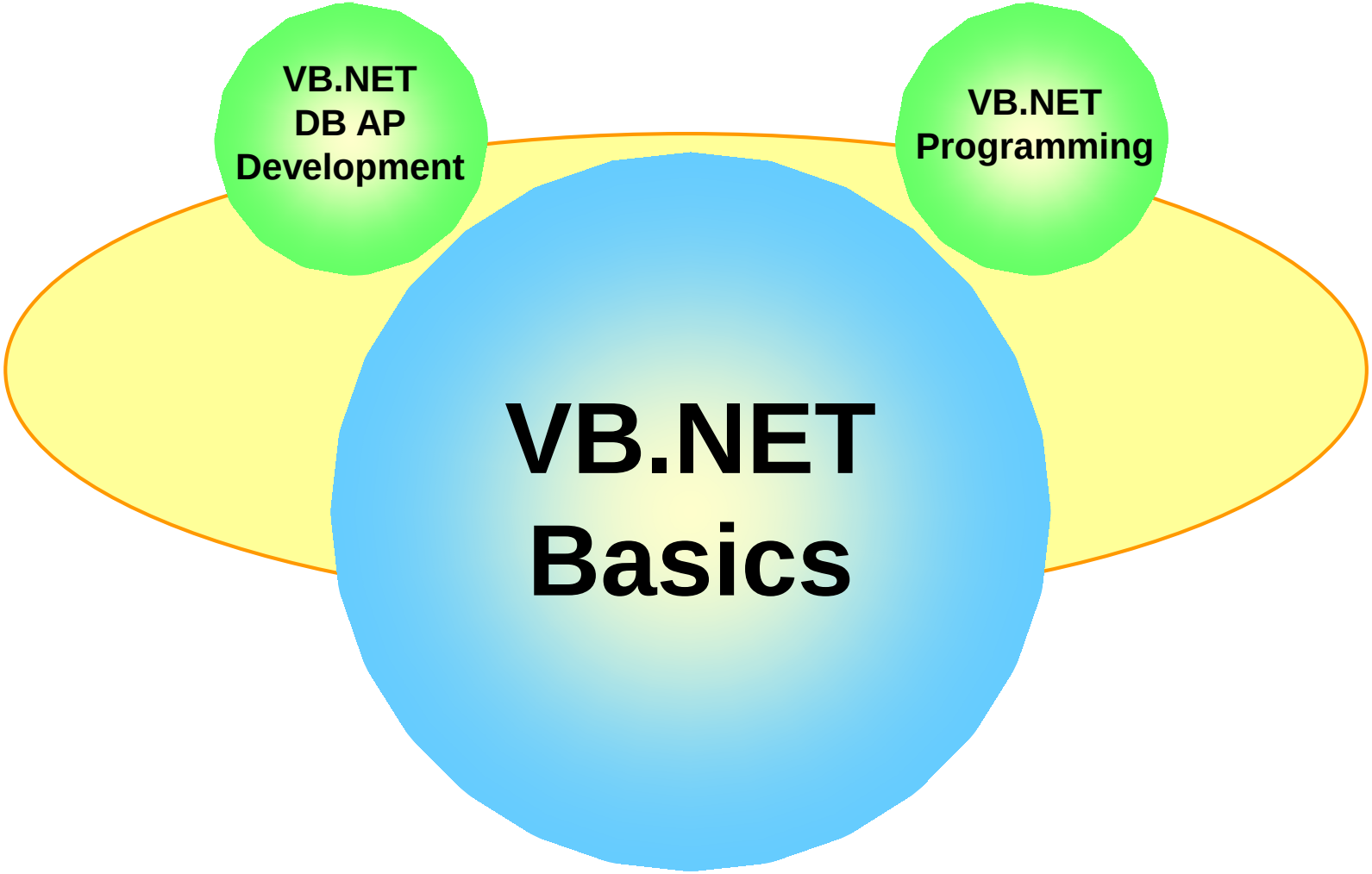
Contents

VB.NET Basics



Objectives

Upon completion of this subject, you are expected to :

1. Understand main components of .NET Framework.
2. Explain how to use tools provided by Visual Studio .NET.
3. Explain how to make basic applications using VB.NET.
4. Explain Object Oriented Programming concepts of VB.NET.

Contents

1. Outline of Visual Basic .NET

1.1 Outline of .NET Framework

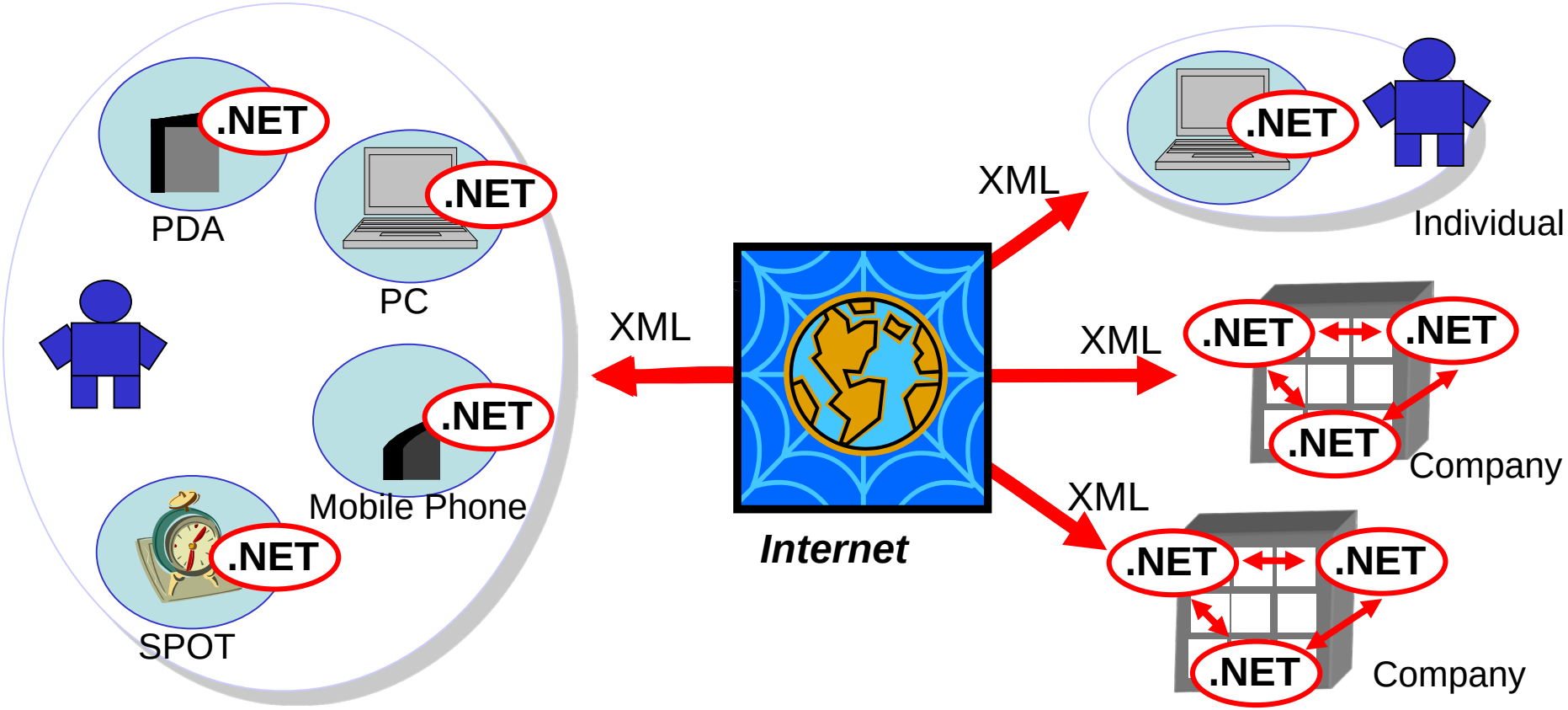
.NET Framework is the platform that supports the development and execution of software that uses Microsoft® software .NET technology

[Characteristics of .NET Framework]

- By adopting a CLR that provides strong functionality, a robust application can be easily created
- Programming productivity increases due to the abundant class libraries that have been provided
- The programmer can apply his various skills, irrespective of the language.

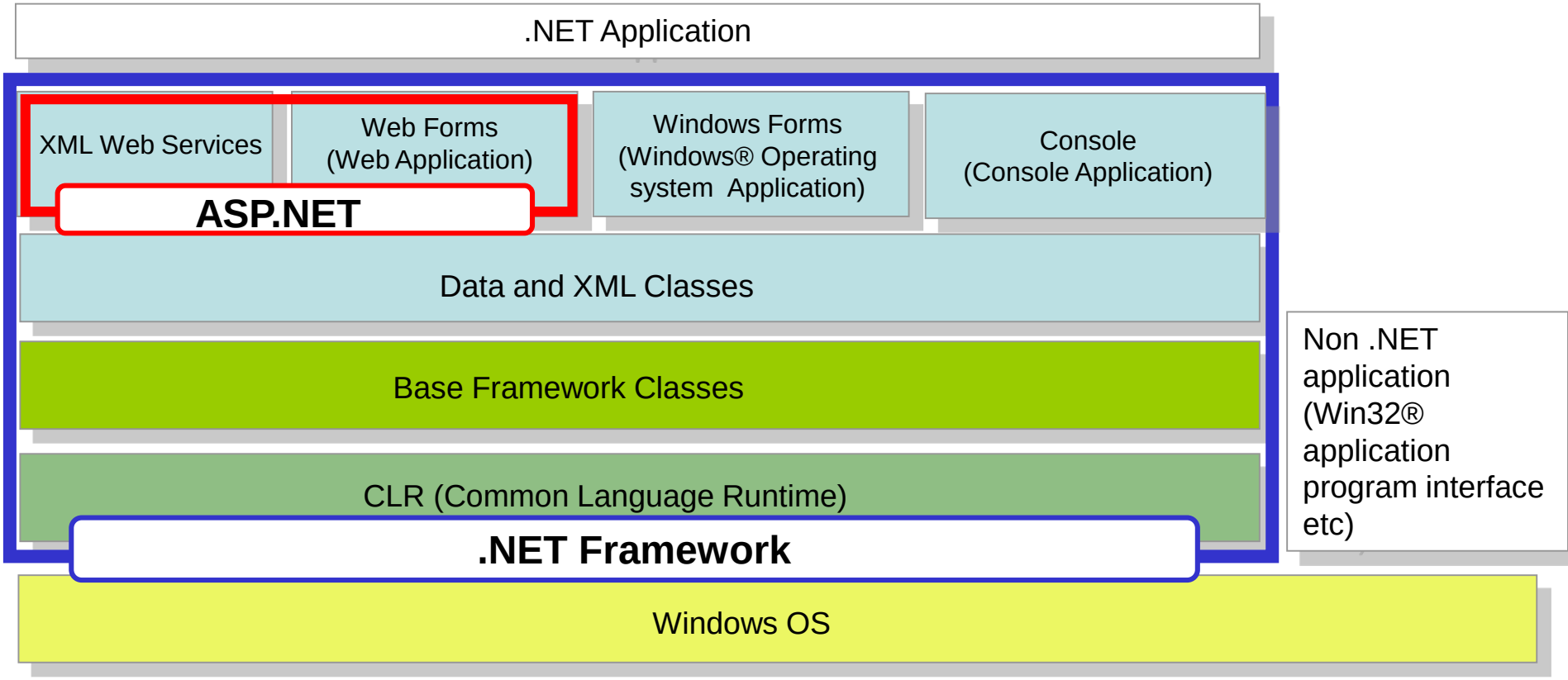
.NET Framework (1/2)

Microsoft .NET refers to the next generation of internet technology that is promoted by Microsoft.



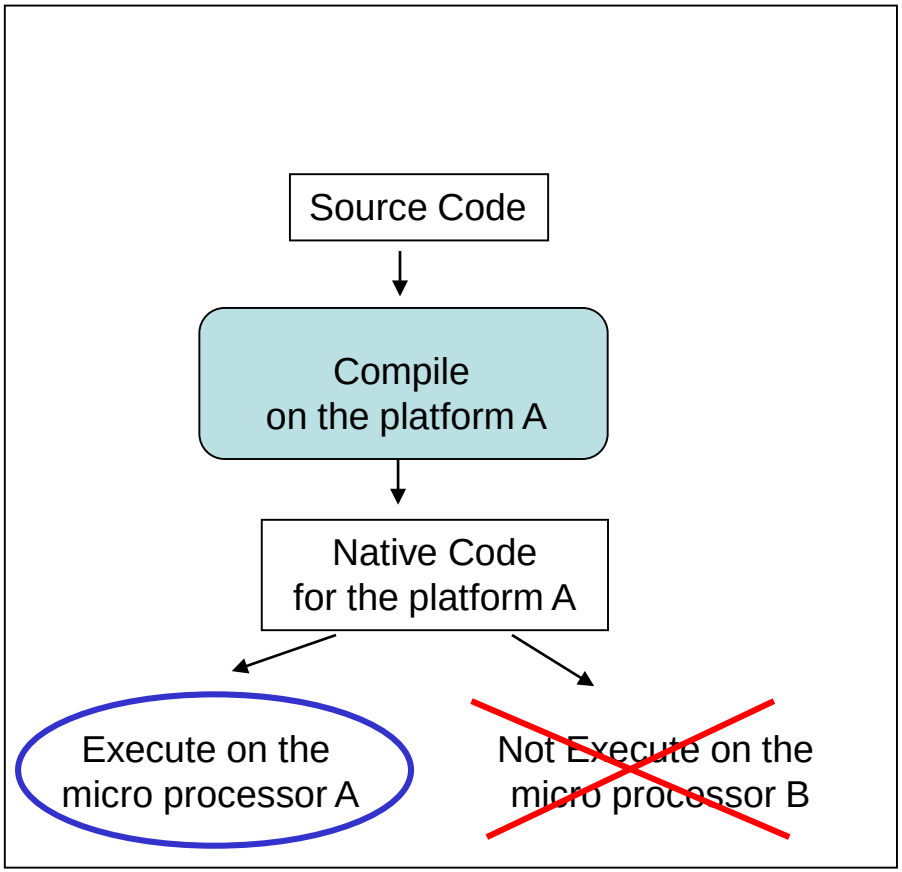
.NET Framework (2/2)

.NET Framework refers to the platform that supports the development and execution of software that uses Microsoft .NET technology

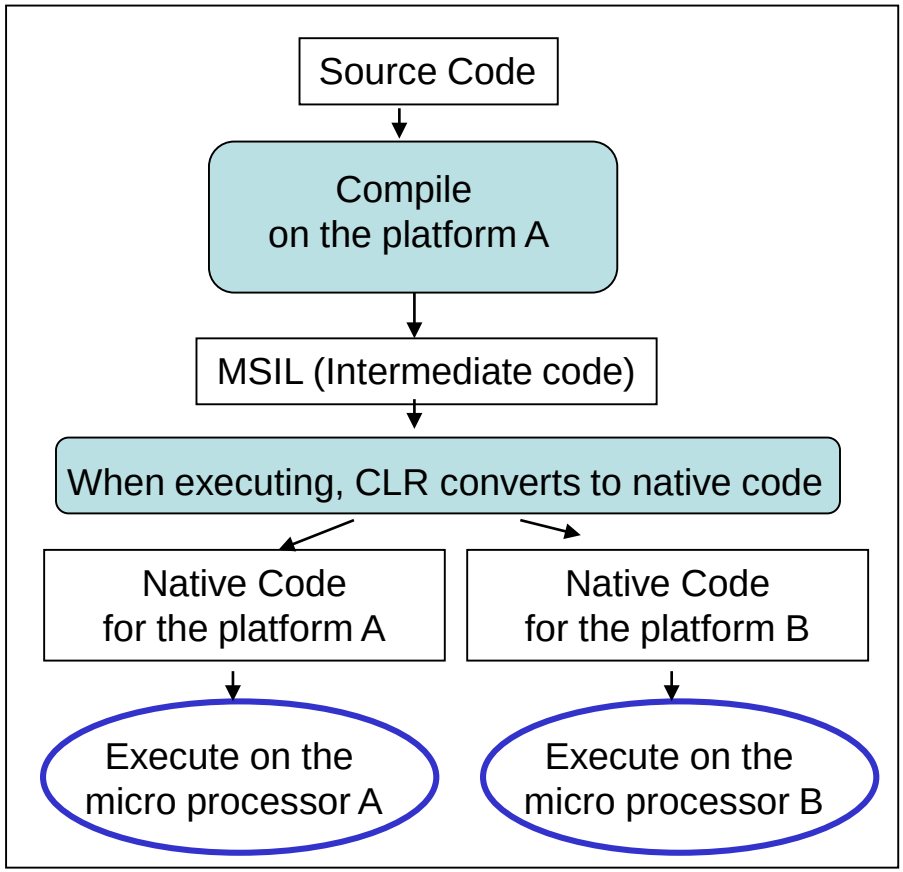


CLR (Common Language Runtime) (1/3)

CLR is the engine used for loading and executing .NET application



Conventional Program



.NET Program

CLR (Common Language Runtime) (2/3)

CLR provides a feature like Garbage Collector.

- Garbage Collector is the memory management functionality provided by CLR
  - Perform the allocation and release of memory that is used by the application

- Garbage Collection refers to the automatic release by Garbage Collector, of object memory that is no longer used by the application

CLR (Common Language Runtime) (3/3)

The .NET application that is run on CLR is known as Managed Code

- Applications that are not run on CLR are known as Unmanaged Code
  - →For example, Win32 applications or Visual Basic® Development system 6.0 applications etc.

.NET Framework Class Library

In .NET Framework, each type of Managed Code oriented software component is provided as a class library

- Base Framework Library
- Data and XML Classes
- XML Web Services, Web Forms, Windows Forms, Console

Non dependency on Languages

Examples of programming languages that can be used in .NET Framework

Language	Main Characteristics
Visual Basic® Development system .NET	The programming language in succession to Visual Basic 6.0. To enable multiple language compliance in the .NET Framework environment, language specifications have been expanded widely, and object oriented programming is supported.
Visual C++® Development system .NET	The programming language in succession to Visual C++ ® Development sytem6.0. Among the programming languages supported by Visual Studio.NET, it is the sole language that can create an unmanaged code.
Visual C# ® Development tool	It is the programming language that supports the newly provided object oriented technology for Microsoft .NET platform. However, practically speaking, there are language specifications that are similar to Java.
Visual J# ® Development tool	The programming language in succession to Visual J++ ® Development system 6.0.

1.2 Introduction to Visual Basic .NET

- The programming language in succession to Visual Basic

- To enable multiple language compliance in the .NET Framework environment, language specifications have been expanded widely, resulting in a full-fledged Object Oriented programming language

Characteristics of Visual Basic .NET (1/2)

Visual Basic .NET is the programming language developed with the objective of enabling the easy creation of a large scale, and moreover, a high performance system.

