could be terminated at any time with the Exit Do statement.



- The syntax for this loop is as below:

```
Do { While | Until } condition
     [statements]
     [continue Do ]
     [statements]
     [Exit Do]
     [statements]
Loop
condition
```

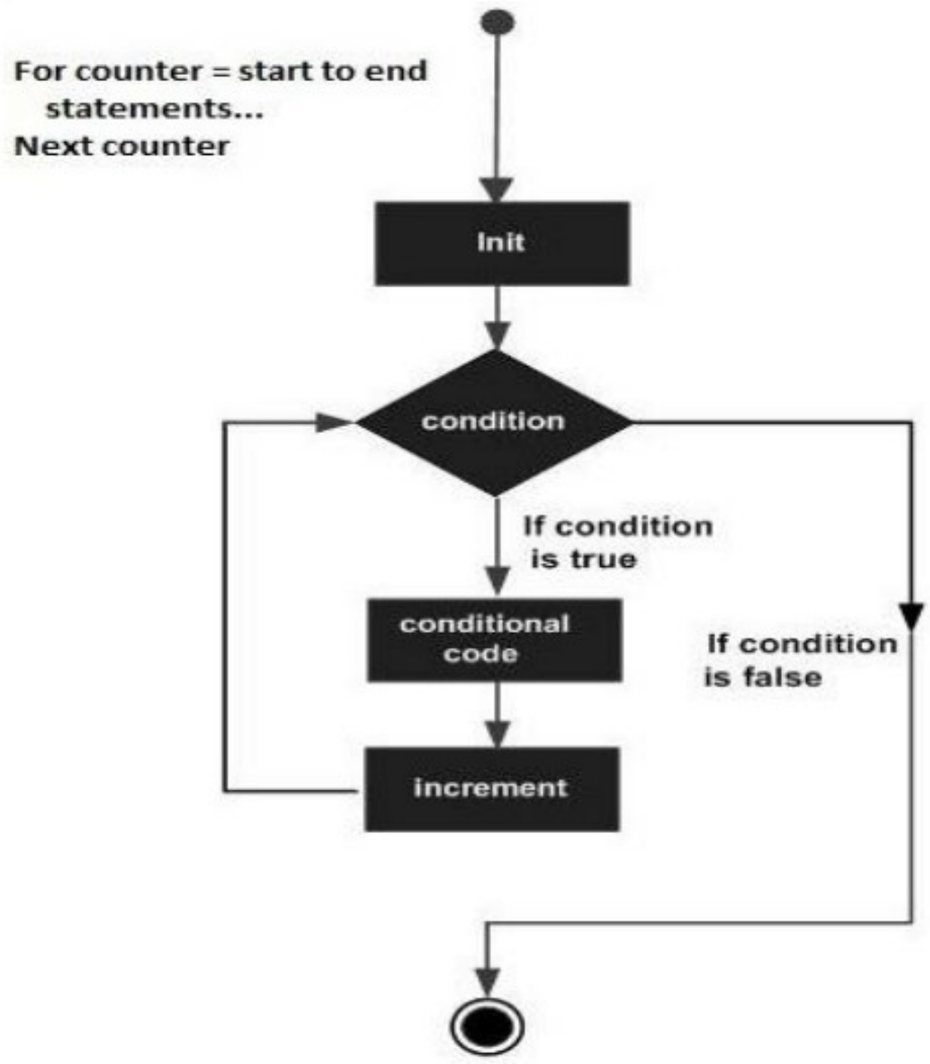
```
        Do
         [statements]
         [Continue Do]
         [statements]
         [Exit Do]
         [statements]
   Loop { While | Until}
```

- Example of the Do Loop is as following;

```
Module loops
   Sub Main()
      ' local variable definition
      Dim a As Integer = 10
      'do loop execution
      Do
          Console.WriteLine("value of a: {0}", a)
          a = a + 1
      Loop While (a < 20)
      Console.ReadLine()
   End Sub
End Module
```

For...Next Loop:

- It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.