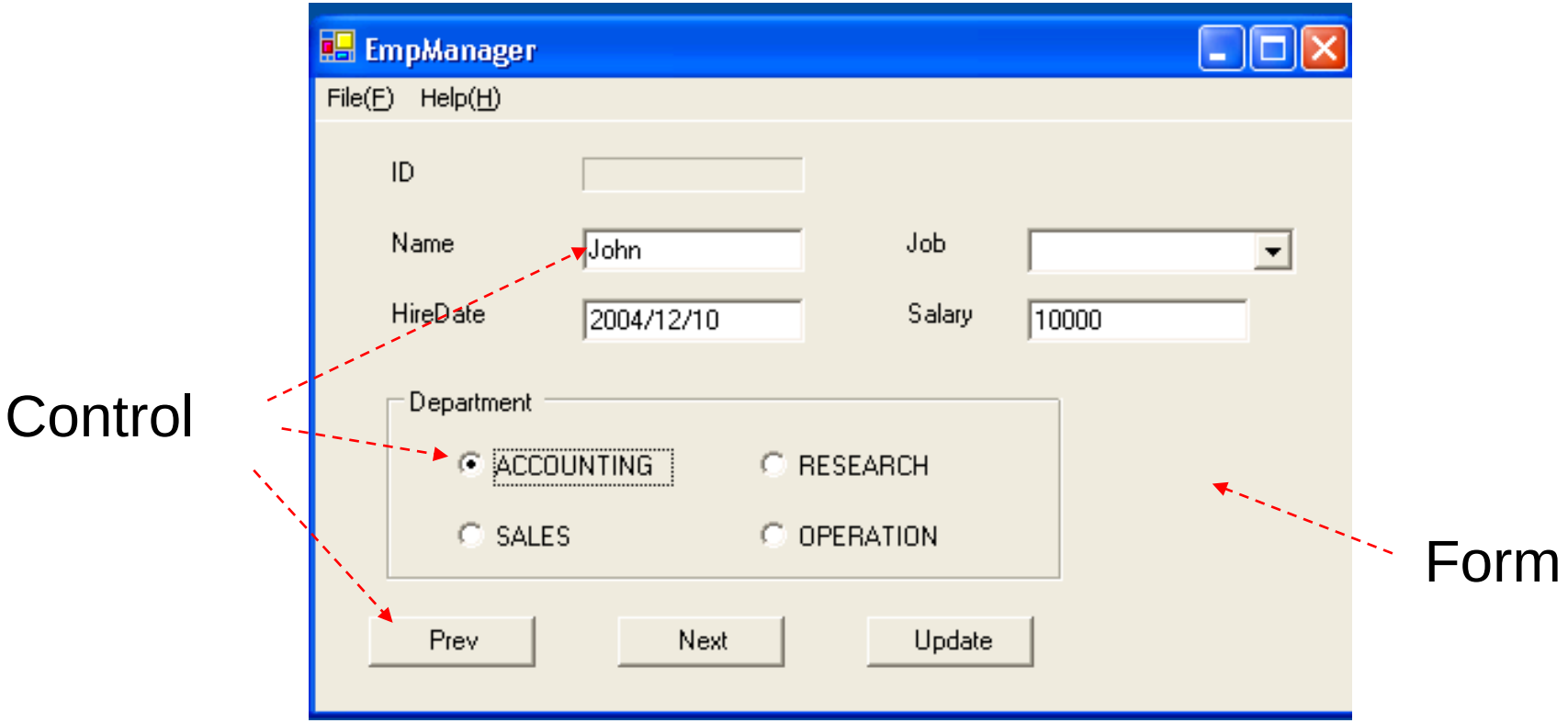
- The various applications that can be created
  - Windows Application
  - Console Application
  - Web Application
  - XML Web Service
- Object Oriented support
  - Inheritance of code
  - Overload
  - Override
  - Constructor

Characteristics of Visual Basic .NET (2/2)

- Exception handling support
- By using the various functions provided in .NET Framework, it becomes possible to create a durable application
- Deployment of the created program is easy
- Enables Inter-usability with other programming languages (Visual C++.NET,C# etc) that are supported by .NET Framework
- However, there isn't 100% compatibility with Visual Basic 6.0

Windows Application

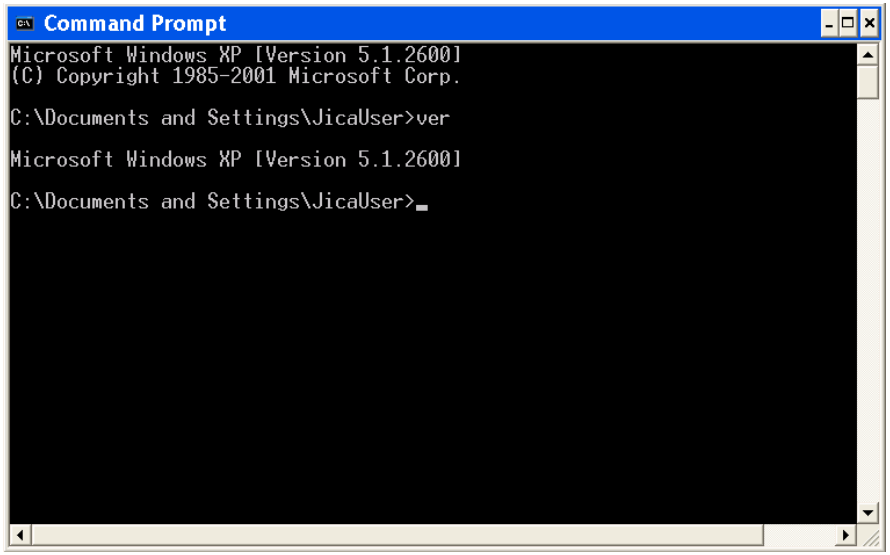
Windows application is the application that has a rich user interface functionality that uses the Windows form



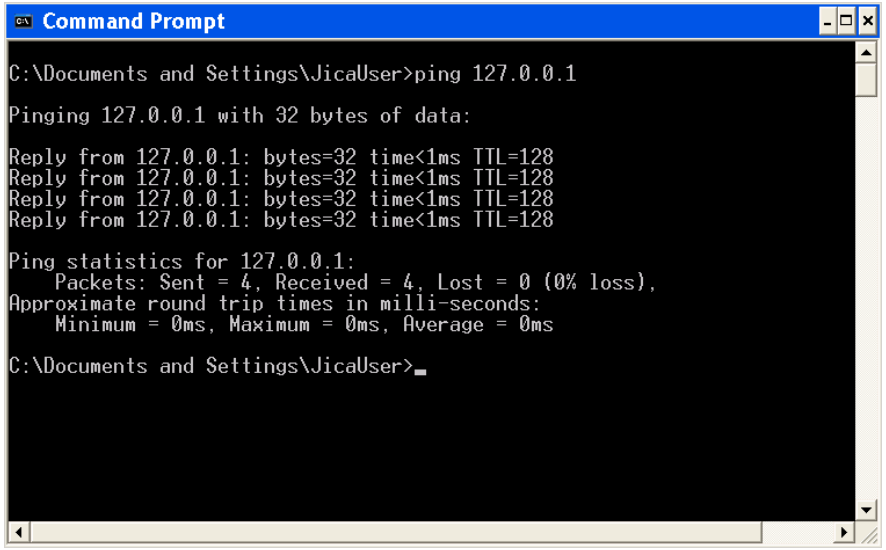
Console Application

Console application refers to the application that does not create the window and is used from Command Prompt

(example 1) ver command



(example 2) ping command



Create and Run Console Application (1/4)

One can create a console application such that when the program is run, “Please Input Words” is displayed and after entering a string, the program displays the entered string after ”Your Input Words:”

<Console screen image>

```
>Program is run
Please Input Words
Hello World!!
Your Input Words : Hello World!!

>Program terminated
```

Display when Program is run.

User enters a string

The entered string is displayed

Create and Run Console application (2/4)

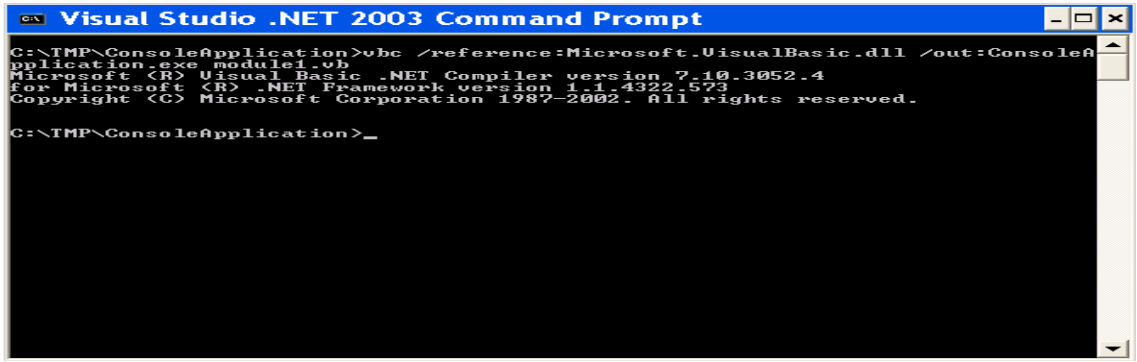
1) Create the following code in notepad,
and save by the file name “module1.vb”.

```
Imports System

Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Please Input Words")
        str = Console.ReadLine()
        Console.WriteLine("Your Input Words : {0}", str)
    End Sub

End Module
```

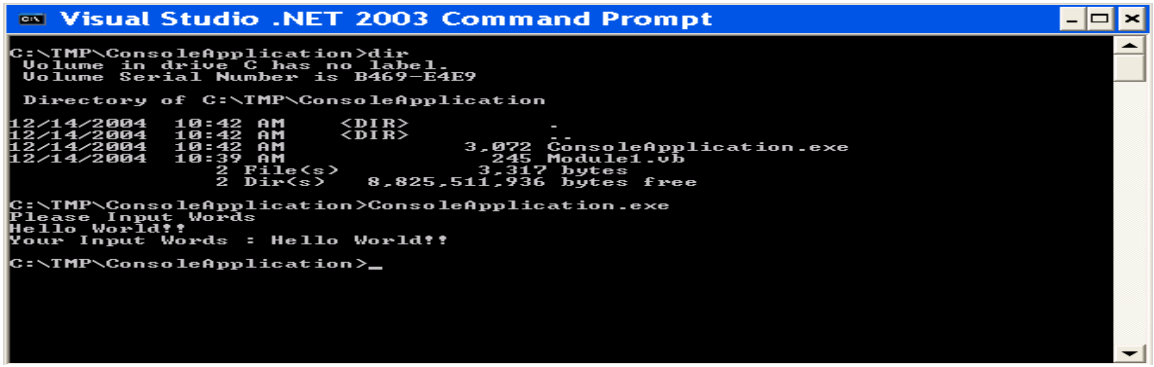
- Generally, in Visual Basic .NET, file extension of the defined source code is “.vb”.

Create and Run Console Application (3/4)

•Compile Command
>vbc /reference:Microsoft.VisualBasic.dll /out:ConsoleApplication.exe module1.vb



2) Run Visual Studio .NET 2003 Command Prompt, move to the directory in which the module1.vb file exists, and run compile command.



3) ConsoleApplication.exe file can be created in the directory in which module1.vb file exists. Thus, run it from Command Prompt.

Create and Run Console Application (4/4)

• Visual Studio .NET 2003 Command Prompt is run in the following order.
[Start]menu – [All Programs] – [Microsoft Visual Studio .NET 2003] – [Visual Studio .NET Tools] – [Visual Studio .NET 2003 Command Prompt]

• When compiling from command line, as it is necessary to explicitly refer to Microsoft Visual Basic runtime library, it is also required to specify Microsoft.VisualBasic.dll in the /reference option.

• To run the created ConsoleApplication.exe, after opening a new Command Prompt, and after moving it to the directory where ConsoleApplication.exe exists, run the following command
">ConsoleApplication.exe"

• Even if console application is directly run from the created ConsoleApplication.exe file, as the console closes when program execution is complete, it is not possible to confirm execution results. Therefore, it is necessary to run the program on Command Prompt

• In vbc command of the Visual Studio .NET 2003 Command Prompt, it is possible to compile the program that was created in Visual Basic .NET from command line, and create an executable file (exe file).

• By specifying ">vbc /?", it is possible to refer to Help of vbc command. For an explanation on the Vbc command option, refer to Help of the vbc comment.

Other Applications that can be created

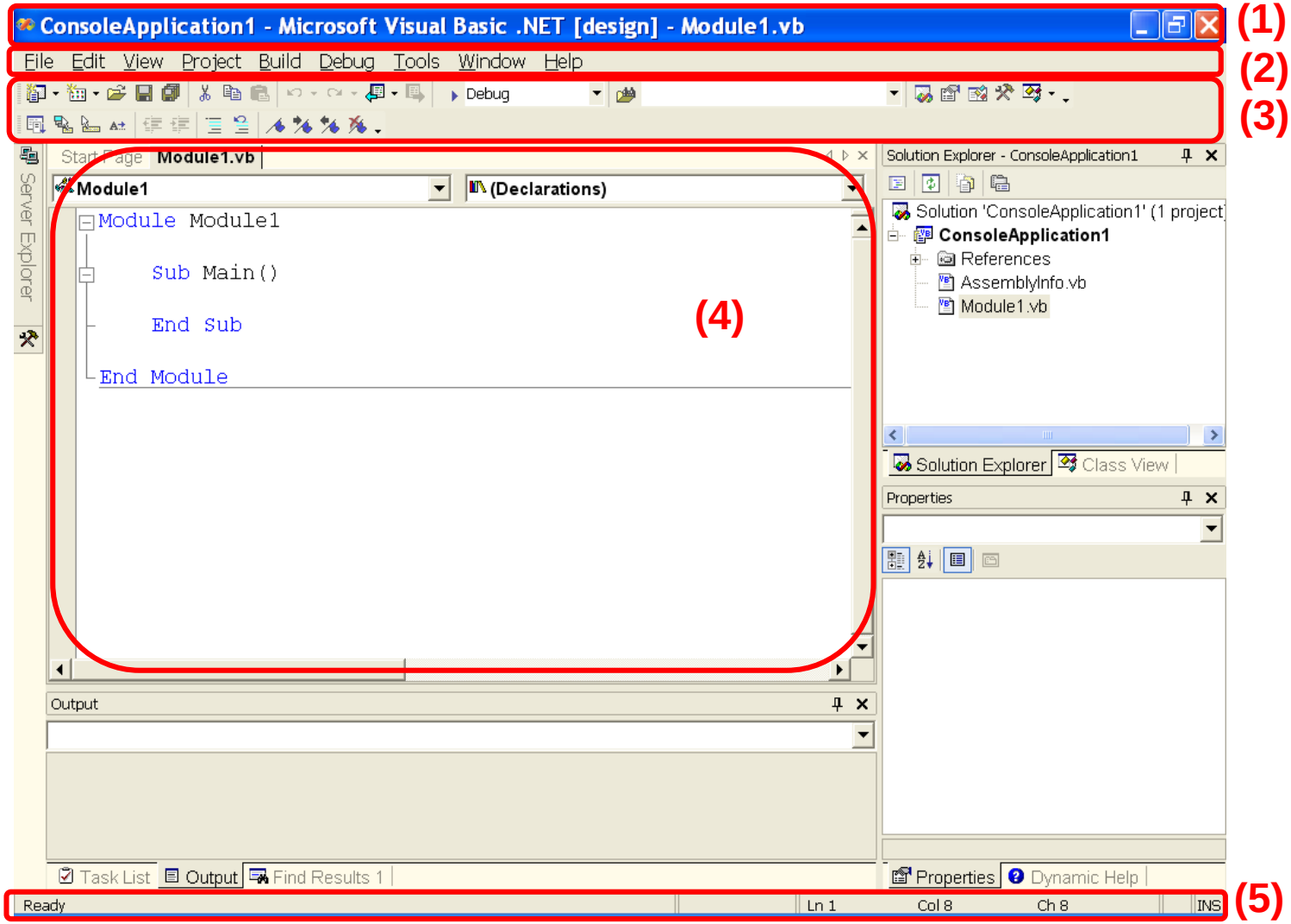
- Web Application
  Refers to the application that provides dynamic contents when requests are made from web browser

- XML Web Service
  Refers to the linking of autonomous applications on the network using SOAP/XML format message conversion

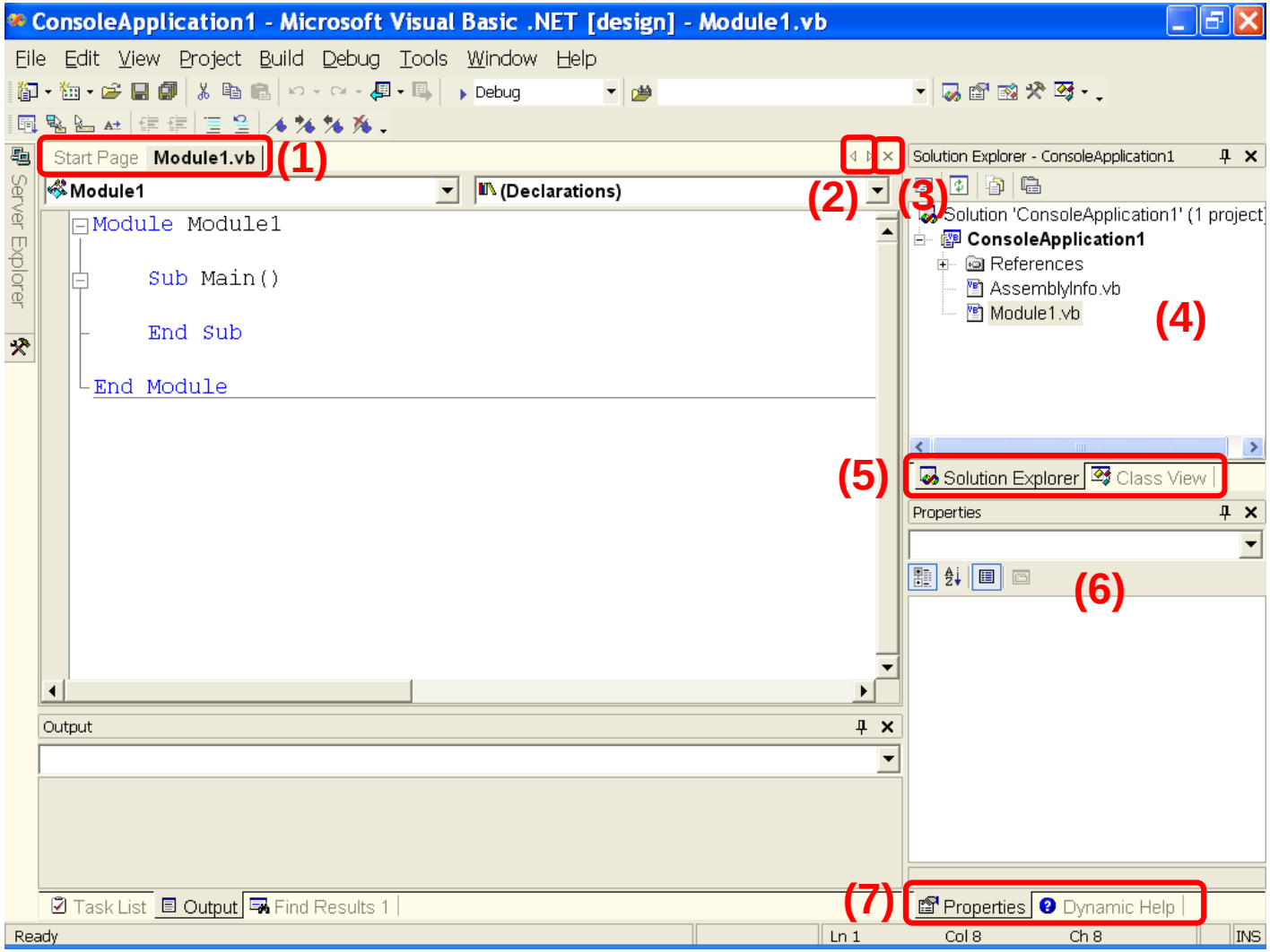
1.3 Introduction to Visual Studio .NET

It is the development tool provided by Microsoft, where, by using this development tool, it becomes possible to create quickly and easily, various programs that are .NET Framework compliant

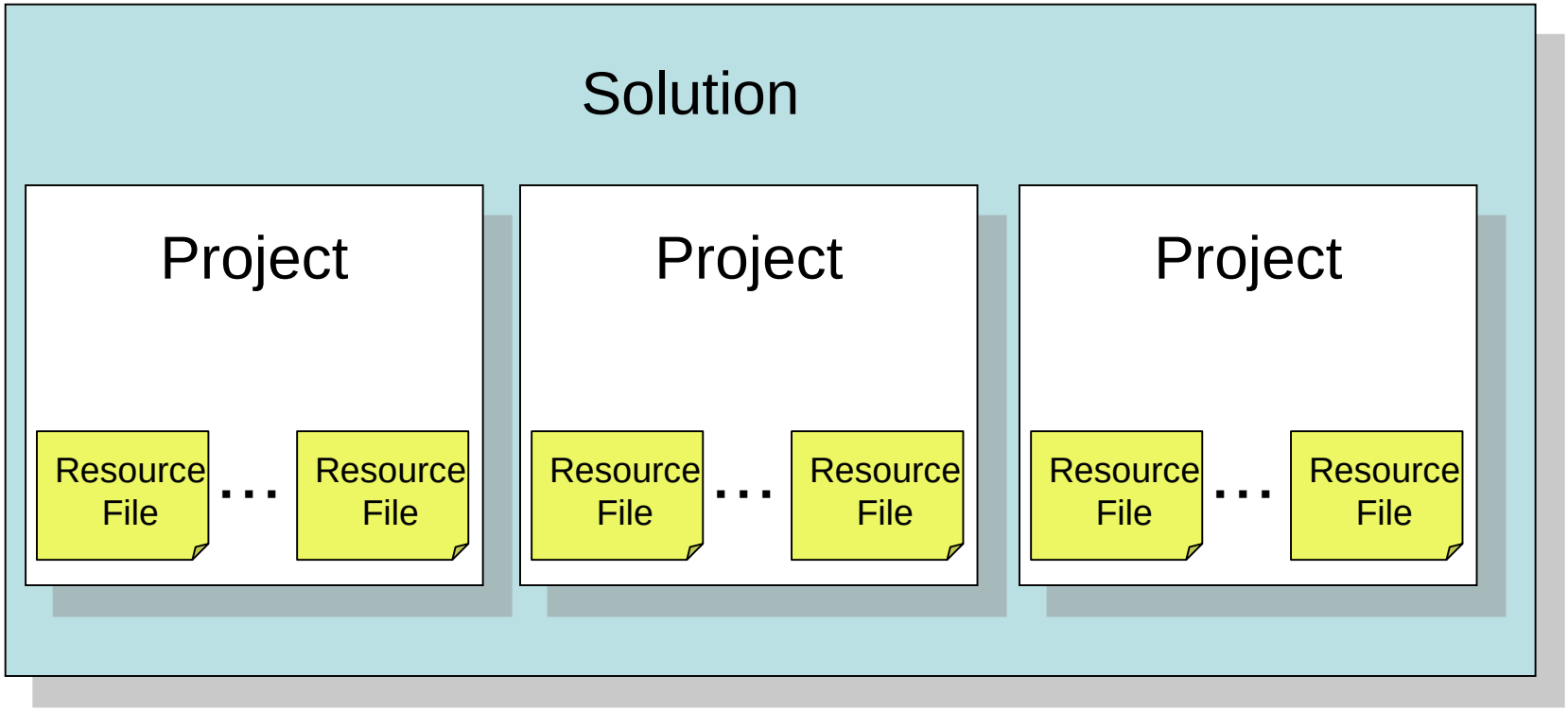
Screen Configuration of Visual Studio .NET (1/2)



Screen Configuration of Visual Studio .NET (2/2)

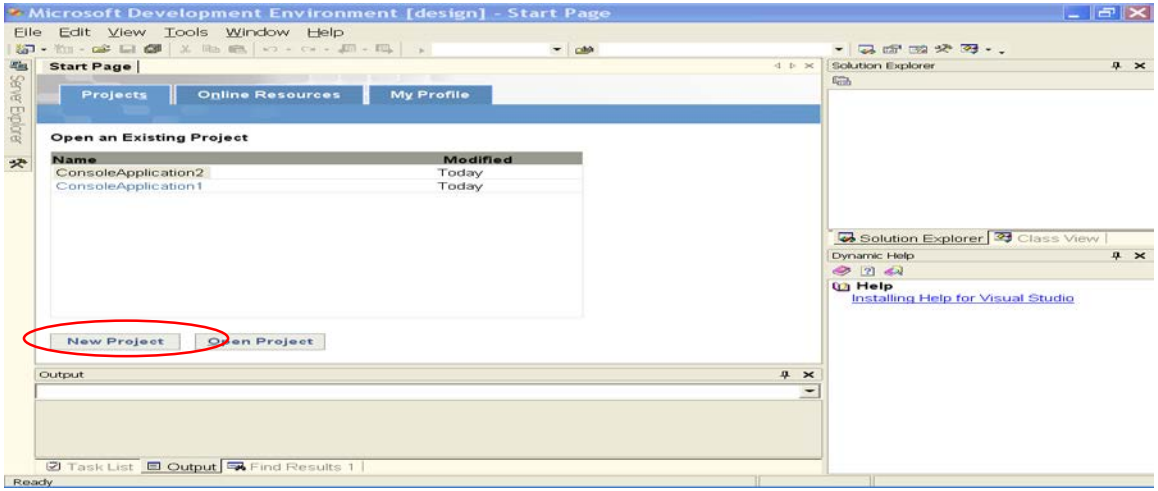


Solution and Project

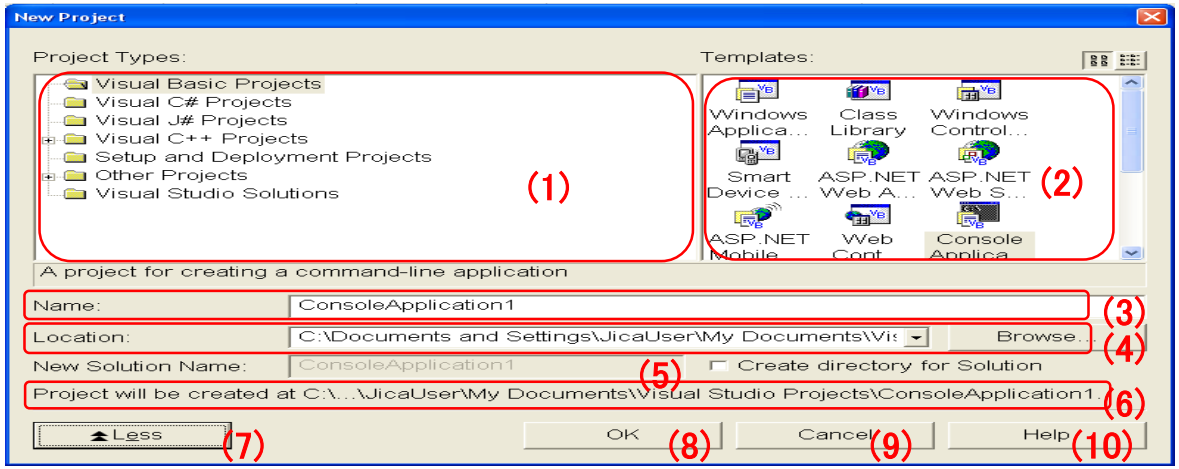


New Project (1/2)

To create a new program, it is necessary to create a new project



1) Click [New Project] on Start page



2) The [New Project] dialog box is displayed

To create a console application in Visual Basic .NET, select [Visual Basic Project] in project types and [Console Application] in template

New Project (2/2)

Visual Studio .NET is run by – [Start]Menu – [All Programs] – [Microsoft Visual Studio .NET 2003] – [Microsoft Visual Studio .NET 2003]. (When running a different version of Visual Studio.NET, select from menu, the relevant version name)

Select [File] – [New] – [Project...] in Menu, or create a new project in [Ctrl]+[Shift]+[N].

In the [New Project] dialog box, select the project type and template that meets the objectives of the program to be created.

(1) In [Project Types], projects are classified into categories.
Select the programming language to be used here.

(2) In [Templates], the project (template for project creation) item is displayed.
In compliance with this template, generate the files etc., required by the program.

(3) [Name] indicates the project name.

(4) [Location] indicates the creation location of the project folder.
As a default, a folder known as "Visual Studio Projects" is specified in My Documents.

(5) If the [Create directory for Solution] checkbox is checked, [New Solution Name] can be entered. When solution name is specified, check it.

(6) With the selection results of (3), (4) and (5) as base, the determined [Project Will be Created at] is displayed.

(7) If the [Less] button is clicked, [New Solution Name] and [Create directory for Solution] in (5) is hidden, and button display changes to [More]. If the [More] button is clicked, the hidden items are re-displayed, and button display once again returns to [Less].

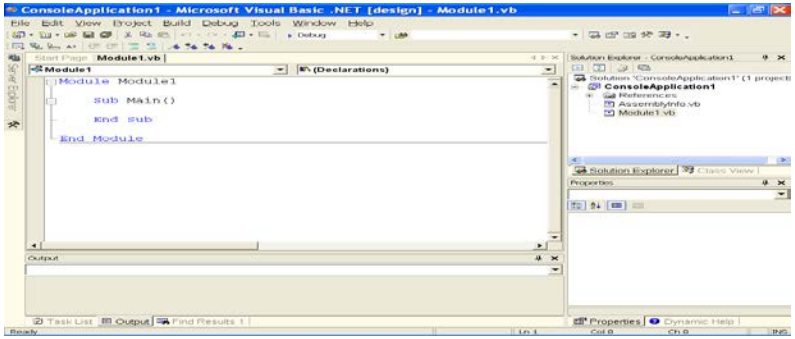
(8) When the [OK] button is clicked, a project is created.

(9) When the [Cancel] button is clicked, project creation is interrupted.

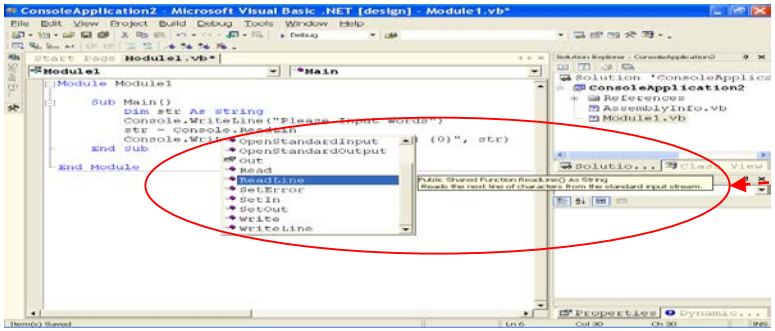
(10) When the [Help] button is clicked, [New Project] dialog related help is displayed.

Writing Code (1/2)

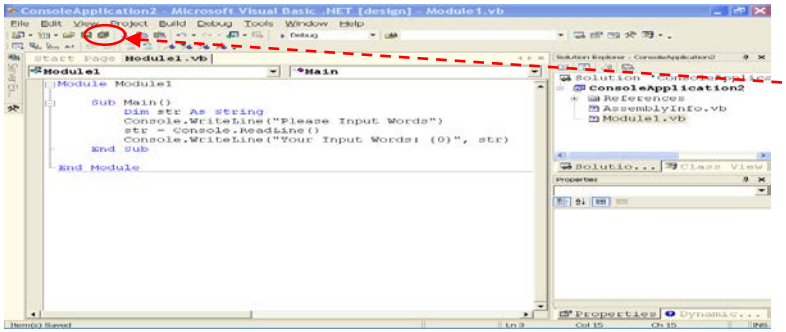
Write Code in Code Editor



1) In the code written in notepad in "Create and Run Console Application" in Chapter 1.2, the code from "Sub Main()" to "End Sub" is written from "Sub Main()" to "End Sub" of code editor.



2) While writing the code, a supplementary list is displayed in a format that complements the entered contents.
This function is known as IntelliSense



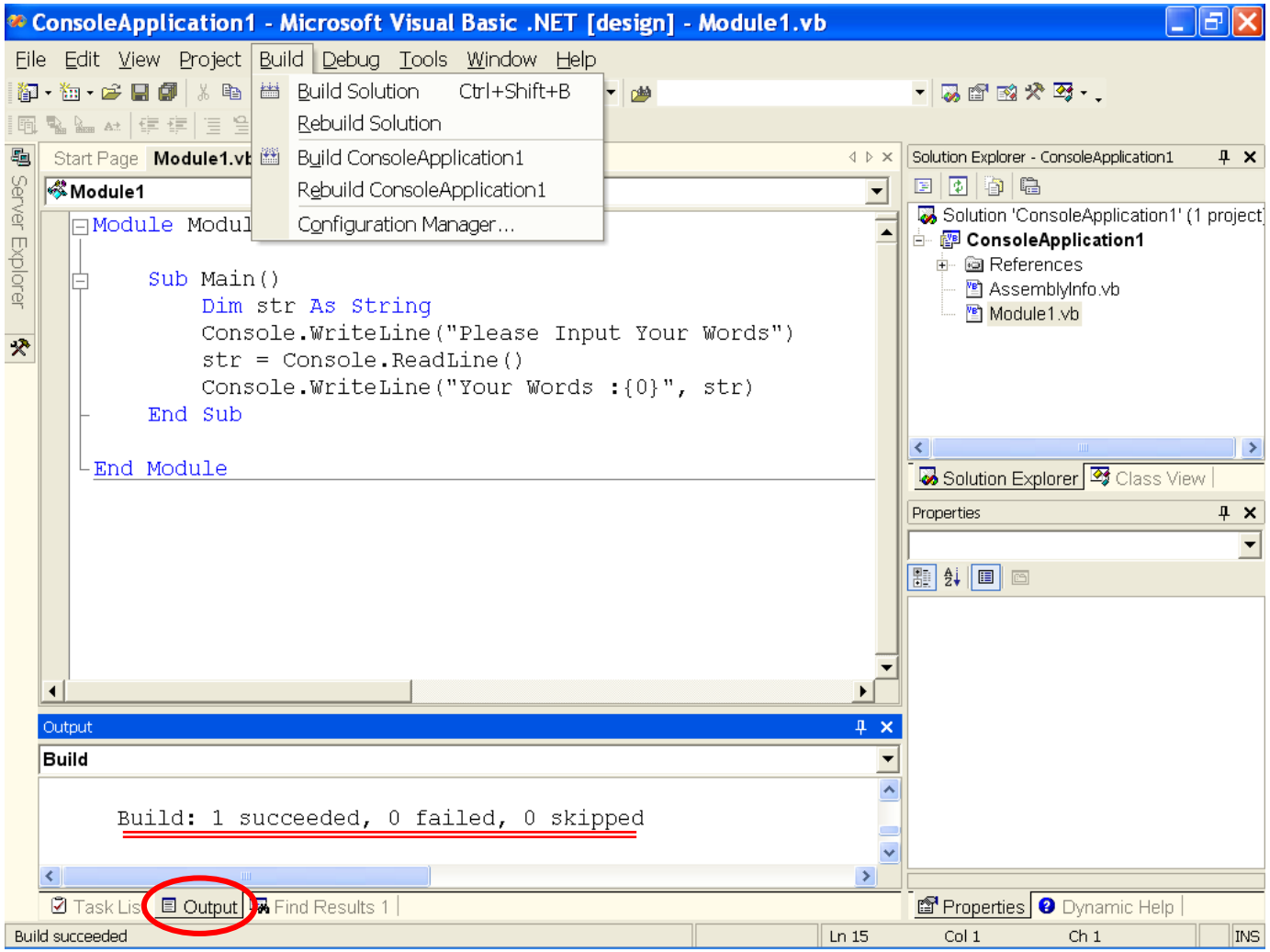
3) On completing code input, Save the project by clicking  .

Writing Code (2/2)

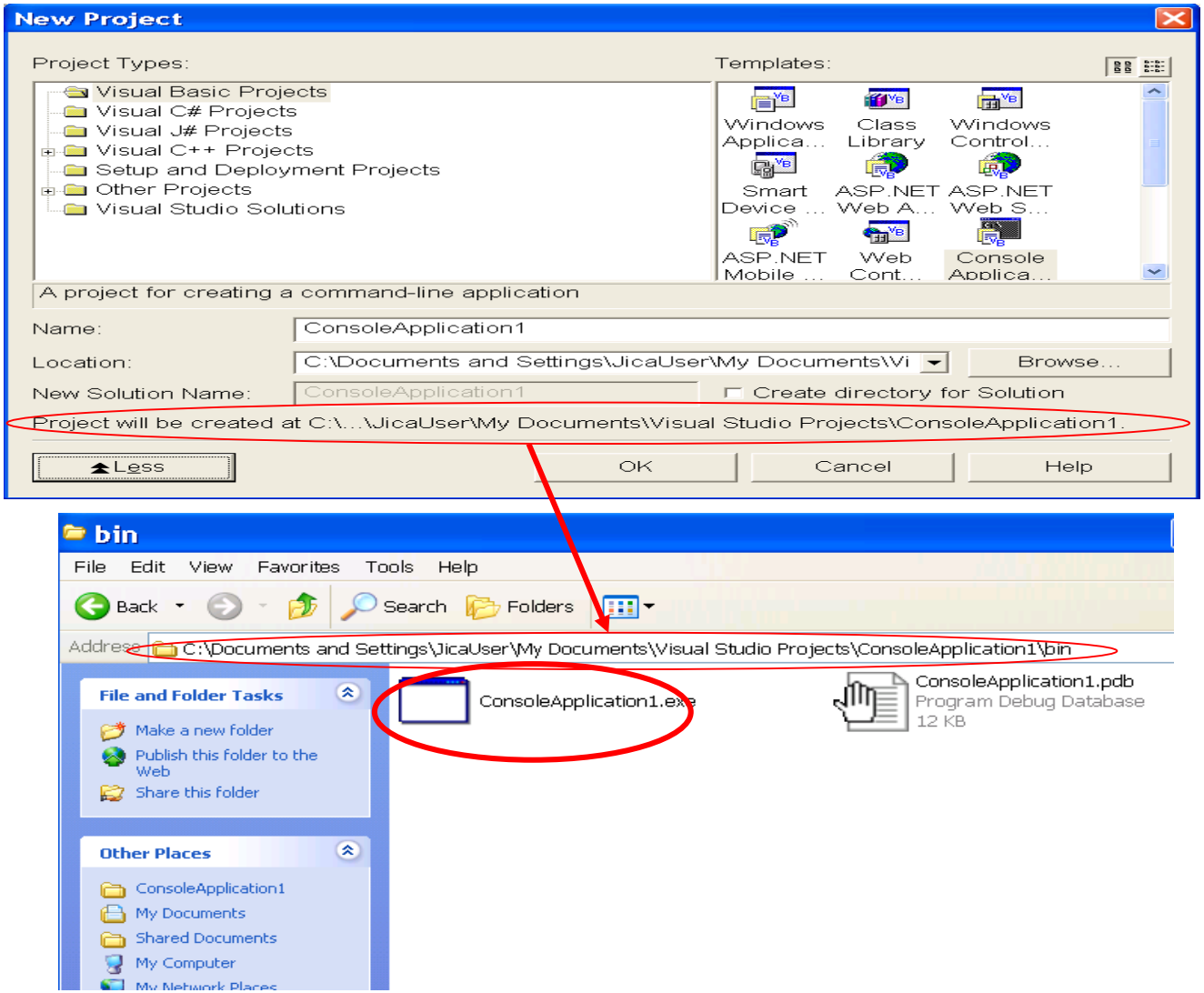
By using IntelliSense, not only can a code be written without referring to the manual, but it also becomes possible to prevent mistakes in input, and productivity can be increased. IntelliSense can be called by even using [Ctrl] + [space].

It is also possible to save the code that has been created by clicking [File] – [Save All] in menu.

Create an executable file

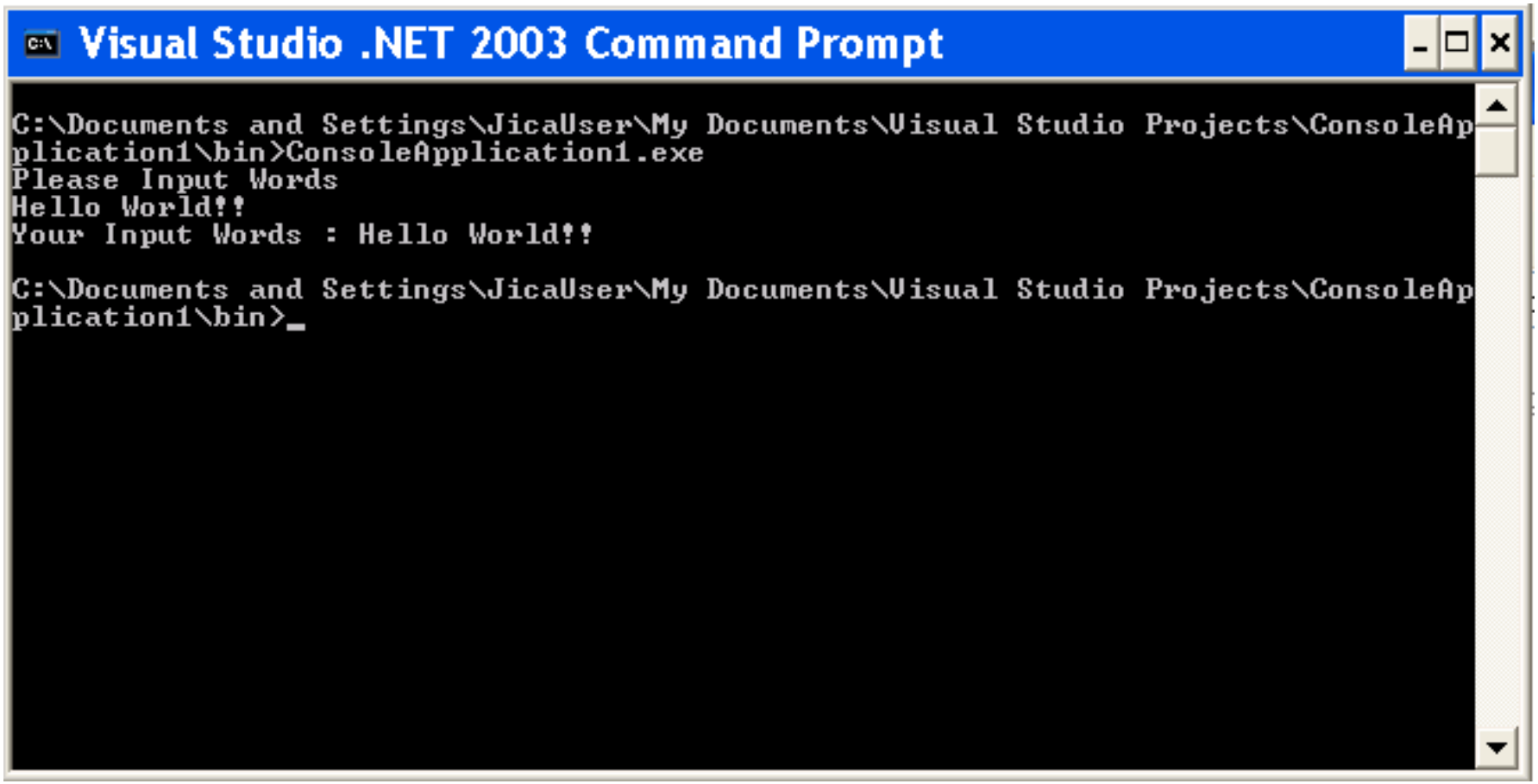


Running the created Program (1/2)

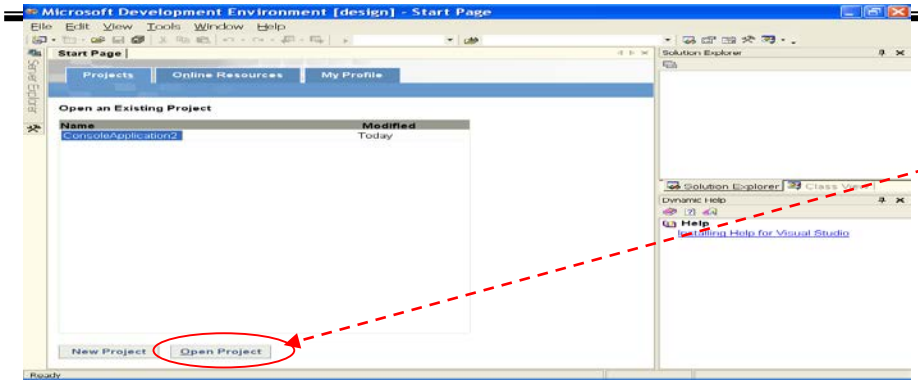


Running the created Program (2/2)

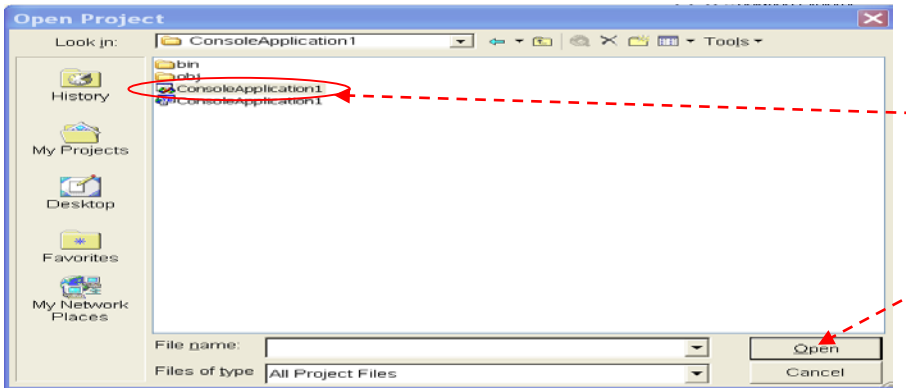
<OUTPUT>



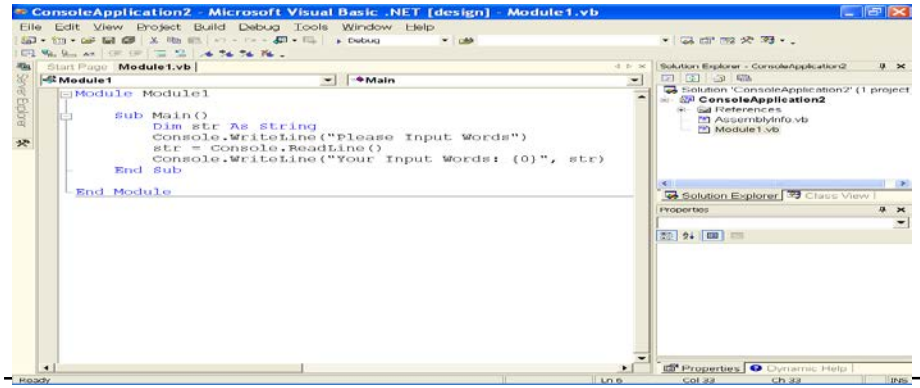
<Reference> Open a saved Project



1) Run Visual Studio .NET, and select [Open Project] in [Start Page]
(Can also open with [File] – [Open Solution...] in menu)



2) Select solution file (.sln) that includes the project to be opened and press [Open] ie file displayed with mark is solution file.