- The syntax for this loop construct is:

```
For counter [ As datatype ] = start to end [Step step ]
     [statements]
     [Continue For]
     [statements]
     [ Exit For ]
     [statements]
Next [counter]
```

- Example of the For..Next Loop is as following:

```
Module loops
   Sub Main()
      Dim a As Byte
      ' for loop execution
      For a = 10 To 20
          Console.WriteLine("value of a: {0}", a)
      Next
      Console.ReadLine()
   End Sub
End Module
```

For Each… Next Loop:

- It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
- The syntax for this loop construct is:

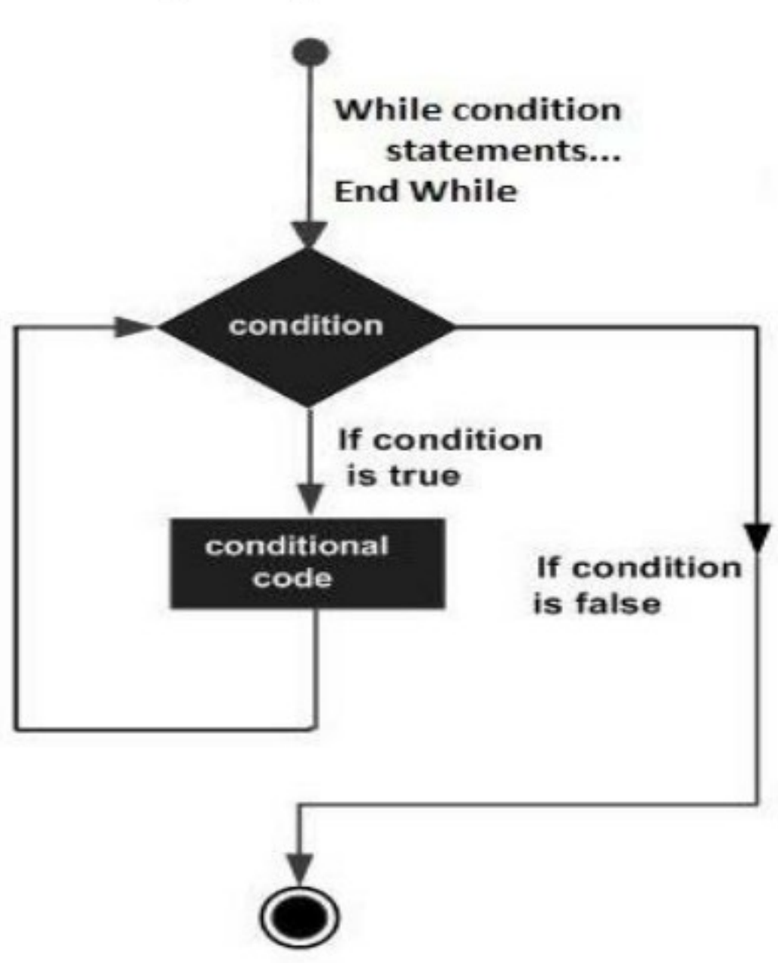
```
For Each element [ As datatype ] In group
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ element ]
```

- Example of For Each…Next Loop

```
Module loops
   Sub Main()
      Dim anArray() As Integer = {1, 3, 5, 7, 9}
      Dim arrayItem As Integer
     'displaying the values
      For Each arrayItem In anArray
          Console.WriteLine(arrayItem)
  Next
      Console.ReadLine()
   End Sub
End Module
```

While…End While Loop:

- It executes a series of statements as long as a given condition is true.