3) A saved project is opened

1.4 Creating a Console Application

In console application, the process known as Main is the starting point of the program

The program has to start from Main

```
Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Please Input Words")
        str = Console.ReadLine()
        Console.WriteLine("Your Input Words : {0}", str)
    End Sub

End Module
```

Standard Output (1/2)

- Display only one row of characters in console

```
Console.WriteLine("Character string to be displayed")
```

- Perform display inclusive of the values of variables

```
Console.WriteLine  ("Character string to be displayed{0}Character string to be displayed", Variable)
```

Standard Output (2/2)

<Example of Console.WriteLine>

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello World!!")

        Dim str1 As String
        Dim str2 As String
        str1 = "VB.NET"
        str2 = "VisualBasic.NET"
        Console.WriteLine("{0} is {1}", str1, str2)

    End Sub

End Module
```

<OUTPUT>

```
>ConsoleApplication1.exe
Hello World!!
VB.NET is VisualBasic.NET
>
```

Standard Input

To fetch a character string from console, use Console.ReadLine()

```
Module Module1

    Sub Main()
        Dim str As String

        Console.WriteLine("Please Input Words")

        str = Console.ReadLine()

        Console.WriteLine("Your Input Words : {0}", str)
    End Sub

End Module
```

Define the variables that receive the input

Fetch input from the console

Display input from the console

Command line arguments (1/2)

When using command line arguments,

```
Sub Main()

End Sub
```

is written as

```
Sub Main(ByVal CmdArgs() As String)

End Sub
```

Command line arguments (2/2)

<Example of command line arguments>

```
Module Module1

    Sub Main(ByVal CmdArgs() As String)
        Dim str As String
        For Each str In CmdArgs
            Console.WriteLine("Your Input Words : {0}", str)
        Next
    End Sub

End Module
```

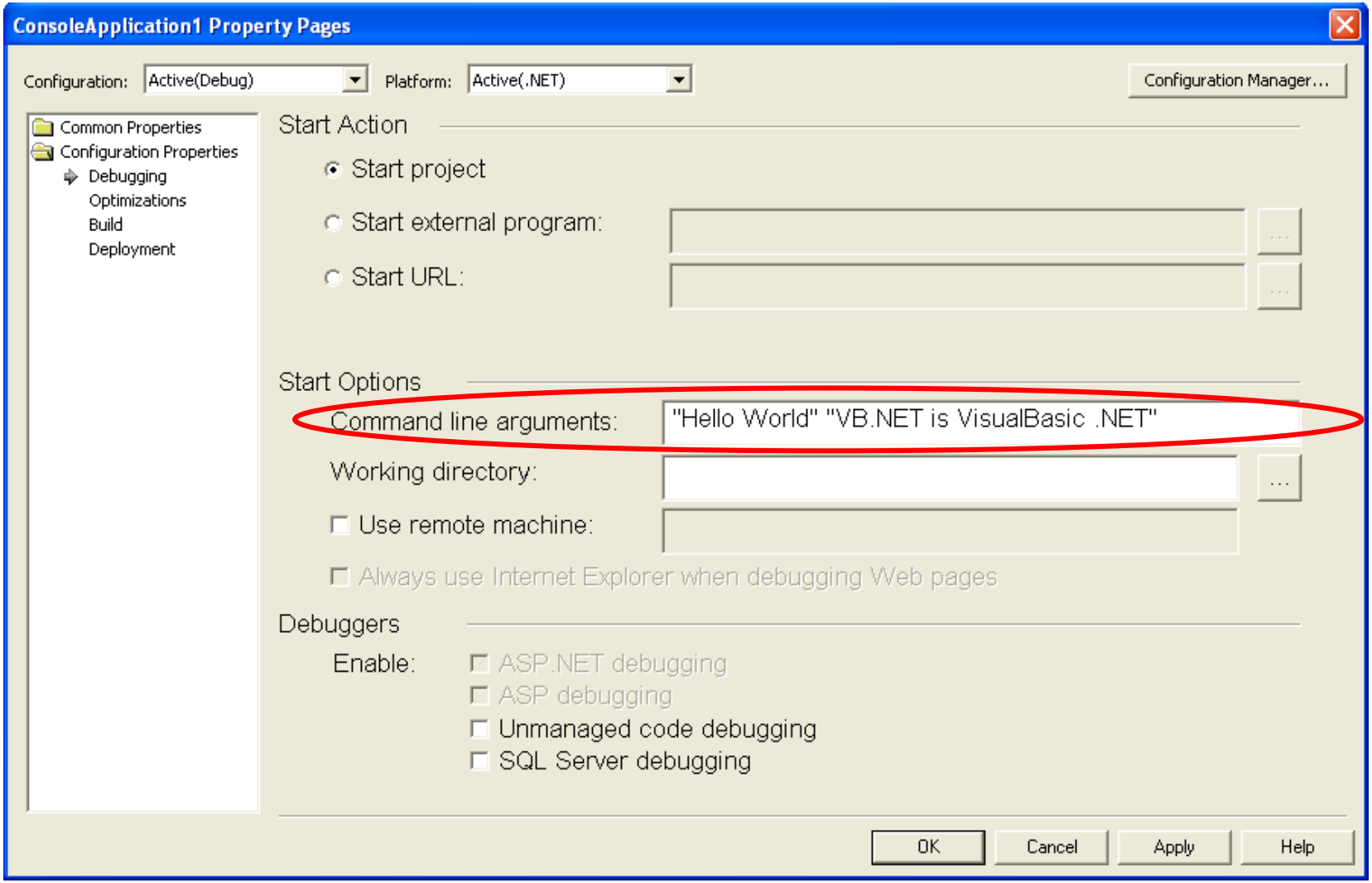
Receive command line arguments in Main

Output the received values

<OUTPUT>

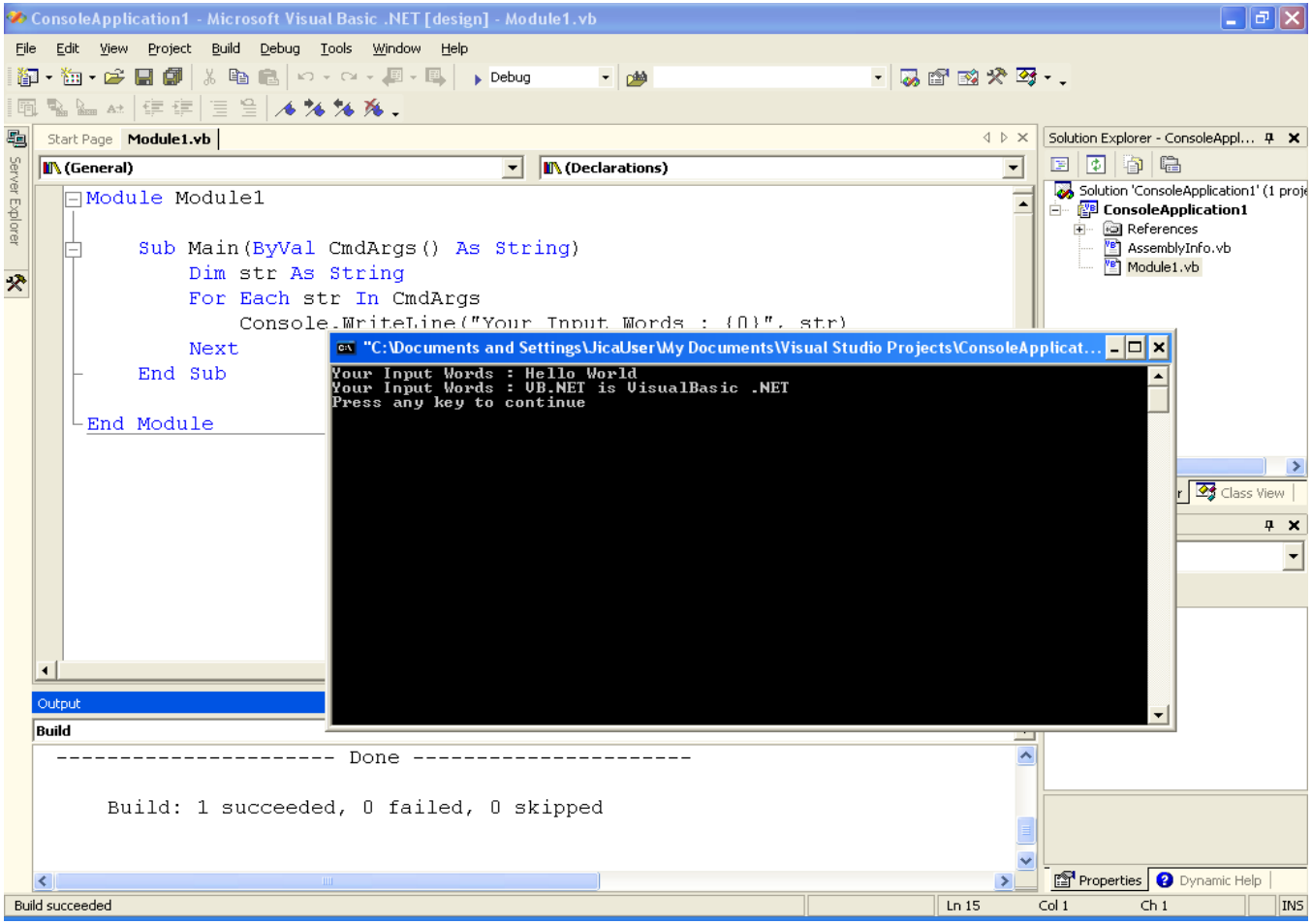
```
> ConsoleApplication1.exe “Hello World” “VB.NET is VIsualBasic.NET”
Your Input Words : Hello World
Your Input Words : VB.NET is VisualBasic.NET
>
```

Method for executing the program easily (1/2)



Method for executing the program easily (2/2)

<Specify Command line arguments in Project Properties, the result of implementation of [Ctrl]+[F5]>



Thanks You

Lecturer: CHHOM MAKARA
Phone : +(855) 69 2222 63, +(855) 92 92 50 61
E-mail : makarasumsung@gmail.com; facebook: “chhom makara”
Url : N/A

Work experiences: - IT Software Development Officer at ISI STEEL.
- Software Developer at SI Group.
- IT Manager at CUS Unversity.
- IT Lecturer at SETEC, CUS.

2. Usage of Variable and Array

2.1 Structure of Source Code

Source code of Visual Basic.NET has the following structure

- Module
- Method
- Statements
- Comment

```
Module Module1

    Sub Main()
        Dim a(10), i As Integer
        '  initialize array
        For i = 0 To 10
            a(i) = i
        Next
    End Sub

End Module
```

Declaration Statement
Comment
Executable Statement
Assignment Statement

Method

Module

Statements (1/2)

- Assignment Statement

  Use assignment operator to substitute the left hand side variable with a value.

```
a = 100
b = a - 20
```

- Declaration Statement

  Assign a name to method, variable, array and constant, and declare.

```
Dim  a  As Integer
Dim  b(10)  As  Double
```

- Executable Statement

  Statement that runs processes such as the conditional judgment structure (Chapter 3.2), loop structure (Chapter 3.3) etc.

```
If  a = 100  Then  ･･･
For  i  =  0  To 100 ･･･
```

Statements (2/2)

- Comment

```
REM   comment1
‘  comment2
Dim  a  As  Integer   ‘  comment3
```

- Long Statement

By using underscore (_), one can insert linefeed in the middle of a statement

```
Console.WriteLine(“{0}   {1}   {2}", _
                        first,  second, third)
```

Same meaning

```
Console.WriteLine(“{0}   {1}   {2}",  first,  second, third)
```

Keywords

AddHandler	AddressOf	AndAlso	Alias	And	Ansi
As	Assembly	Auto	Boolean	ByRef	Byte
ByVal	Call	Case	Catch	CBool	CByte
CChar	CDate	CDec	CDbl	Char	CInt
Class	CLng	CObj	Const	CShort	CSng
CStr	CType	Date	Decimal	Declare	Default
Delegate	Dim	DirectCast	Do	Double	Each
Else	ElseIf	End	Enum	Erase	Error
Event	Exit	False	Finally	For	Friend
Function	Get	GetType	GoSub	GoTo	Handles
If	Implements	Imports	In	Inherits	Integer
Inteace	Is	Let	Lib	Like	Long
Loop	Me	Mod	Module	MustInherit	MustOverride
MyBase	MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On	Option
Optional	Or	OrElse	Overloads	Overridable	Overrides
ParamArray	Preserve	Private	Property	Protected	Public
RaiseEvent	ReadOnly	ReDim	REM	RemoveHandler	Resume
Return	Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String	Structure
Sub	SyncLock	Then	Throw	To	True
Try	TypeOf	Unicode	Until	Variant	When
While	With	WithEvents	WriteOnly	Xor	

2.2 Variables and Constants

- Variable
  - Variable is the receptacle for saving data within memory.
  - Before using variable, it is necessary to declare name and type of data being handled.
  - In declaration of variable, use Dim keyword

```
Dim  VariableName  As  DataType
```

VariableName	Name of Variable
DataType	Data type of Variable

  - Variable can also be initialized at the time of declaration
  - When performing multiple declarations simultaneously, punctuate with commas (,).

```
Dim  a  As  Integer  =  100
Dim  b  As  String  = "Hello"
Dim  c, d  As  Integer
dim c as byte : dim e as int16
```

Initialize when declaring

When performing multiple declarations, punctuate with commas.

Rules of Variable name

There are rules for variable names

- The first character should be an alphabet or underscore(_)
- Characters other than alphabets, numbers, underscore(_) cannot be used
- Keywords cannot be used (If they are only included, there is no problem)
- It should be within 16383 characters
- There is no upper case/lower case distinction

Valid Variable names	Invalid Variable names
abc _abc abc100	1a Starting character is a number a.b. cannot be used $abc $ cannot be used

Constants

A Constant is a variable whose value does not change within the program

- Advantages of Constant

By avoiding hard coding of numeric values:

- Coding errors are reduced
- Source code becomes easier to read
- Changes in source code are easier

To declare a Constant, use Const keyword

Const *ConstantName* As *DataType* = *Value*

ConstantName	Name of Constant
DataType	Data type of Constant
Value	Value of Constant

2.3 Data Type

- Elementary Data Types
  - Numeric Data Types
    - + Integral Types
    - + Nonintegral Types
  - Character Data Types
  - Miscellaneous Data Types
    - + Boolean Type
    - + Date Type
    - + Object Type
- Composite Data Types
  - Structure
  - Array
  - Class
- Enumeration

Various Data Types (1/4)

- Integral Types
  - Signed Integral Types: Short (16 bit),Integer (32 bit), Long (64 bit)
  - Unsigned Integral Types:Byte (8 bit), UInt16,UInt32, Uint64, Ushort, ..
- Nonintegral Types
  - Decimal Data Type:Decimal (128 bit)
  - Single-precision floating-point number : Single (32 bit)
  - Double-precision floating-point number : Double (64 bit)
- Character Data Type

  Char refers to 1 character, String refers to a character string of any length
  - Character Type : Char
  - String Type : String
- Boolean Data Type

  Boolean

  Expresses the logical value with True or False

Various Data Types (2/4)

- Date Data Type
  - Date (64 bit)
- Object Data Type
  - Object

```
Dim  a  As  Integer
a  =  10

Dim  pi  As  Double  =  3.14159

Dim  name  As  String =  “OIC”

Dim  z  As  Boolean
z  =  False

Dim  var  As  Object
var  =  name
var  =  100
```

Various Data Types (3/4)

Data Type	Data Type	Byte	Value Range
Boolean	Boolean Type	2	True or False
Byte	Integral Types	1	0 through 255 (unsigned)
Char	Character Data Type	2	0 through 65535 (unsigned)
Short	Integral Types	2	-32,768 through 32,767.
Integer	Integral Types	4	-2,147,483,648 through 2,147,483,647.
Long	Integral Types	8	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807.
Decimal	Nonintegral Types	16	0 through +/- 79,228,162,514,264,337,593,543,950,335 with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28).

Various Data Types (4/4)

Data Type	Data Type	Byte	Value range
Date	Date Type	8	0:00:00 on January 1, 0001 through 11:59:59 PM on December 31, 9999.
Double	Nonintegral Types	8	-1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values
Single	Nonintegral Types	4	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values.
String	Character Data Type	Depends on platform	0 to approximately 2 billion Unicode characters
Object	Object Data Type	4	Any type can be stored in a variable of type Object

Reference: http://msdn.microsoft.com/en-us/library/47zceaw7(vs.71).aspx

Structure

Structure is a user defined data type

- Has member variables and methods as the configuration elements
- Define using Structure keyword
- Structure is a Value Type (As regards Value Types, refer to “Value Type and Reference Type”)

```
Structure  StructName
     StructMemberDecl
End  Structure
```

StructName	Name of Structure
StructMemberDecl	Definition of member variable of Structure

Example of Structure

Example of Structure

```
Structure ss
    Public a As Integer
    Public b As Integer
    Public Function sum() As Integer
        Return a + b
    End Function
End Structure
```

Usage of Structure

```
Dim  ms  As  ss
ms.a  =  0
ms.b  =  1
Console.WriteLine(ms.sum)
```

Declare Structure

Access Member Variable of Structure

Access Method of Structure

Enumerations

Enumeration is relational data type having Integral Type values within a series of names

```
Enum  EnumName  As  DataType
   EnumMemberDelcs
End  Enum
```

EnumName	Name of Enumeration
DataType	Data types used in Enumeration
EnumMemberDecls	Declaration of enumerator

Example of Enumeration

```
Enum  DayOfWeek As  Integer
   Sunday
   Monday
   Tuesday
   Wednesday
   Thursday
   Friday
   Saturday
End  Enum
```

Output of Enumerations

- Refer to the Enumeration member

  Assign a period mark (.) to Enumeration name, and refer

```
Dim  today  As  DayOfWeek
today  =  DayOfWeek.Sunday
```

Declare the Enumeration Variable, and set the value

- When Enumeration is output in Console.WriteLine, the name is outputted as itself

```
Dim  today As DayOfWeek = DayOfWeek.Sunday
Console.WriteLine("Today is {0}", today)
```

<OUTPUT>

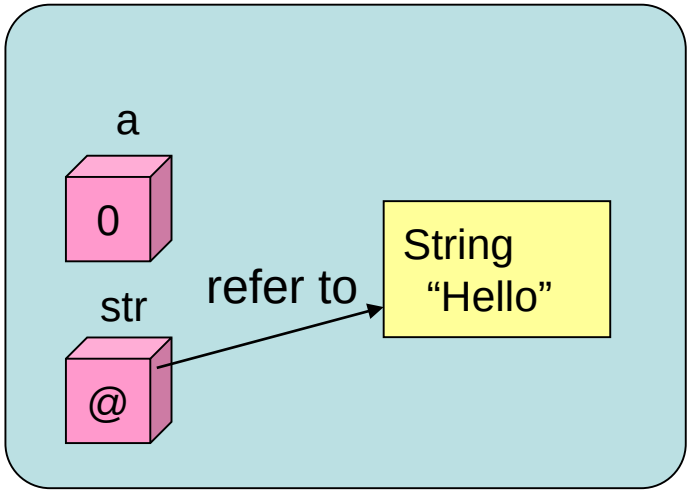
```
Today is Sunday
```

Value Types and Reference Types

The two fundamental categories of data type are Value Type and Reference Type.

- Value Type
  - Numeric Data Type (Integral Types , Nonintegral Types)
  - Boolean Type,Char Type,Date Type
  - Structure and Enumeration
  - String Type
- Reference Type
  - Array
  - Class Type

```
Dim  a  As  Integer
a  =  0

Dim  str  As  String
str  =  "Hello"
```



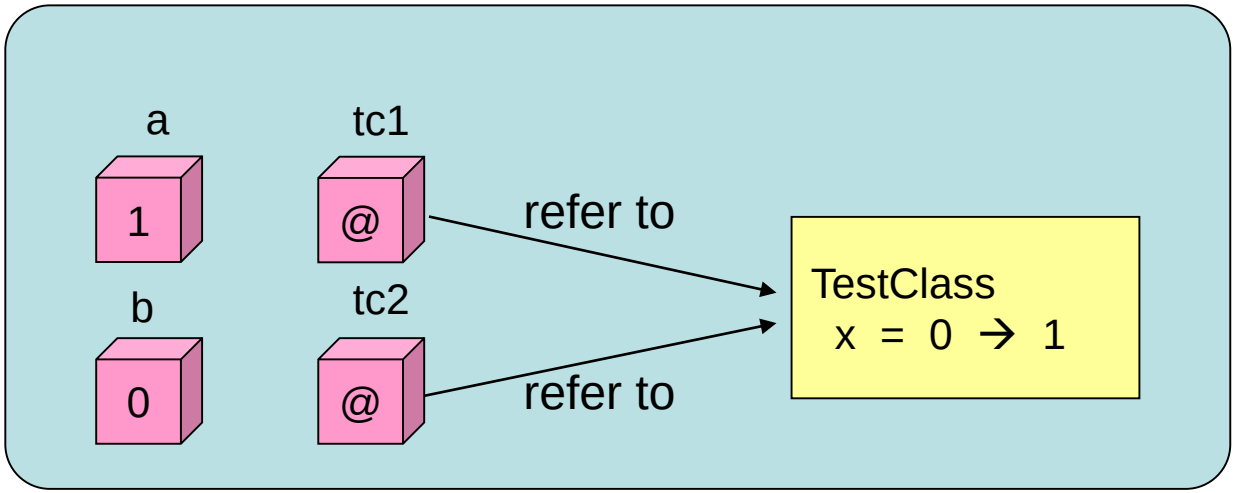
Difference between Value Type and Reference Type(1/2)

```
Module Module1
  Public Class TestClass
    Public x As Integer
  End Class

  Sub Main()
    Dim a, b As Integer
    Dim tc1, tc2 As TestClass

    a = 0
    b = a
    a = 1
    Console.WriteLine("a = {0}  b = {1}", a, b)

    tc1 = New TestClass
    tc1.x = 0
    tc2 = tc1
    tc1.x = 1
    Console.WriteLine("tc1.x = {0}  tc2.x = {1}", tc1.x, tc2.x)
  End Sub
End Module
```

Difference between Value Type and Reference Type (2/2)

<OUTPUT>

```
a = 1  b = 0
tc1.x = 1  tc2.x = 1
```



tc1 and tc2 are Reference Type Variables, that refer to the same memory area

2.4 Array

- In array, multiple data is collected using the same name and index and then handled
- The index of array starts from 0
- There are one dimensional Arrays and multidimensional arrays

```
Dim  ArrayName([MaxIndex])  As  DataType
```

ArrayName	Name of variable
MaxIndex	Maximum value of index (Therefore array size is MaxIndex+1)
DataType	Data type of array

```
Dim  a(2)  As Integer
a(0)  =  1
a(1)  =  10
a(2)  =  100
Dim  b(1, 2)  As  Integer
b(0, 0)  =  1
b(1, 2)  =  5
```

Usage of Array

- Initializing an array

  When initializing at the time of array declaration, arrange the initialization values within {}

```
Dim  a()  As  Integer  =  {0, 1, 2}
```

- Substituting an array

  An Array can be substituted for another array

```
Dim  a()  As  Integer  =  {0, 1, 2}
Dim  b()  As  Integer
b  =  a
```

- The size of an array can be fetched as array name .Length

```
Dim  a(100)  As Integer
‘  Array size 101 is outputted
Console.WriteLine(a.Length)
```

Redimensioning Arrays (1/2)

- **ReDim** : is used to redimension an array in Visual Basic .NET by clear the all old element of the array and create new dimension array with the same name.

```
Dim     x( )   As   String
ReDim    x(5)
```

```
Dim  y(2)   As   String
ReDim    y(5)
```

Redimensioning Arrays (2/2)

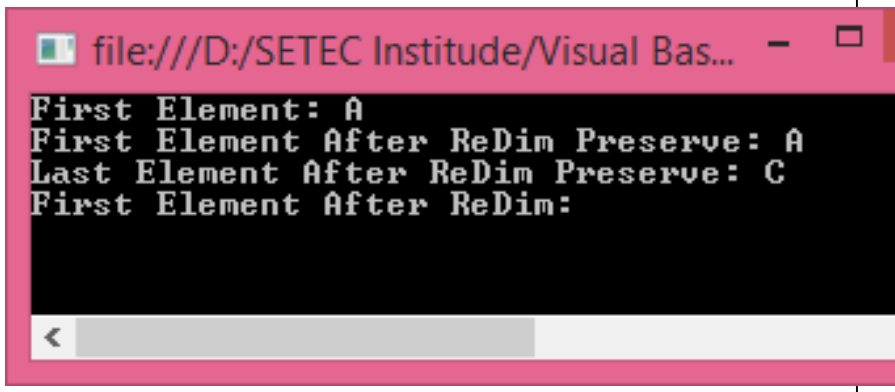
- **ReDim Preserve** : is used to reduce or extend the number of element of the array. The existing array still keep the same value.

```
Dim     x( )   As   String
ReDim  Preserve   x(5)
```

```
Dim  y(2)   As   String
ReDim  Preserve   y(5)
```

Example of Redimensioning Arrays

```
Module Module2
    Sub Main()
        Dim x() As String
        ReDim x(1)
        x(0) = "A"
        x(1) = "B"
        Console.WriteLine("First Element: " & x(0))
        ReDim Preserve x(2)
        x(2) = "C"
        Console.WriteLine("First Element After ReDim Preserve: " & x(0))
        Console.WriteLine("Last Element After ReDim Preserve: " & x(2))
        ReDim x(2)
        Console.WriteLine("First Element After ReDim: " & x(0))
        Console.Read()
    End Sub
End Module
```



3. Control Structure

3.1 Operator

Operator refers to the symbols or keywords that indicate the operation contents

- In Visual Studio .NET, various operators are supported
  - Unary Operator
  - Arithmetic Operator
  - Concatenation Operator
  - Relational Operator
  - Logical Operator

Unary Operator

Unary Operator refers to the operator that is attached before the value

Operator	Explanation
+ (Unary Plus Operator)	Does not change the sign of the value
- (Unary Minus Operator)	Reverses the sign of the value
Not (Unary Negation Operator)	Logical negation of the value and compliment for every bit

Arithmetic Operator

Arithmetic Operator refers to the operator that performs arithmetical operations

Operator	Explanation
+ (Addition Operator)	Calculates the sum of 2 numeric values
- (Subtraction Operator)	Calculates the difference between 2 numeric values
* (Multiplication Operator)	Calculates the product of 2 numeric values
/ (Division Operator)	Performs division of 2 numeric values, and returns the result as a floating decimal point number
\ (Integer Division Operator)	Performs division of 2 numeric values, and returns the result of division as an integer value
Mod (Mod Operator)	Calculates the remainder of division of 2 numeric values
^ (Exponentiation Operator)	Calculates the value of the numeric value of the 1st term raised to the power of the numeric value of the 2nd term

Concatenation Operator

Concatenation Operator refers to the operator that performs the linking of character strings

Operator	Explanation
& (Concatenation Operator)	Returns the character string that links 2 values from left to right When the value is not a character string, convert to character string type (String)

Relational Operator

Relational Operator refers to the operator that compares 2 values

Operator	Explanation
= (Equivalent Operator)	Compares the 2 values, and if they are of the same value, returns True, if other than that, returns False
<> (Non equivalent Operator)	Compares the 2 values, and if the values differ, returns True, if other than that, returns False
< (< Operator)	Compares the 2 values, and if the first value is smaller than the second value, returns True, if other than that, returns False
> (> Operator)	Compares the 2 values, and if the first value is larger than the second one, returns True, if other than that returns False
<= (<= Operator)	Compares the 2 values, and if the first and the second value are the same, or if the first value is smaller, returns True, if other than that, returns False
>= (>= Operator)	Compares the 2 values, and if the first and second values are the same or if the first value is larger, returns True, if other than that, returns False
Like (Like Operator)	Compares the 2 character strings, and when the first character string and second character string match or their patterns match, returns True, if other than that returns False
Is (Is Operator)	Compares the 2 Object variables, and if both refer to the same Object, returns True if other than that, returns False
TypeOf...Is (TypeOf...Is Operator)	If the first Object and second type (Object Type) are the same, returns True if other than that returns False

Logical Operator

Logical Operator refers to the operator that performs Boolean operations

Operator	Explanation
And	When requesting for the logical product of a boolean expression that requests for the logical product of 2 boolean expressions or the product of each bit of 2 numeric expressions, the result is True, only when both are True
Or	When requesting for a logical OR of a boolean expression that requests for a logical OR of 2 boolean expressions or an OR for every bit of 2 numeric expressions, when one or both are True, the result is True
Xor	When requesting for an exclusive logical OR of a boolean expression that requests for the exclusive logical OR for 2 boolean expressions or for every bit of the 2 numeric expressions, only when one is True, the result is True

Priority order of the Operator

Priority order	Category	Operator
High	Exponential operations	^
	Unary minus sign	+,-
	Multiplication operations	*,/
	Integer division	\
	Division	Mod
	Addition operations	+,-
	Connection	&
	Shift	<<,>>
	Relational calculation	=,<>,<,>,<=,>=,Like,Is, TypeOf...Is
	Logical NOT	Not
	Logical AND	And
	Logical OR	Or
Low	Logical XOR	Xor

Assignment Operator

Assignment Operator is the operator that assigns value to the variable or property, etc.

Operator	Explanation
=	Assigns the the value on the right to the variable or property specified on the left
&=	Links the right hand side character string expression (String), with the variable of the left hand side character string type (String), and substitutes the result in the left hand side variable
^=	Requests for the right hand side value for the left hand side variable, and substitutes the result in the left hand side variable
*=	Multiplies the right hand side Variable with the left hand side value, and substitutes the result in the left hand side variable
+=	Adds the left hand side variable to the right hand side value, and substitutes the result in the left hand side variable
-=	Subtracts the right hand side variable from the left hand side value, and substitutes the result in the left hand side variable
/=	Divides the left hand side variable with the right hand side value, and in the double point precision floating point number (Double) of the quotient, substitutes with the left hand side variable

Implicit Conversions

```
Dim A As Integer = 10
Dim D As Byte

D = A
```

Integer Type ”10” is converted to a Byte Type”10”

Explicit Conversion

When Ctype is used, type conversion can be implemented explicitly

Ctype (Conversion source expression, conversion object type)

(Example)

```
Dim str1 As String
Dim i1 As Integer

str1 = “2”
i1 = CType(str1,Integer)
```

Convert to Integer type 2 ,
the character string “2” that is
saved in String type variable str1,
and substitute with Integer type i1

Parse

Parse : is used convert data to any data type

DataType.Parse (Source)

(Example)

```
Dim str1 As String
Dim i1 As Integer

str1 = “2”
i1 = Integer.Parse(str1)
```

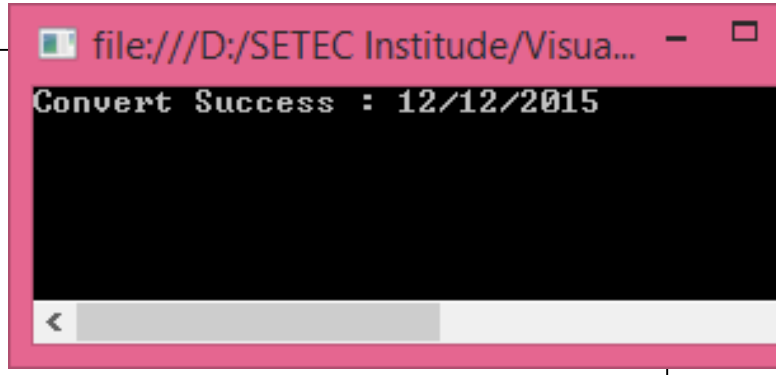
Convert to Integer type 2 ,
the character string “2” that is
saved in String type variable str1,
and substitute with Integer type i1

TryParse

TryParse : is used test and convert data to any data type

DataType.Parse (Source, returnVariable)

```
Sub Main()
    Dim str1 As String
    Dim D As Date

    str1 = "2015-12-12"
    Dim B As Boolean = Date.TryParse(str1, D)
    If B = False Then
        Console.WriteLine("This string cannot convert to date time!")
    Else
        Console.WriteLine("Convert Success : " & D)
    End If
    Console.Read()
End Sub
```



Conversion in sort form

Other Ways to convert data type

```
Dim   STR  As String  = “12”

Dim   I    As  Integer  =  Cint(STR)

Dim   I    As  Decimal  =  Cdec(STR)

……..
……
```

3.2 Conditional Structure

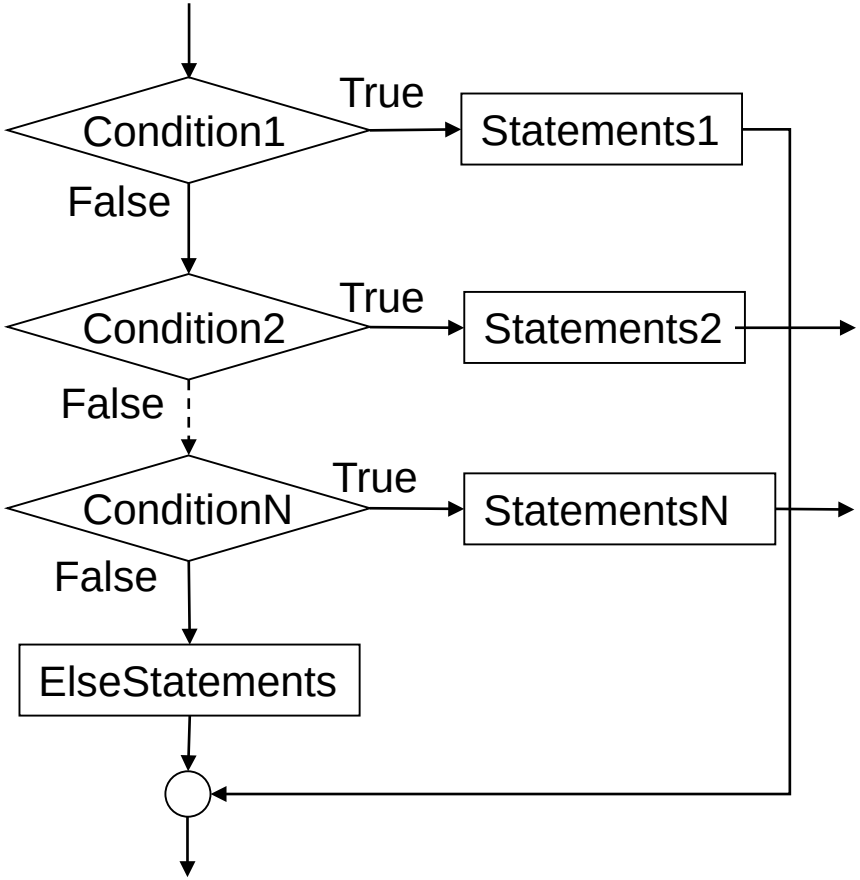
- Search for the condition using the conditional judgment structure and,
  process in compliance with the result

- In Visual Basic .NET, the following conditional judgment structures are supported
  - If ... Then ... Else
  - Select ... Case

If ... Then ... Else

```
If  Condition1  Then
    Statements1
[ElseIf  Condition2  Then
Statements2          ]
        .
        .
        .
[Else
        ElseStatements ]
End If
```

<Flow of Processing>



Usage Example of If ... Then ... Else

<Example of Program>

```
Module Module1

    Sub Main()
        Dim str1 As String
        Dim str2 As String
        Dim i1 As Integer
        Dim i2 As Integer

        Console.WriteLine("Input Number")
        str1 = Console.ReadLine()
        i1 = CType(str1,Integer)
        Console.WriteLine("Input Number")
        str2 = Console.ReadLine()
        i2 = CType(str2,Integer)

        If i1 > i2 Then
            Console.WriteLine("{0} is greater than {1}", i1, i2)
        ElseIf i1 < i2 Then
            Console.WriteLine("{1} is greater than {0}", i1, i2)
        Else
            Console.WriteLine("{0} equal {1}", i1, i2)
        End If

    End Sub

End Module
```

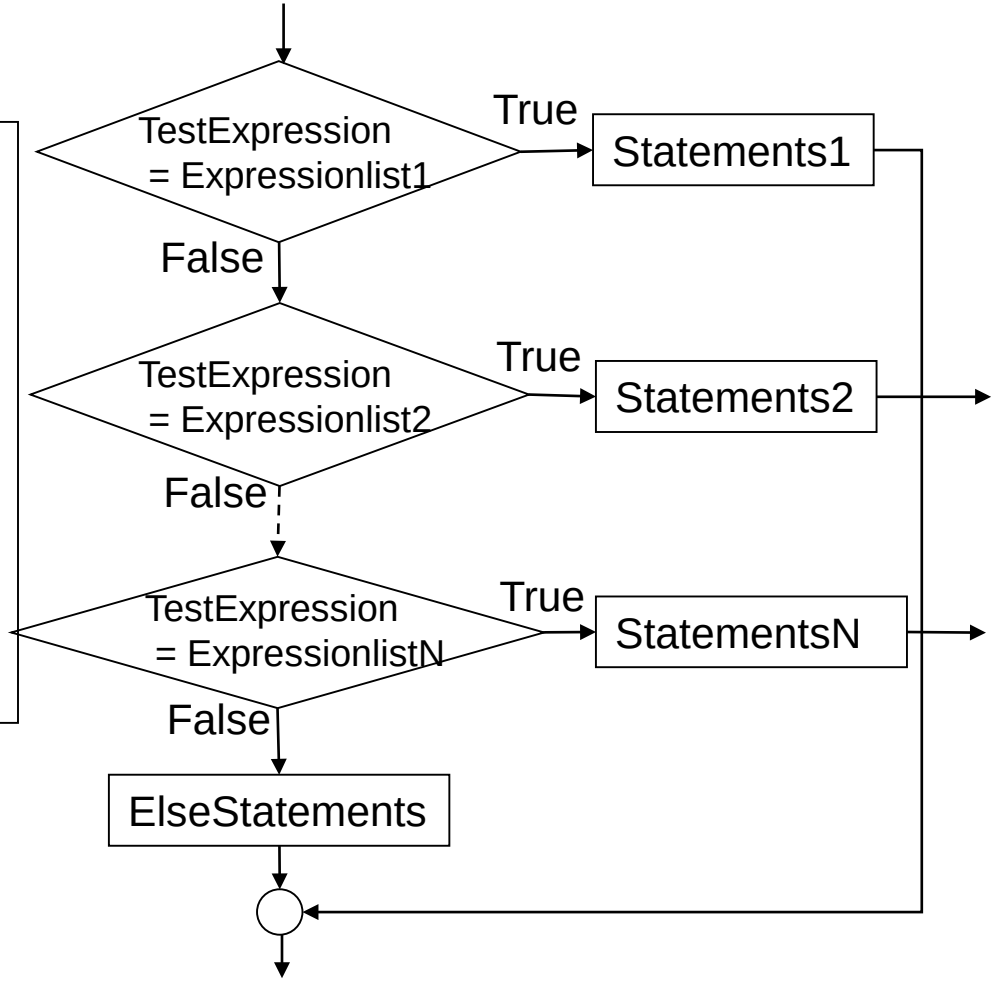
<OUTPUT>

```
>ifElse.exe
Input Number
100
Input Number
10
100 is greater than 10

>ifElse.exe
Input Number
10
Input Number
100
100 is greater than 10

>ifElse.exe
Input Number
100
Input Number
100
100 equal 100
```

Select ... Case

```
Select Case  TestExpression
Case Expressionlist1
Statements1
        .
        .
        .
 [Case Else
   ElseStatements   ]
End Select
```

<Flow of Processing>



Usage example of Select ... Case

<Example of Program>

```
Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Input Animal")
        str = Console.ReadLine()

        Select Case str
            Case "dog", "DOG"
                Console.WriteLine("BOW!!")
            Case "cat", "CAT"
                Console.WriteLine("MEW")
            Case Else
                Console.WriteLine("I don't know")
        End Select

    End Sub

End Module
```

<OUTPUT>

```
>case.exe
Input Animal
dog
BOW!!