- Key point of the While loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.
- The syntax for this loop construct is as below.
- Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is logical true. The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.

```
While condition
    [ statements ]
    [ Continue While ]
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

- Example of While…End While Loop

```
Module loops
   Sub Main()
      Dim a As Integer = 10
      ' while loop execution '
      While a < 20
          Console.WriteLine("value of a: {0}", a)
          a = a + 1
      End While
      Console.ReadLine()
   End Sub
End Module
```

With…End With Statement:

- It is not exactly a looping construct. It executes a series of statements that repeatedly refers to a single object or structure.
- The syntax for this loop construct is:

```
With object
     [statements]
End With
```

- Example of this loop Is as follows:

```
Module Module1
    Public Class Book
        Public Name As String
        Public author As String
        Public Subject As String
    End Class
```

```
    Sub Main()
        Dim obj As New Book
        With obj
            .Name = ".Net Programming"
            .author = "ABCD-XYZ"
            .Subject = "VB.NET"
        End With
        With obj
            Console.WriteLine(.Name)
            Console.WriteLine(.author)
            Console.WriteLine(.Subject)
        End With
        Console.ReadKey()
    End Sub
End Module
```

Nested…Loop:

- The syntax for a nested For loop statement in VB.Net is as follows:

```
For counter [ As datatype1 ] = start1 To end1 [ Step step1 ]
    For counter [ As datatype2 ] = start2 To end2 [Step step2 ]
    ……
    Next [ counter2]
Next [ counter1 ]
```

- The syntax for a nested While loop statement in VB.Net is as follows:

```
While conditional1
    While condition2
    ……
    End While
End While
```

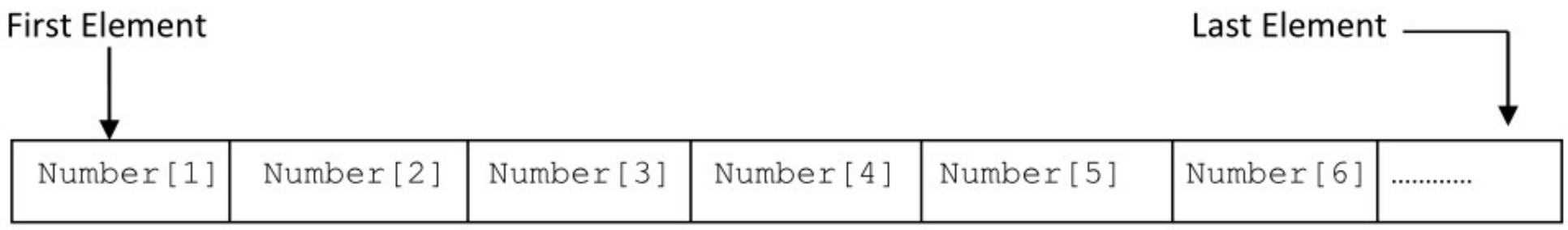
- The syntax for a nested Do...While loop statement in VB.Net is as follows:

```
 Do { While  | Until } condition1
     Do { While | Until  } condition2
     ……
     Loop
Loop
```

Array:

- An **array stores a fixed-size sequential collection of elements of the same type.**
- All arrays consist of contiguous memory locations.

- The lowest address corresponds to the first element and the highest address to the last element.



Creating Arrays in VB.NET:

- To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)As Integer ' an array of 31 elements
Dim strData(20) As String ' an array of 21 strings
Dim twoDarray(10, 20) As Integer 'a two dimensional array of
integers
Dim ranges(10, 100) 'a two dimensional array
```

- You can also initialize the array elements while declaring the array. For example,

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim branches() As String = {"CE","IT","EC","MECH","CIVIL"}
```

- The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
Module arrayApl
   Sub Main()
      Dim n(10) As Integer  ' n is an array of 11 integers '
      Dim i, j As Integer   ' initialize elements of array n '
      For i = 0 To 10
          n(i) = i + 1      ' set element at location i to i + 100
      Next i                ' output each array element's value '
      For j = 0 To 10
          Console.WriteLine("Element({0}) = {1}", j, n(j))
      Next j
      Console.ReadKey()
   End Sub
End Module
```

Multidimensional Array:

- VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays.
- You can declare a 2-dimensional array of strings as:

  Dim twodisplay(10,20) As String

- or, a 3-dimensional array of Integer variables:

  Dim threedispalay(10,20,30) As Integer

- The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
   Sub Main()
      ' an array with 5 rows and 2 columns
      Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4,
8}}
      Dim i, j As Integer
      ' output each array element's value '
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
          Next j
      Next i
      Console.ReadKey()      End Sub    End Module
```

Dynamic Array:

- Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the ReDim statement.
- Syntax for ReDim statement:

  ReDim [Preserve] arrayname (subscripts)

- Where,
  - The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
  - **arrayname** is the name of the array to re-dimension.
  - **subscripts** specifies the new dimension.

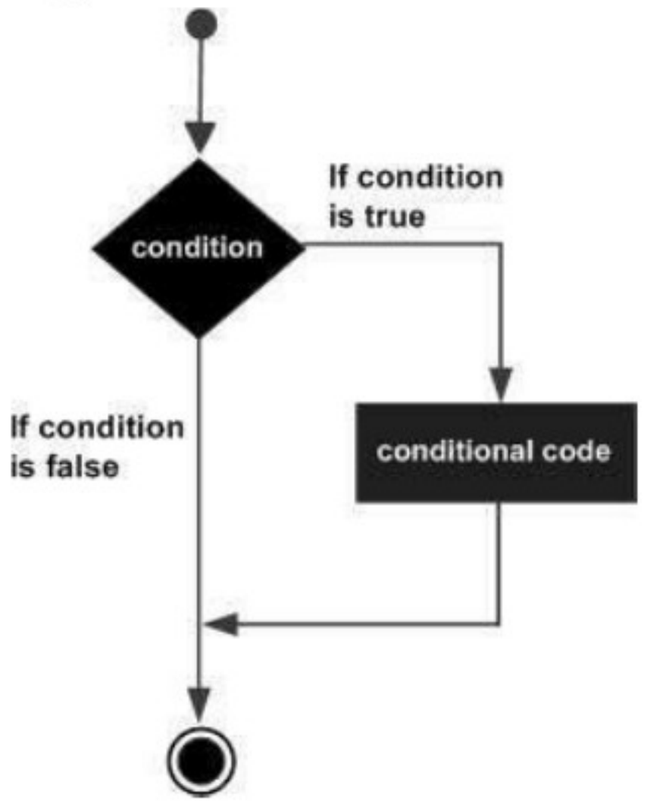
```
Module arrayApl
   Sub Main()
      Dim marks() As Integer
      ReDim marks(2)
      marks(0) = 85
      marks(1) = 75
      marks(2) = 90
      ReDim Preserve marks(10)
      marks(3) = 80
      marks(4) = 76
      marks(5) = 92
      marks(6) = 99
      marks(7) = 79
      marks(8) = 75
      For i = 0 To 10
          Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
   End Sub
End Module
```

Conditional Structure / Decision Making:

- The conditional structure helps to conditionally execute a group of statements or statement, depending on the value of an expression.
- VB.Net provides the following types of decision making statements which given below:
  - If...Then Statement
  - If...Then...Else Statement
  - Nested If Statements
  - Select Case Statement

If...Then Statement:

- It is the simplest form of control statement.
- It frequently used in decision making and changing the control flow of the program execution.
- The Flow chart for If... Then statement is given below:



- The syntax for this statement is:

```
If condition Then
      [Statement(s)]
End If
```

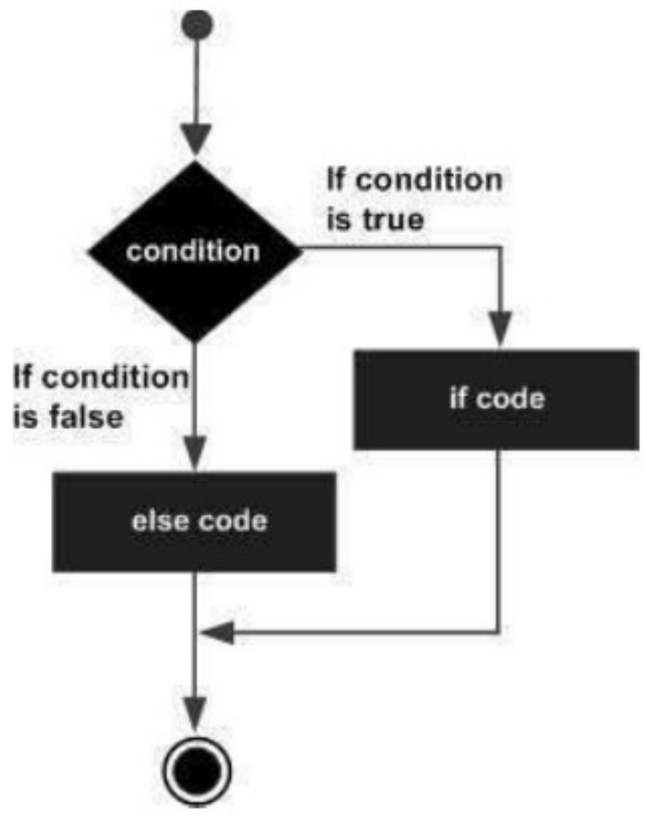
- Example of for If... Then statement is:

```
Module loops
   Sub Main()
      Dim a As Integer = 0
      ' check the boolean condition using if statement
      If (a < 1) Then
            Console.WriteLine("a is less than 1")
       End If
        Console.WriteLine("value of a is : {0}", a)
        Console.ReadLine()
   End Sub
End Module
```

- If the condition **(a < 1) evaluates to true**, then the block of **code inside the If statement** will be executed.
- If condition **(a < 1) evaluates to false**, then the code after the **End If** of the If statement will be executed.

If...Then...Else Statement:

- It conditionally executes a group of statements, depending on the value of an expression.
- If the Boolean expression evaluates to true, then the block of **If** part will be executed, otherwise the block of **Else** part will be executed.
- The Flow chart for If... Then...Else statement is given below:



- The syntax for this statement is:

```
If(boolean_expression or condition) Then
   'statement(s) will execute if the Boolean expression is true
Else
  'statement(s) will execute if the Boolean expression is false
End If
```

- Example of for If... Then...Else statement is:

```
Module loops
   Sub Main()
      Dim a As Integer = 0
      ' check the boolean condition using if statement
      If (a < 1) Then
            Console.WriteLine("a is less than 1")
       Else
            Console.WriteLine("a is not less than 1")
       End If
        Console.WriteLine("value of a is : {0}", a)
        Console.ReadLine()
   End Sub
End Module
```

If...Else If...Else Statement

- The If statement can be followed by an optional Else if...Else statement, which is used to test various conditions using single If...Else If statement.
- It useful when you want to check more than one condition in a program.
- The syntax for this statement is:

```
If(boolean_expression 1 or condition 1)Then
   ' Executes when the boolean expression 1 is true
ElseIf( boolean_expression 2 or condition 2)Then
   ' Executes when the boolean expression 2 is true
ElseIf( boolean_expression 3 or condition 3)Then
   ' Executes when the boolean expression 3 is true
Else
   ' executes when the none of the above condition is true
End If
```

- Example of for if...else if... else statement is:

```
Module decisions
   Sub Main()
      Dim a As Integer = 10
      If (a = 1) Then
          Console.WriteLine("Value of a is 1")
      ElseIf (a = 2) Then
          Console.WriteLine("Value of a is 2")
      ElseIf (a = 3) Then
          Console.WriteLine("Value of a is 3")
      Else
          Console.WriteLine("No values is matching")
      End If
      Console.WriteLine("Exact value of a is: {0}", a)
      Console.ReadLine()
   End Sub     End Module
```

Nested If Statements

- By using If-Then-Else statements, you can use one If or ElseIf statement inside another If ElseIf statement(s).
- The syntax for this statement is:

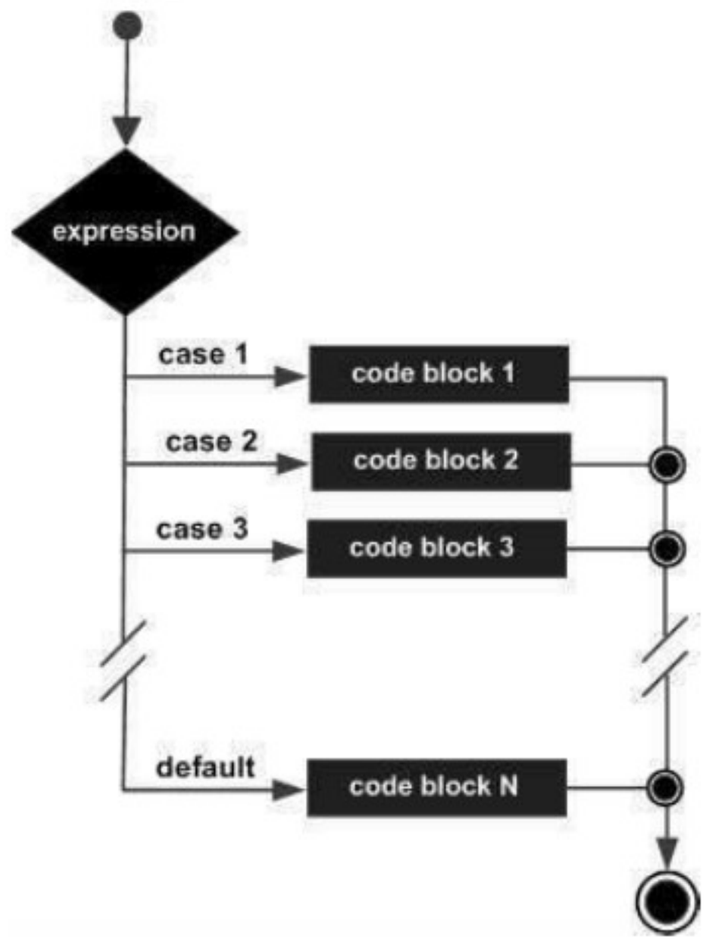
```
If( boolean_expression 1)Then
   'Executes when the boolean expression 1 is true
   If(boolean_expression 2)Then
         'Executes when the boolean expression 2 is true
   End If
End If
```

- Example of for Nested If statement is:

```
Module decisions
   Sub Main()
      Dim a As Integer = 1
      Dim b As Integer = 2
      If (a = 1) Then
        If (b = 2) Then
            Console.WriteLine("Value of a is 1 and b is 2")
        End If
      End If
      Console.WriteLine("Exact value of a is : {0}", a)
      Console.WriteLine("Exact value of b is : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

Select Case Statement:

- In **Select Case statement,** it allows a variable to be tested for equality against a given list of values.
- Each value is called a **case**, and the variable being switched on is checked for each select case.
- The Flow chart for select case statement is given below:



- The syntax for this statement is:

```
Select [Case] expression
    [Case expressionlist
        [statements]]
    [Case Else
        [else statements]]
End Select
```

- Example of for select case statement is:

```
Module decisions
   Sub Main()
      Dim grade As Char
      grade = "B"
      Select grade
          Case "A"
              Console.WriteLine("Excellent!")
          Case "B", "C"
              Console.WriteLine("Very Good")
          Case "D"
              Console.WriteLine("Good")
          Case "F"
              Console.WriteLine("Better try again")
          Case Else
              Console.WriteLine("Invalid grade")
      End Select
      Console.WriteLine("Your grade is  {0}", grade)
      Console.ReadLine()
   End Sub
End Module
```