>case.exe
Input Animal
CAT
MEW

>case.exe
Input Animal
DOG
BOW!!

>case.exe
Input Animal
rat
I don’t know
```

3.3 Loop Structure

- By using Loop Structure, it becomes possible to run 1 row or multiple rows of code (statement) repeatedly

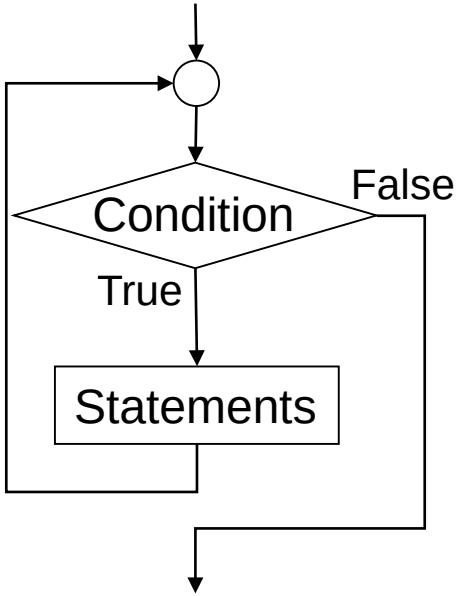
- Statement can be executed repeatedly till a particular condition is fulfilled

- In Visual Basic .NET, the following Loop Structure is supported
  - While
  - Do ... Loop
  - For ... Next
  - For Each ... Next

While

```
While Condition
        Statements
End While
```

<Flow of Processing>



Usage example of While

<Example of Program>

```
Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Where is the capital in Japan?")

        While str <> "Tokyo"
            Console.WriteLine("Input Please.")
            str = Console.ReadLine()
        End While

        Console.WriteLine("GOOD!!")

    End Sub

End Module
```

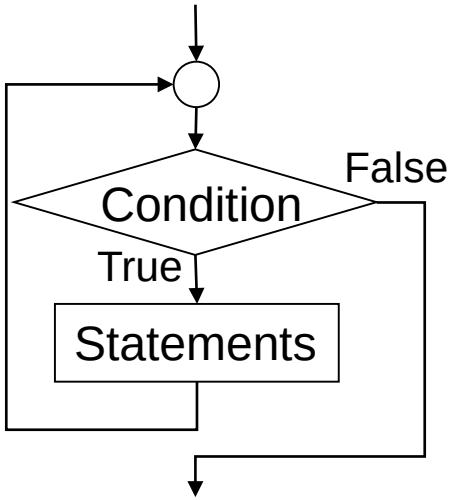
<OUTPUT>

```
>While.exe
Where is the capital in Japan?
Input Please
Osaka
Input Please
Kobe
Input Please
Yokohama
Input Please
Tokyo
GOOD!!

>
```

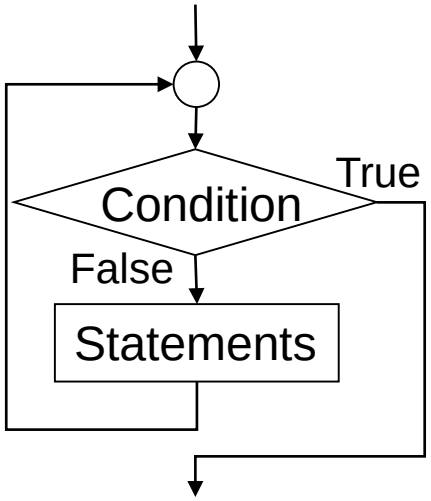
Do ... Loop(Pre-test Loop)

<Flow of Processing>

Do While *Condition*
Statements
Loop



Do Until *Condition*
Statements
Loop



Usage example of Do...Loop (Pre-test Loop)

<Example of Program>

```
Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Where is the capital in Japan?Input Please.")
        str = Console.ReadLine()

        Do While str <> "Tokyo"
            Console.WriteLine("Input Please.")
            str = Console.ReadLine()
        Loop

        Console.WriteLine("GOOD!!")

    End Sub

End Module
```

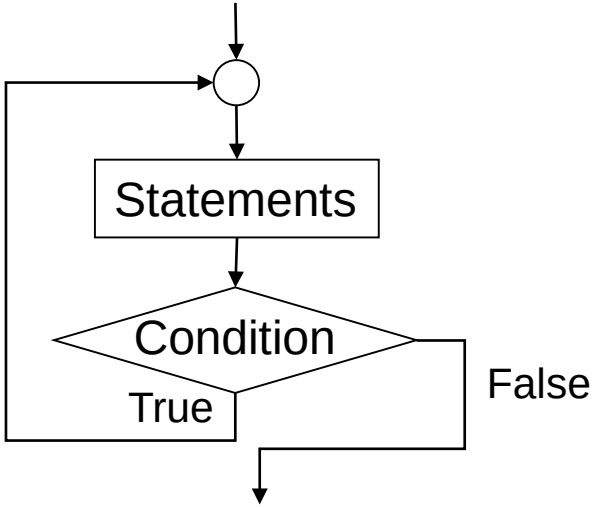
<OUTPUT>

```
>PreDoLoop.exe
Where is the capital in Japan?Input
Please.
Osaka
Input Please
Kobe
Input Please
Yokohama
Input Please
Tokyo
GOOD!!

>
```

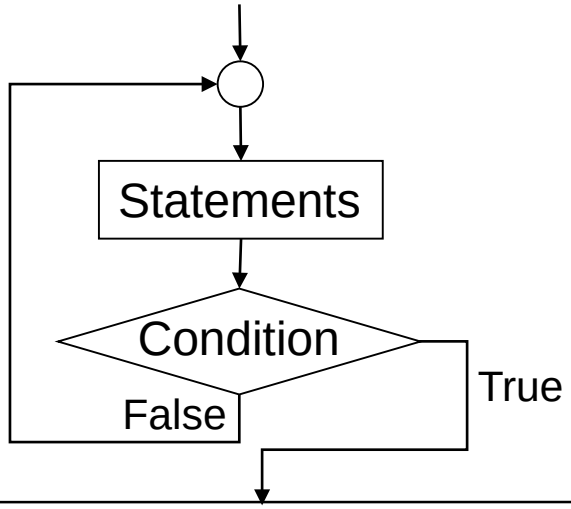
Do ... Loop(Post-test Loop)

<Flow of Processing>

```
Do
    Statements
Loop While Condition
```




```
Do
    Statements
Loop Until Condition
```



Usage example of Do...Loop(Post-test Loop)

<Example of Program>

```
Module Module1

    Sub Main()
        Dim str As String
        Console.WriteLine("Where is the capital in Japan?")

        Do
            Console.WriteLine("Input Please.")
            str = Console.ReadLine()
        Loop While str <> "Tokyo"

        Console.WriteLine("GOOD!!")
    End Sub

End Module
```

<OUTPUT>

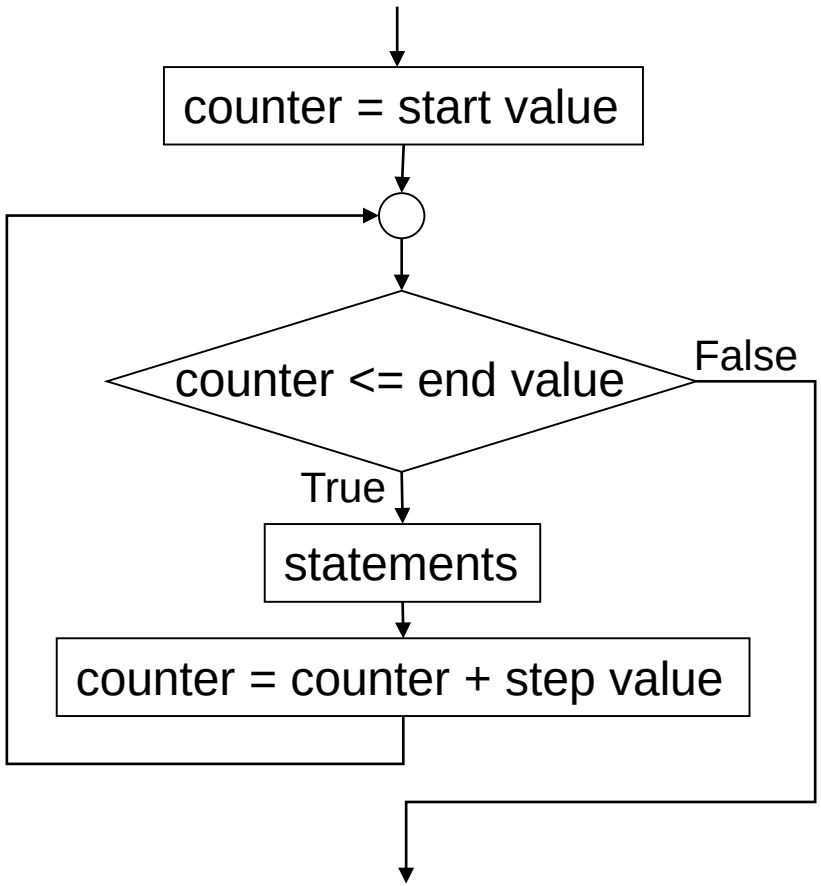
```
>PostDoLoop.exe
Where is the capital in Japan?
Input Please
Osaka
Input Please
Kobe
Input Please
Yokohama
Input Please
Tokyo
GOOD!!

>
```

For ... Next

For *counter* = *start* To *end* [Step *step*]
statements
Next

counter:Counter (Variable)
start:Initial value
end:Last value
step:Added number

<Flow of Processing>



Usage example of For ... Next

<Example of Program>

```
Module Module1

    Sub Main()
        Dim i As Integer
        Dim sum As Integer
        sum = 0

        For i = 1 To 10
            sum = sum + i
        Next

        Console.WriteLine("SUM(1:10) = {0}", sum)

    End Sub

End Module
```

<OUTPUT>

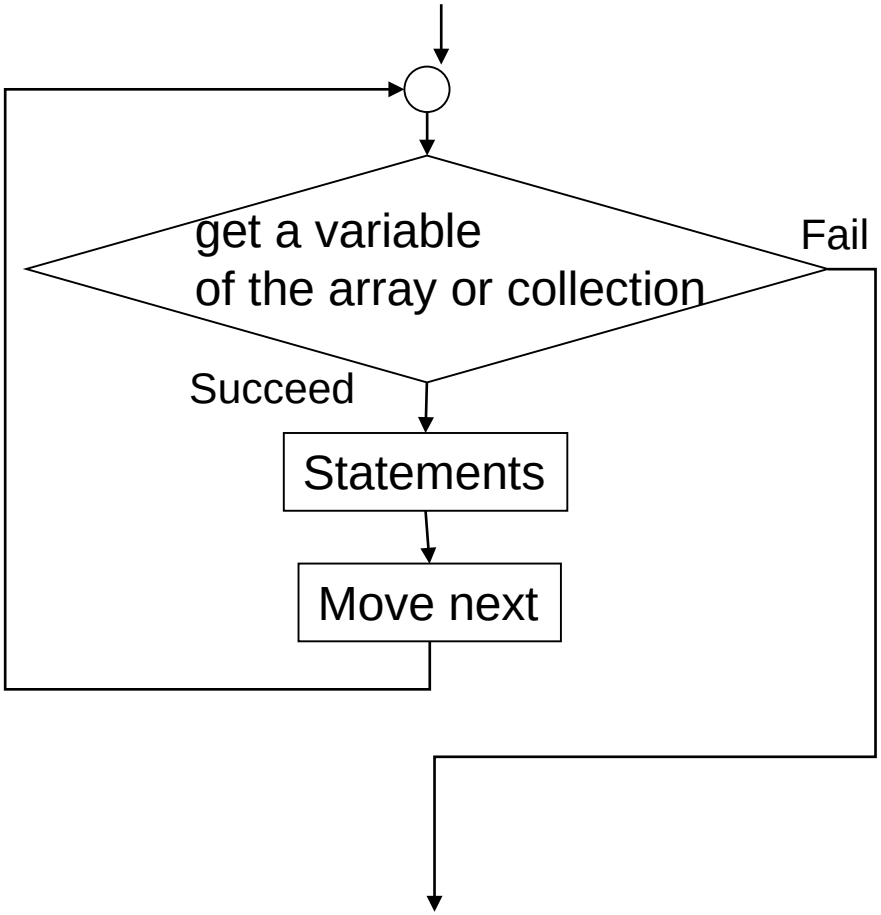
```
>for.exe
SUM(1:10) = 55

>
```

For Each ... Next

For Each *Variable* in *Group*
Statements
Next [*Variable*]

Variable:Variable used to iterate through the elements of the array or collection
group:Name of an object array or collection

<Flow of Processing>



Usage example of For Each ... Next

<Example of Program>

```
Module Module1

    Sub Main()
        Dim sNames() As String
        sNames = {"Chhom Makara", "Chhom Seyha", "Keo Sereyroth"}

        For Each sName As String In sNames
            Console.WriteLine(sName)
        Next
    End Sub

End Module
```

<OUTPUT>

```
>ForEach.exe
Chhom Makara
Chhom Seyha
Keo Sereyroth
```

3.4 Jump Statement

1- Exit : is used to exit from Loop or Sub Routine.

2- Continue : is used to skip one step in loop.

3- Goto Label : is used to change the starting point of running code.

Example of Exit Loop(For)

```
Module Module1
    Sub Main()
        Dim counter As Integer
        For counter = 1 To 10
            If counter = 3 Then
                Exit For
            End If
            Console.WriteLine(counter)
        Next
        Console.Read()
    End Sub
End Module
```

<OUTPUT>

```
>Exit.exe
1
2
```

Example of Exit Sub

```
Module Module1
    Sub Main()
        Dim counter As Integer
        For counter = 1 To 10
            If counter = 3 Then
                Exit Sub
            End If
            Console.WriteLine(counter)
        Next
        Console.Read()
    End Sub
End Module
```

Example of Continue

```
Module Module1
    Sub Main()
        Dim counter As Integer
        For counter = 1 To 10
            If counter = 3 Then
                Continue For
            End If
            Console.WriteLine(counter)
        Next
        Console.Read()
    End Sub
End Module
```

<OUTPUT>

```
>Continue.exe
1
2
4
5
6
7
8
9
10
```

Example of Goto Label

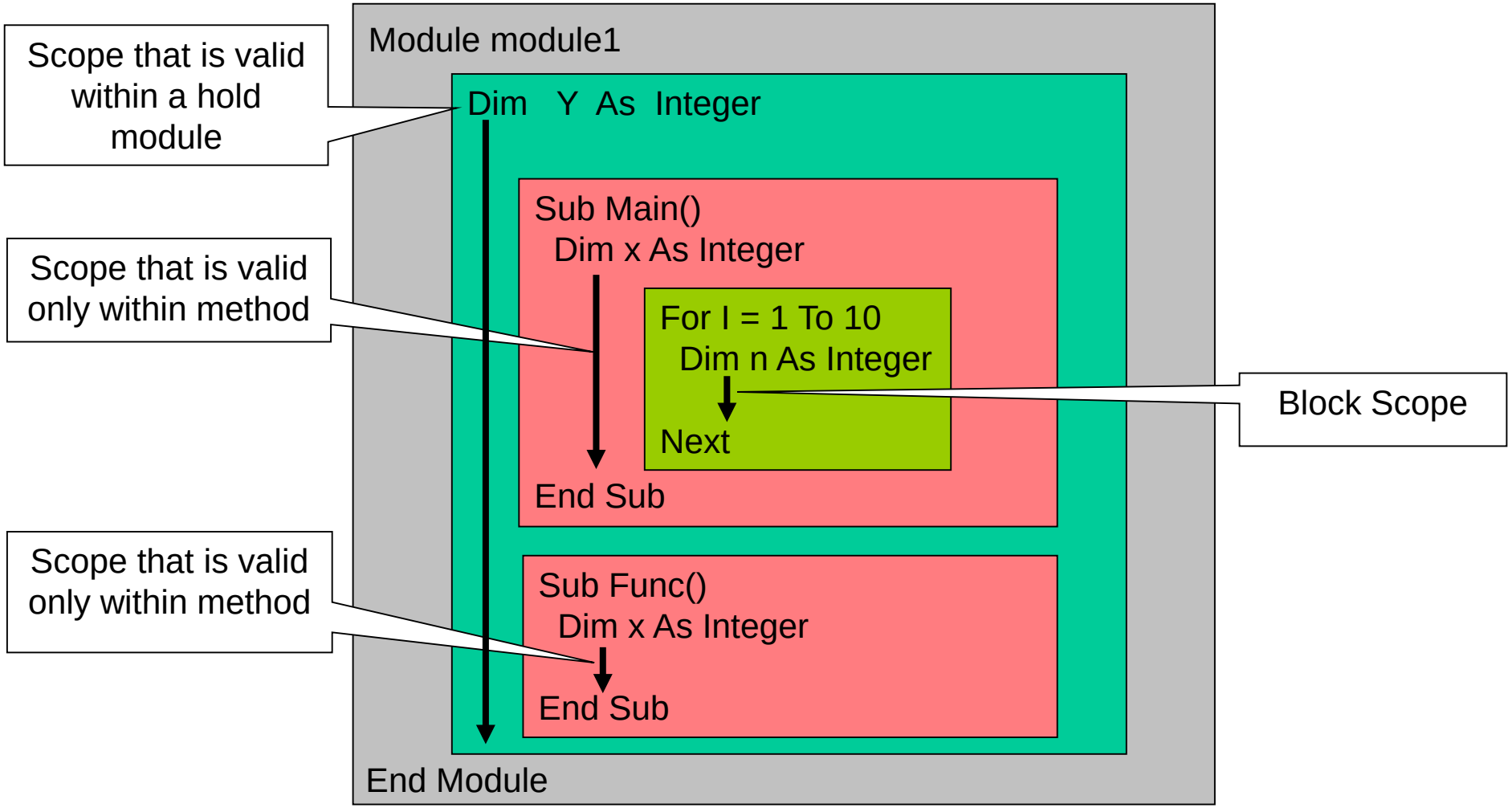
```
Module Module1

  Sub Main( )
    Dim counterVariable As Integer = 0

    repeat:
    Console.WriteLine("counterVariable: {0}",
                                  counterVariable)

    counterVariable += 1
    If counterVariable < 10 Then
      GoTo repeat
    End If
  End Sub

End Module
```

<OUTPUT>

```
>GotoLabel.exe
counterVariable: 0
counterVariable: 1
counterVariable: 2
counterVariable: 3
counterVariable: 4
counterVariable: 5
counterVariable: 6
counterVariable: 7
counterVariable: 8
counterVariable: 9
```

3.5 Scope

The usage range of variable names etc., is known as Scope

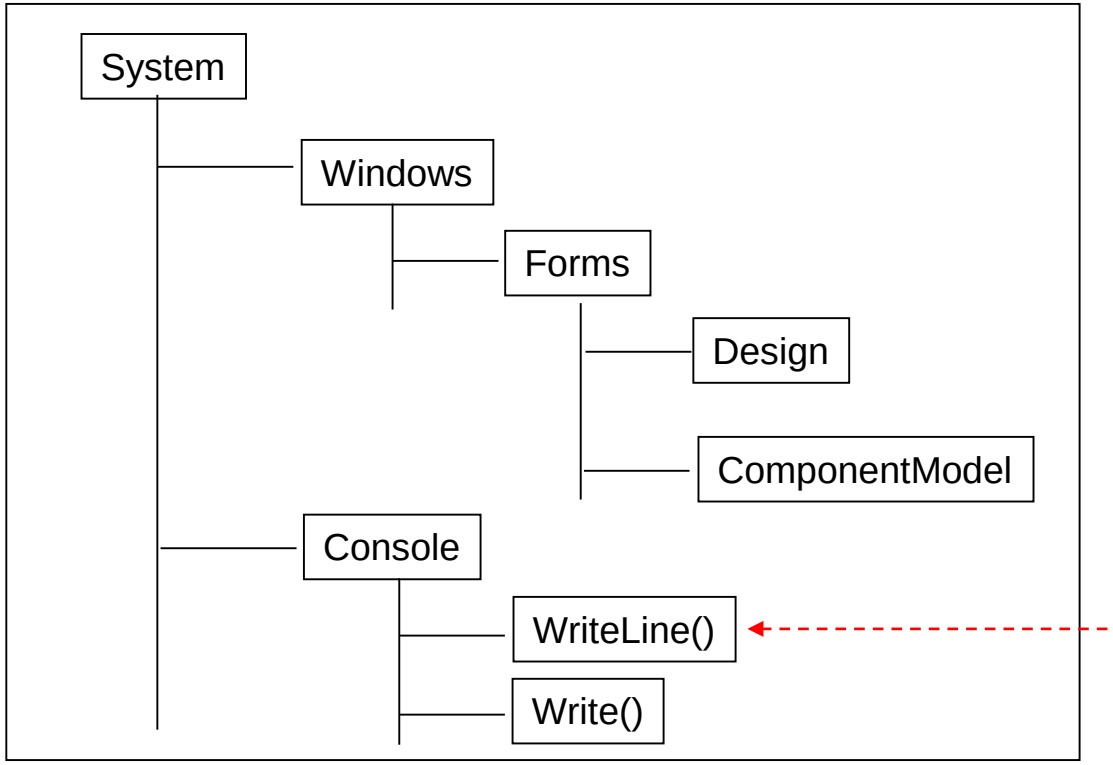


Namespace

Class,Interface, Method etc., are managed by Namespace, so as to prevent conflict in names

- Namespace has a hierarchical structure

(Example of .NET Framework Class Library)



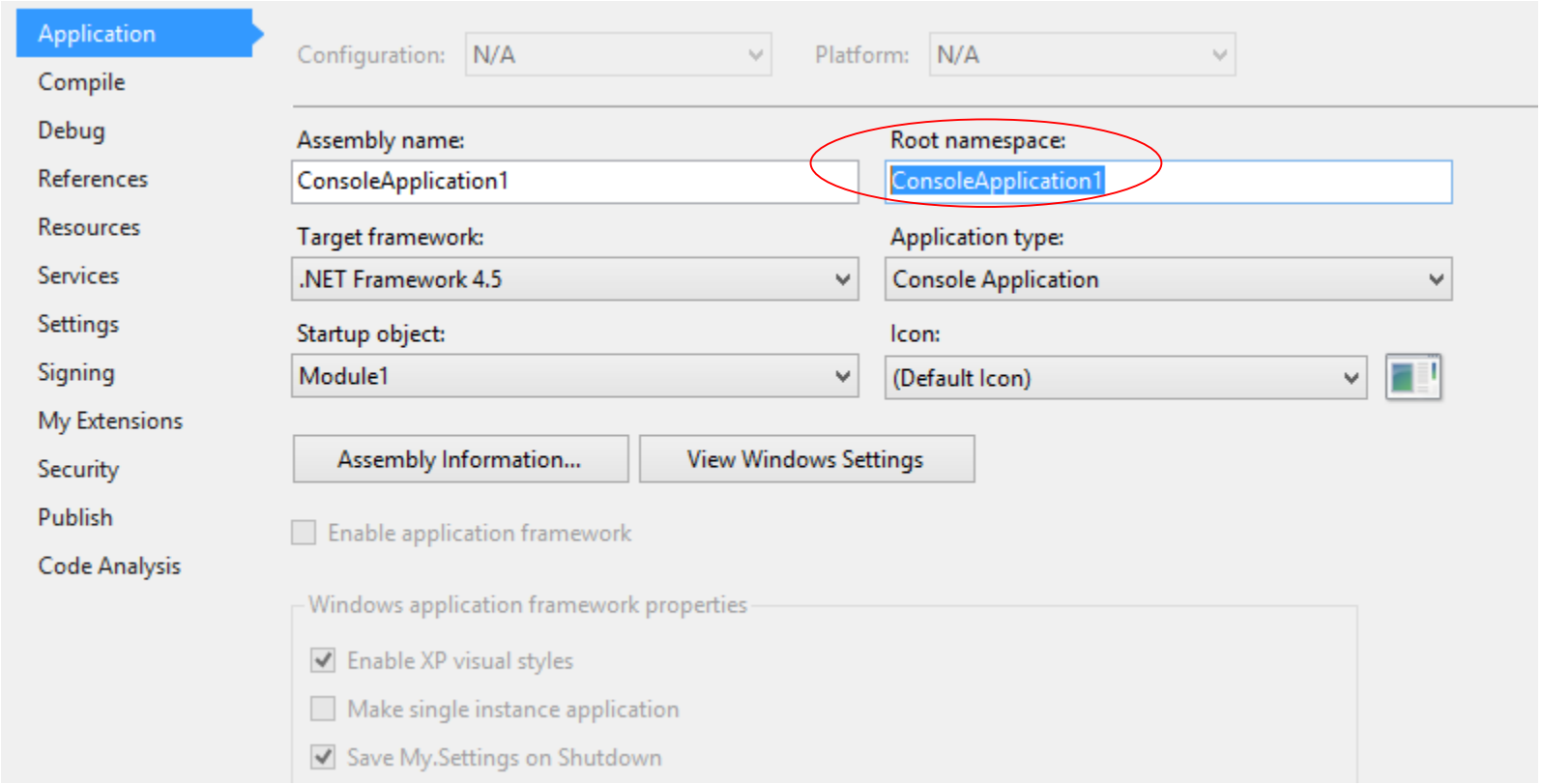
Console.WriteLine() belongs to System Namespace, and Namespace name is ”System.Console.WriteLine()”

Usage of Namespace within the Program

```
Namespace AnotherNamespace
    Module AnotherModule
      Sub ShowMessage()
          Console.WriteLine("Hello, world")
       End Sub
    End Module
End Namespace

Namespace MainNamespace
    Module MainModule
        Sub Main()
            AnotherNamespace.AnotherModule.ShowMessage()
        End Sub
     End Module
End Namespace
```

Application Namespace

Within the application created in Visual Studio .NET, a unique root namespace has been set



Imports

By using the Imports keyword, a long namespace name can be abbreviated and defined

```
Module MainModule
    Sub Main()
        Dim frm As System.Windows.Forms.Form
    End Sub
End Module
```

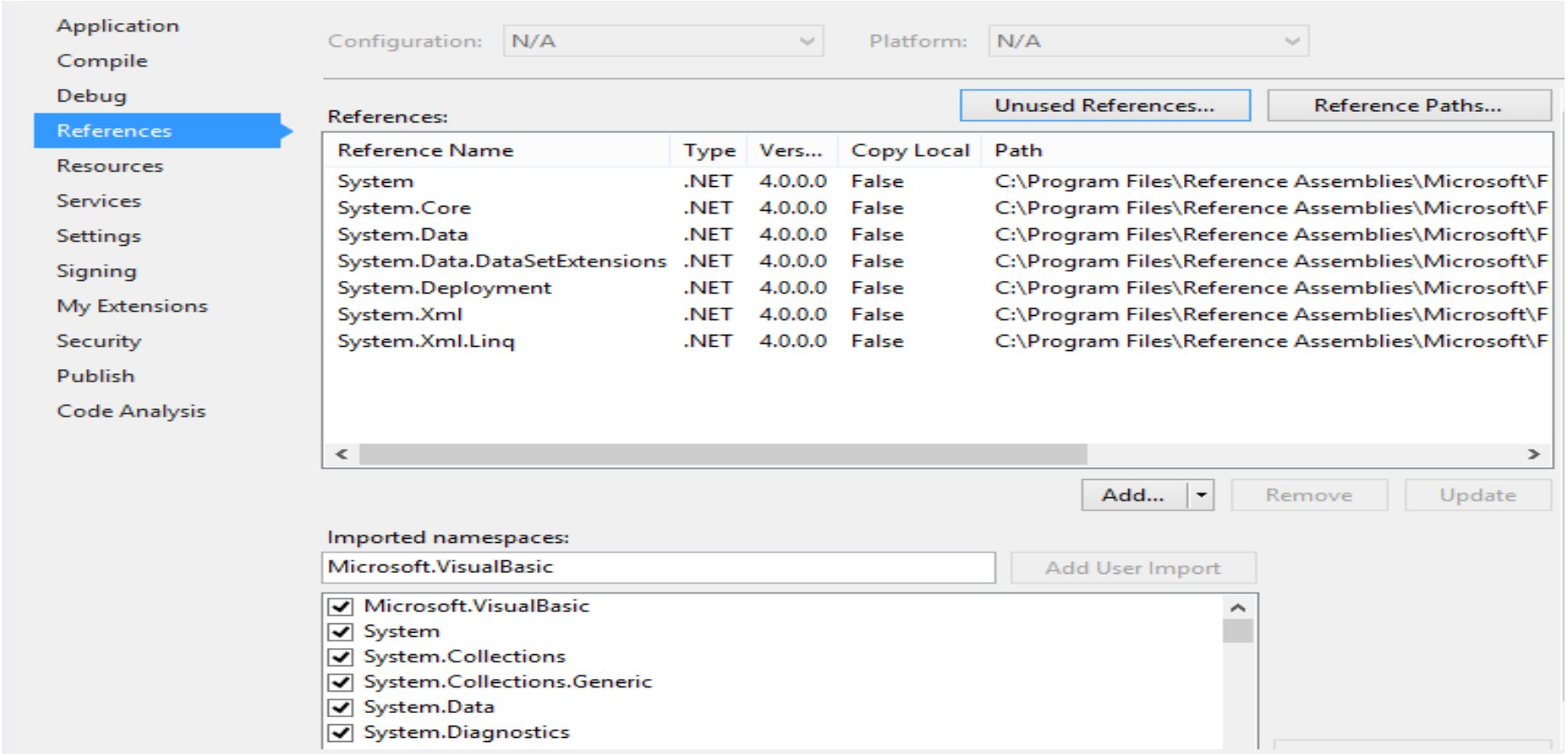
The above has the same meaning as that here under

```
Imports System.Windows.Forms

Module MainModule
    Sub Main()
        Dim frm As Form
    End Sub
End Module
```

Refers to
Dim frm As System.Windows.Forms.Form

Imports specification in Visual Studio .NET

When programming in Visual Studio .NET, it is possible to specify the Imports namespace in project property

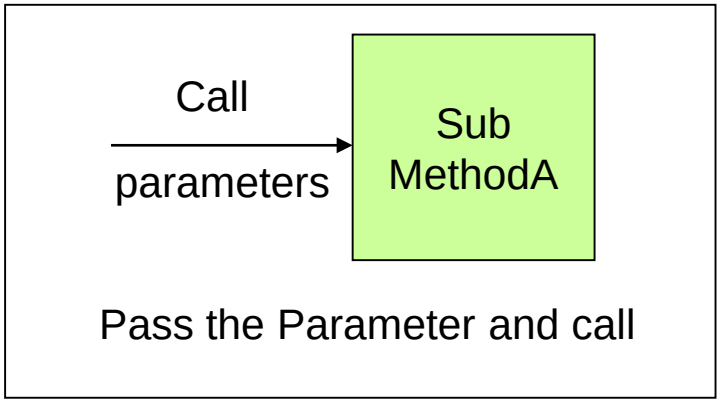


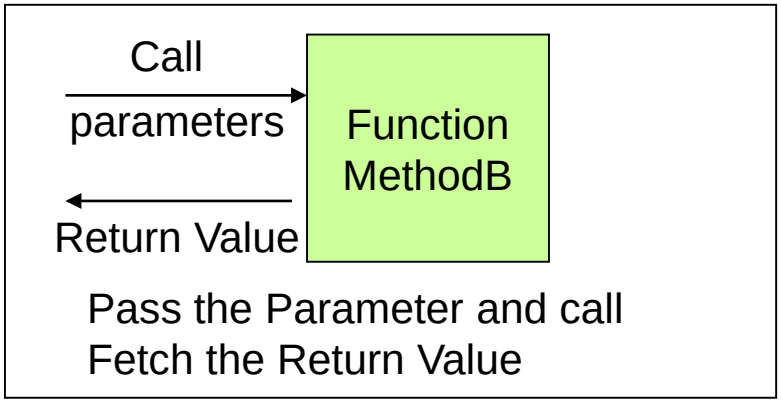
4. Method

4.1 Create Method

A method refers to a series of statements having a name and specific function

- Pass the data and perform a series of processes
- Methods can be classified into 2 categories
  - Sub Method: Performs only the process, and does not return a value
  - Function Method: Performs the process and returns a value
- Specific processes (numeric value calculations, DB access, data output etc.), that are run repeatedly, can be categorized as method





Sub Method

```
[Accessibility] Sub MethodName (ParamList)
    MethodStatements
End Sub
```

Accessibility Specify Access Modifiers such as Public, Private etc
(For Access Modifiers, find the explanation in Chapter 5)

MethodName Name of method

ParamList Parameter list of method

```
(ByVal Prm1 As DataType1, ByRef Prm2 As DataType2, -----)
```

(For ByVal and ByRef, refer to “ByVal and ByRef”)

The Character string passed in parameter str, is outputted only for the frequency mentioned in parameter n

```
Public Sub WriteNTimes(ByVal str As String, ByVal n As Integer)
    Dim i As Integer
    For i = 0 To n - 1
        Console.WriteLine(“{0}”, str)
    Next
End Sub
```

Function Method

```
[Accessibility] Function MethodName (ParamList) As ReturnType
    MethodStatements
End Function
```

Accessibility	Specify Access Modifiers such as Public, Private (For Access Modifiers, find the explanation in Chapter 5)
MethodName	Name of method
ParamList	Parameter list of method (Description method is the same as Sub method)
ReturnType	Specify the Return Value Data Type

- Return Value defined using the Return statement within the method itself

```
Return ReturnValue
```

Fetch the area for the circle whose radius was passed in parameter radius, and return the result

```
Public Function CalcCircleArea ( ByVal radius As Double ) As Double
    Dim Pi As Double = 3.14159
    Return Pi * radius ^ 2
End Function
```

4.2 Method Call

- Method defines method name and the continuing parameter within (), and then calls

MethodName (*Prm1, Prm2, ---*)

- When there is no parameter, () can be omitted

MethodName

- In method, calling oneself is known as recursive call

Example of Sub Method Call

```
Module Module1

    Sub WriteNTimes(ByVal str As String, ByVal n As Integer)
        Dim i As Integer
        For i = 1 To n
            Console.WriteLine("{0}: {1}", i, str)
        Next
    End Sub

    Sub Main()
        WriteNTimes("Hello!", 3)
    End Sub

End Module
```

Method Declaration

Assign ”Hello” and 3 as the parameters, and call WriteNTimes method

<OUTPUT>

```
1: Hello!
2: Hello!
3: Hello!
```

Example of Function Method Call

```
Module Module1

    Public Function CalcCircleArea(ByVal radius As Double) As Double
        Dim Pi As Double = 3.14159
        Return Pi * radius ^ 2
    End Function

    Sub Main()
        Dim  ans  As  Double
        ans  =  CalcCircleArea(3)
        Console.WriteLine("Circle Area is {0}", ans)
    End Sub

End Module
```

Method Declaration

Assign the Return Value of CalcCircleArea as ans

<OUTPUT>

```
Circle Area is 28.27431
```

ByVal and ByRef

ByVal and ByRef are used as the passing methods of parameter

- ByVal refers to Call-by-Value
  When calling a method, pass a copy of the value as the parameter. For that reason, the operations performed for parameter within the method do not affect the value of the calling source variable.

- ByRef refers to Call-by-Reference
  When calling a method, pass the reference as the parameter. For that reason, the operations performed for parameter within the method affect the value of the calling source variable.

The Difference Between ByVal and ByRef (1/2)

the difference between Call-by-Value and Call-by-Reference in value type

```
Module Module1
    Sub TestByVal(ByVal prm As Integer)
        prm = 100
    End Sub

    Sub TestByRef(ByRef prm As Integer)
        prm = 100
    End Sub

    Sub Main()
        Dim i As Integer
        i = 0
        TestByVal(i)
        Console.WriteLine("ByVal: {0}", i)
        i = 0
        TestByRef(i)
        Console.WriteLine("ByRef: {0}", i)
    End Sub
End Module
```

Call-by-Value

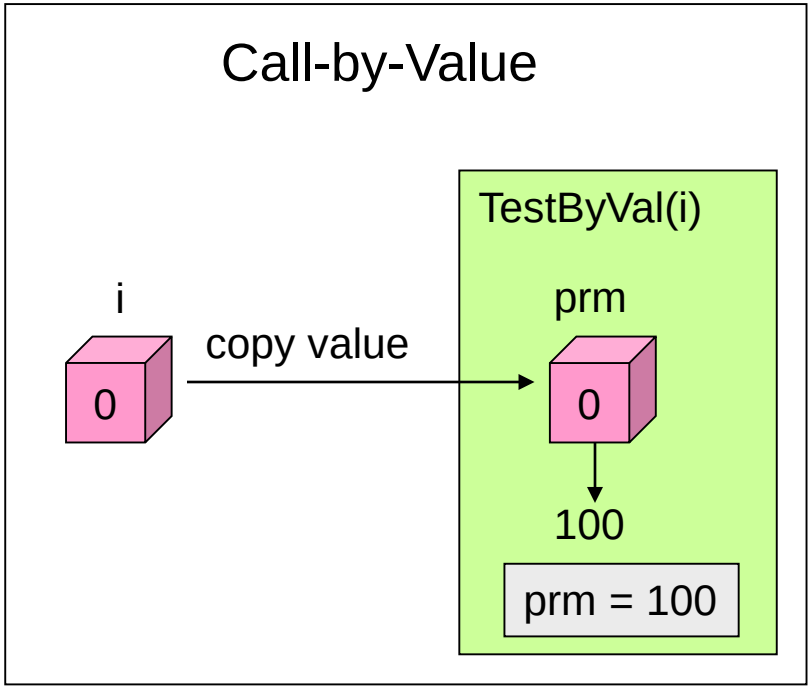
Call-by-Reference

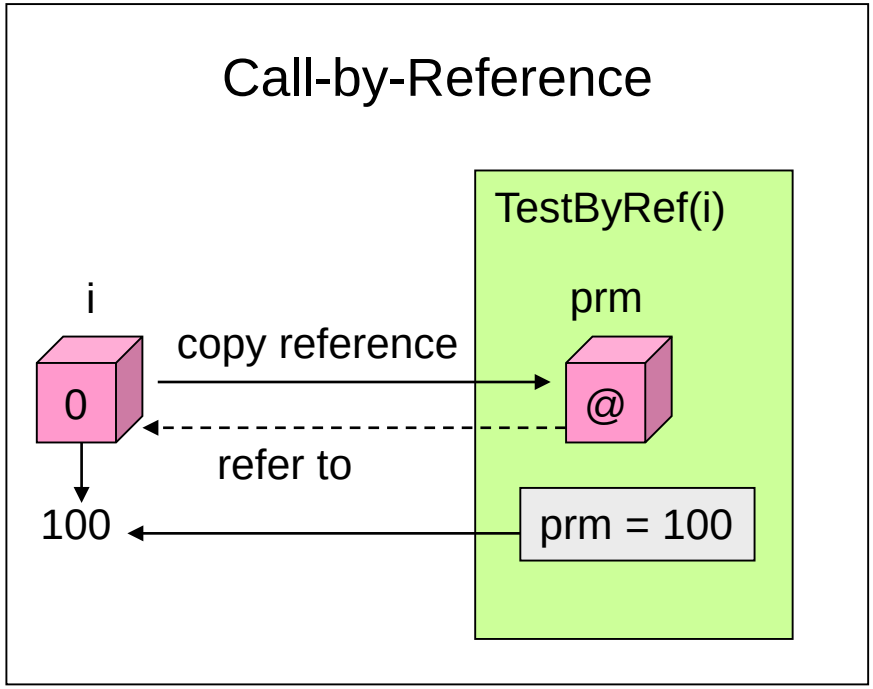
How the difference?

The Difference Between ByVal and ByRef (2/2)

<OUTPUT>

```
ByVal: 0
ByRef: 100
```





Parameter Arrays (1/2)

- When multiple, unspecific parameters are set, use ParamArray keyword
- ParamArray parameter can be used only in ByVal
- A typical example being, Console.WriteLine

Call pattern of Console.WriteLine

```
Console.WriteLine("1 param: {0}", 1)
Console.WriteLine("2 params: {0}  {1}", 1, 2)
Console.WriteLine("3 params: {0}  {1}  {2}", 1, 2, 3)
Console.WriteLine("3 params: {0}  {1}  {2}  {3}", 1, 2, 3, 4)
```

The number of parameters after the 2nd parameter can vary

Console.WriteLine uses parameter array in the 2nd parameter, as seen hereunder. For that reason, the parameters after the 2nd parameter can be of any number

```
Sub WriteLine( ByVal format As String, ByVal ParamArray arg() As Object )
```

Parameter Arrays (2/2)

```
Module Module1
    Function Sum(ByVal ParamArray args() As Integer) As Integer
        Dim i, n As Integer
        For Each i In args
            n += i
        Next
        Return n
    End Function
    Sub Main()
        Console.WriteLine(Sum(1))
        Console.WriteLine(Sum(1, 2, 3, 4, 5))
        Console.WriteLine(Sum(New Integer() {1, 2, 3}))
        Console.readline
    End Sub
End Module
```

The number of Parameters vary

<OUTPUT>

```
1
15
6
```

Optional Parameters

- By using Optional keyword, the default value of the parameter can be defined

```
Module Module1
   Public Sub  WriteNTimes(ByVal  str  As  String,  _
                                Optional ByVal  n  As  Integer  =  1)
      Dim  i  As   Integer
      For  i  =  1  To  n
        Console.WriteLine("{0}: {1}", i, str)
      Next
   End  Sub
   Sub Main()
      WriteNTimes("Hello!",  2)
      WriteNTimes("World")
      console.readline()
   End Sub
End Module
```

For the 2nd parameter, the default value of 1 is specified. In other words, it has the same meaning as WriteNTimes("World", 1)

<OUTPUT>

```
1: Hello!
2: Hello!
1: World
```

Overloading Method

Overload refers to the definition of multiple methods having difference parameter lists with the same name.

- It is used when defining multiple methods, where the parameters differ, but the functionality is the same

Fetch the area for the circle whose radius was passed in parameter radius, and return the result

```
Public Function CalcCircleArea(ByVal radius As Double) As Double
    Dim pi As Double = 3.14159
    Return pi * radius ^ 2
End Function
```

Parameters differ

Calculate the area for the circle using the radius that was passed in parameter radius, and circumference ratio that was passed in pi

```
Public Function CalcCircleArea(ByVal radius As Double, _
                               ByVal pi As Double) As Double
    Return pi * radius ^ 2
End Function
```

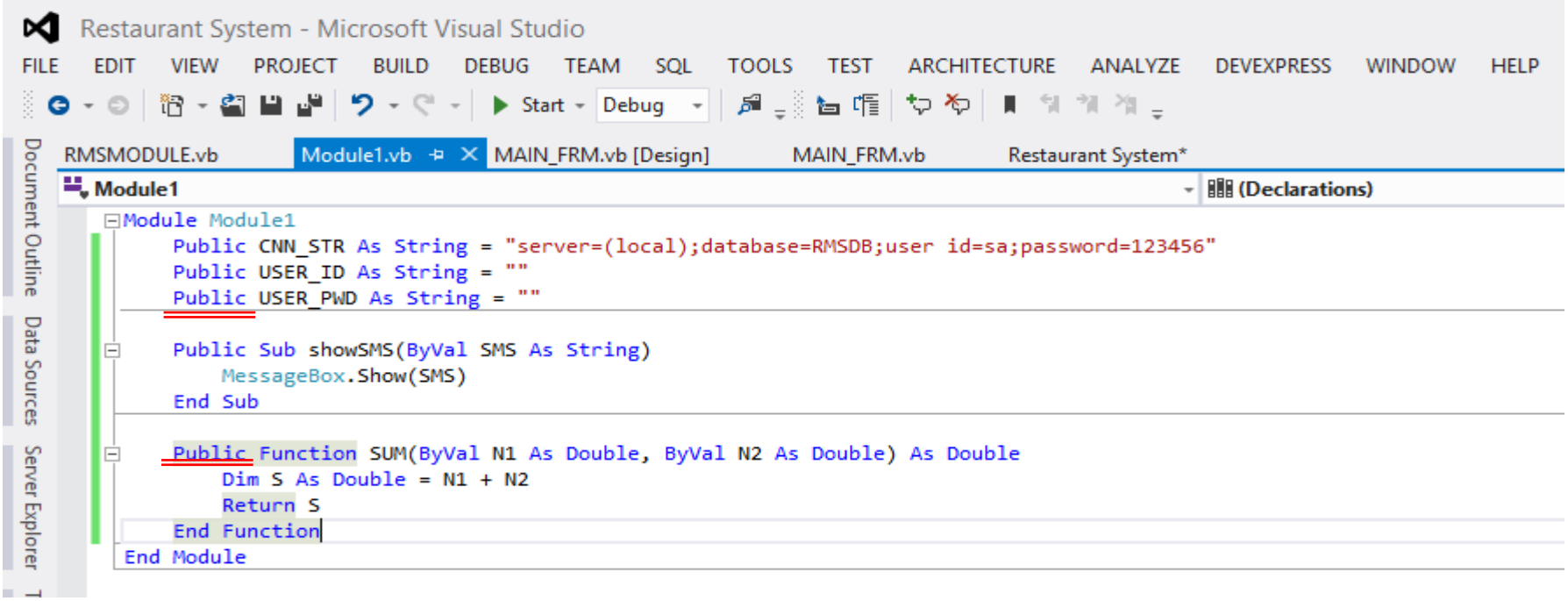
Module

Module is a special class that created itself when program is executing.

Purpose:

Module is used to define global variables or methods.

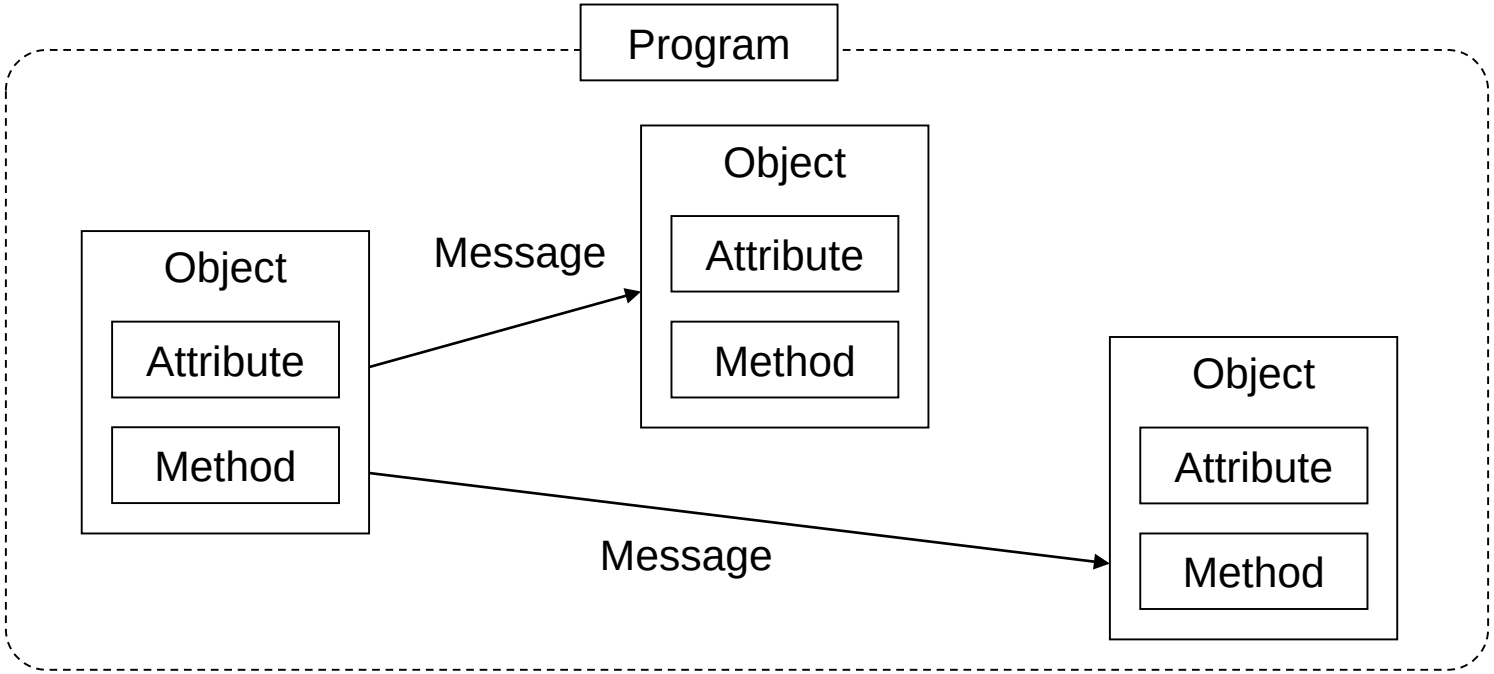
Right-click on [Project]→[Add]→[New Item]→[Module]



5. Object Oriented Programming

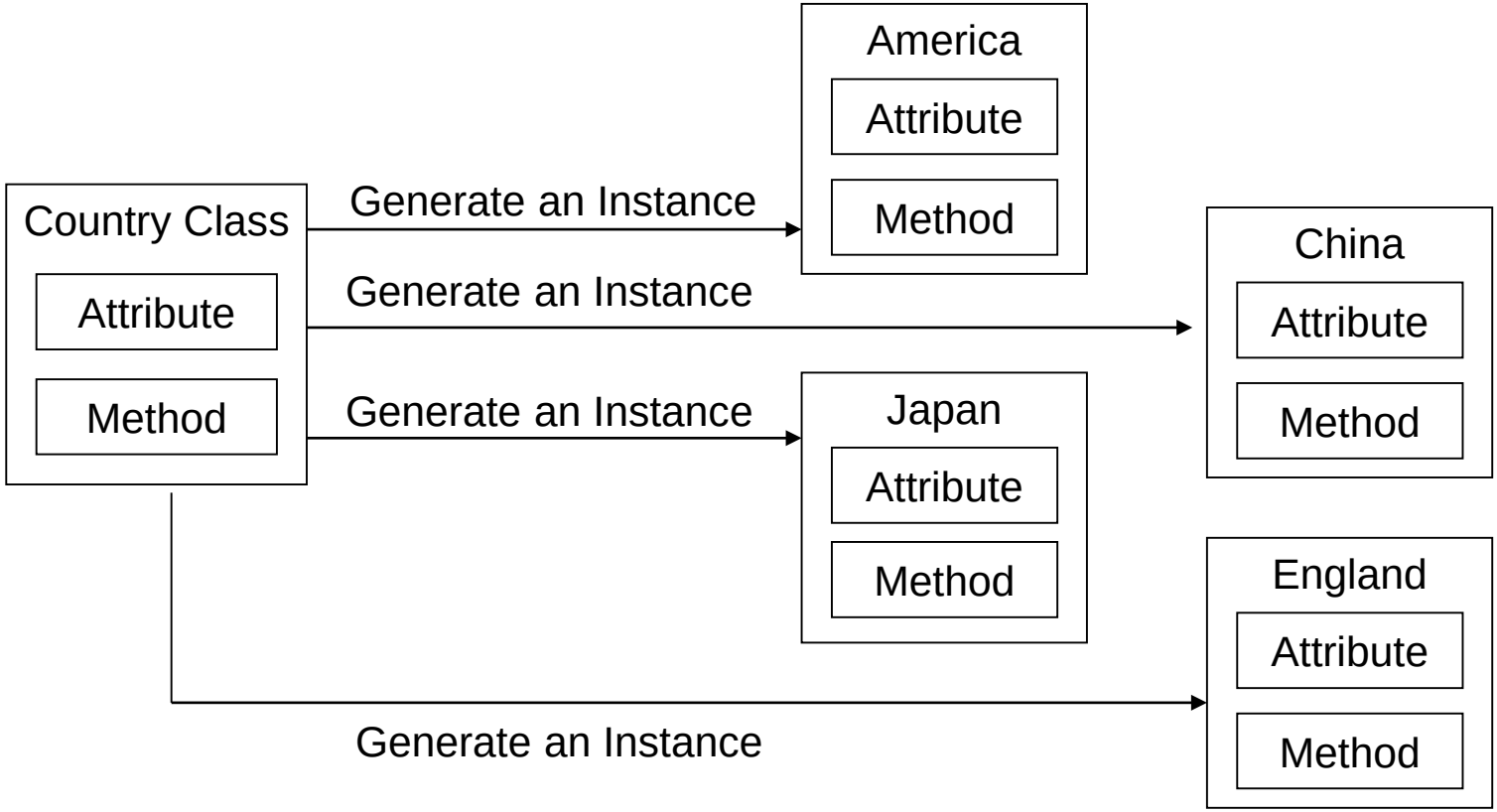
5.1 Introduction to OOP

- Implement the software in units known as objects
- Object is composed of attributes and methods
- Each object has its own role
- The process is implemented by exchanging messages between objects



5.2 Class and Object

- Class is the template of the object
- By generating an instance of class, object is generated
- Multiple objects can be generated from 1 class



Class Declaration

- Class is defined as hereunder

```
[Accessibility]  [ClassModifier]  Class  ClassName
      [Inherits  SuperClassName]
      [Implements  InterfaceName]
   [AttributeStatements]
   [MethodStatements]
End Class
```

Accessibility	Access Modifiers such as Public, Private etc.
ClassModifier	Modifiers such as MustInherit, NotInheritable etc.
ClassName	Name of Class
SuperClassName	Name of SuperClass
InterfaceName	Interface name to be implemented

Example:Dog Class

```
Public Class Dog
  Private Name As String
  Private Age  As  Integer

  Public Sub New(str As String)
     Name = str
  End Sub

  Public Sub Bark()
    Console.WriteLine(Name+“:bowwow")
  End Sub

End Class
```

Attribute

Method

Dog
Name Age
New() Bark()

Generate Instance

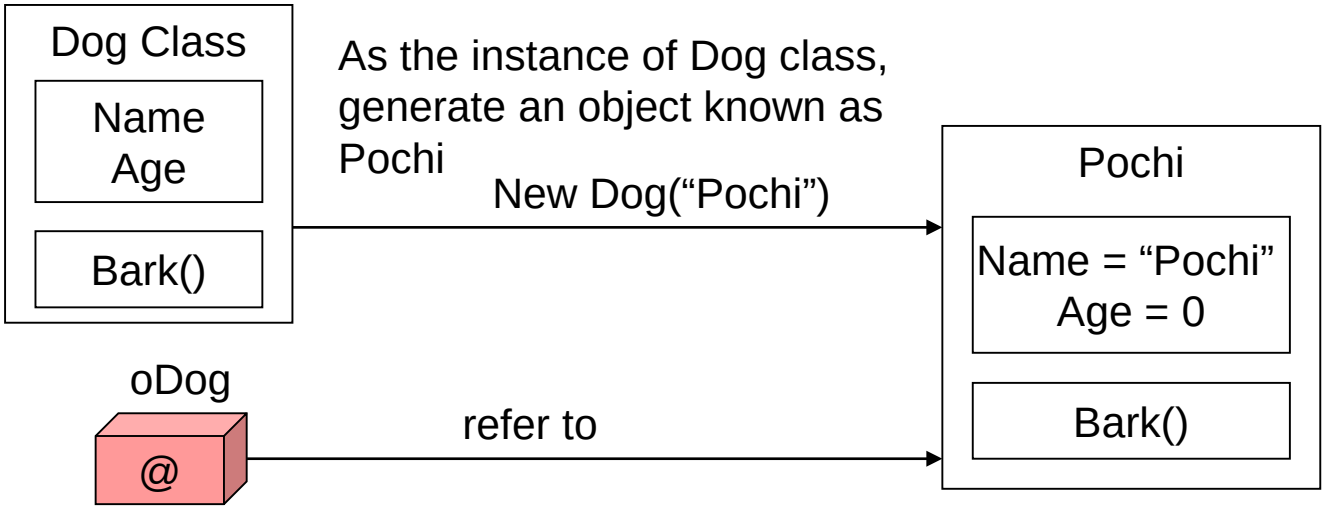
- Generating an object from class is known as instance generation
- In instance generation, use the keyword known as New

```
Dim oDog As Dog
oDog = New Dog("Pochi")
```

```
Dim oDog As New Dog("Pochi")
```

When As New is mentioned, variable declaration and Instance generation can be mentioned in one row

- Memory is allocated in the object, and reference is substituted in the variable



Example of Instance generation

```
Sub Main()
    Dim d1, d2 As Dog
    d1 = New Dog(“Pochi")
    d2 = New Dog(“Hachi")

    d1.Bark()
    d2.Bark()
End Sub
```

Generate an Instance

Call Method

<OUTPUT>

```
Pochi: bowwow
Hachi: bowwow
```

Constructor

- Constructor is the special method that is automatically called when an instance of class is generated, and is defined in Sub New
- Describe the process for class initialization
- It is also possible to define the constructor that has parameters

```
Sub New([ParamList])
```

```
Public Class Dog
    Private mName As String
    Sub New()
        mName = "No name"
    End Sub
    Sub New(ByVal str As String)
        mName = str
    End Sub
End Class
```

Constructor without parameters

Constructor having parameters
When initializing, any initial value can be set in mName

5.3 Attribute and Method

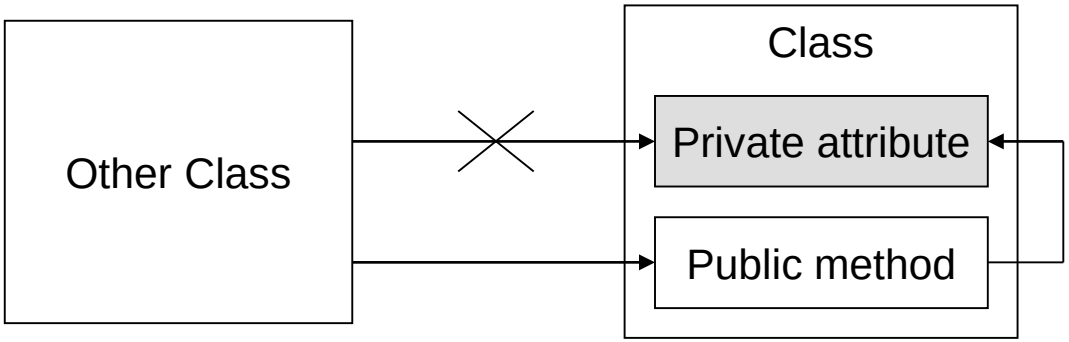
- In class, attribute and method are implemented
- Method refers to the behavior of the object, where each type of process is carried out by calling a method
- Attribute refers to the data contained within class, and is implemented as member variable. For example
  - Human Class Attribute : Name, Sex, Age, Job, etc.
  - Country Class Attribute : Name, Latitude, Longitude, etc.
- Member variable contains data type, and is defined as hereunder

[*Accessibility*] [*Modifier*] *VariableName* As *DataType*

Accessibility	Access modifiers such as Public, Private
Modifier	Member variable modifiers such as Shared, ReadOnly etc
VariableName	Name of member variable
DataType	Data type of member variable

Encapsulation

- Encapsulation is one of the important concepts of OOP
- The attributes within class are hidden from the outside, and can be accessed only from method
- Even if the internal implementing of class is changed, there is no effect on the side using the class



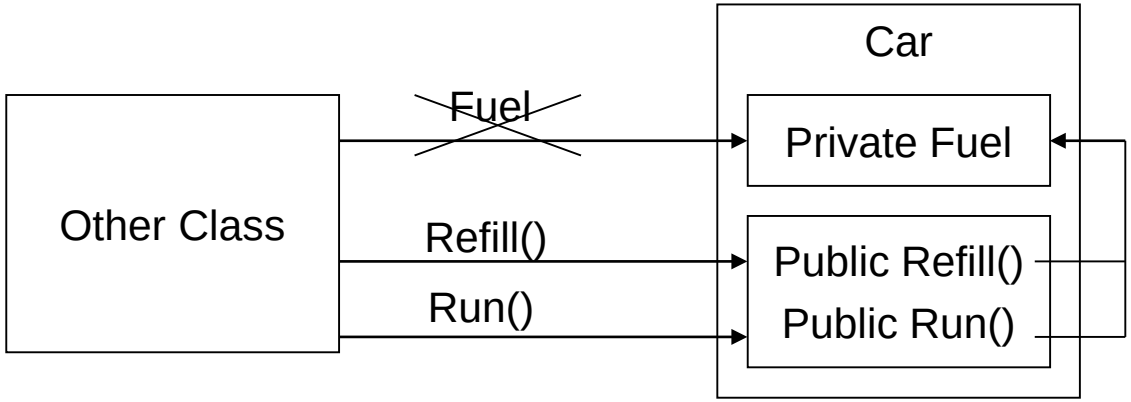
Since Private attribute is hidden from the outside, it cannot be accessed directly, and is accessed using Public method

Accessibility

- The method for accessing attribute or method of class from the outside is defined by accessibility

  The main accessibility methods are the following three

  - Private: Cannot access from other than the same class
  - Public: Can access from any class
  - Protected: Can access from the same class and from the class inherited from that class
  - Friend: Next Slide



Encapsulation Example (1/2)

```
Public Class Car
    Private Fuel As Integer

    Public Sub Refill(ByVal n As Integer)
        If Fuel + n >= 40 Then
            Fuel = 40
        Else
            Fuel += n
        End If
    End Sub

    Public Sub Run()
        If Fuel - 15 < 0 Then
            Fuel = 0
        Else
            Fuel -= 15
        End If
    End Sub

    Public Sub PrintFuel()
        Console.WriteLine("Fuel is {0}", Fuel)
    End Sub
End Class
```

Member variable that represents fuel

Supply fuel such that 40L is not exceeded

When in motion, fuel reduces by 15L each time
The fuel does not become minus

Encapsulation Example (2/2)

```
Sub Main()
    Dim mycar As New Car

    mycar.Refill(30)
    mycar.PrintFuel()
    mycar.Refill(30)
    mycar.PrintFuel()

    mycar.Run()
    mycar.PrintFuel()
    mycar.Run()
    mycar.PrintFuel()
    mycar.Run()
    mycar.PrintFuel()
End Sub
```

30L is supplied twice

Run() is called thrice

<OUTPUT>

```
Fuel is 30
Fuel is 40
Fuel is 25
Fuel is 10
Fuel is 0
```

Re-fuelling is performed twice by 30L each time but Fuel does not exceed 40L

Run() is called 3 times but, Fuel does not go below 0L

Property

- Property is the special method for implementing encapsulation
- Property is usually combined with Private member variable and used
- The definition of property is made up of Set block that sets the value and Get block that fetches the value

```
[Accessibility]  [ReadOnly | WriteOnly] Property  PropertyName()  As  DataType
   Get
      Return  ReturnValue
   End Get
   Set (ByVal Value As DataType)
      Variable  =  Value
   End Set
End Property
```

Accessibility	Access modifiers such as Public, Private etc
ReadOnly	As the Read only property, define only Get block
WriteOnly	As the Write only property, define only Set block
PropertyName	Name of property
DataType	Data type of property
ReturnValue	Return value of property
Variable	Variable that sets data in property

Example of Property and Encapsulation

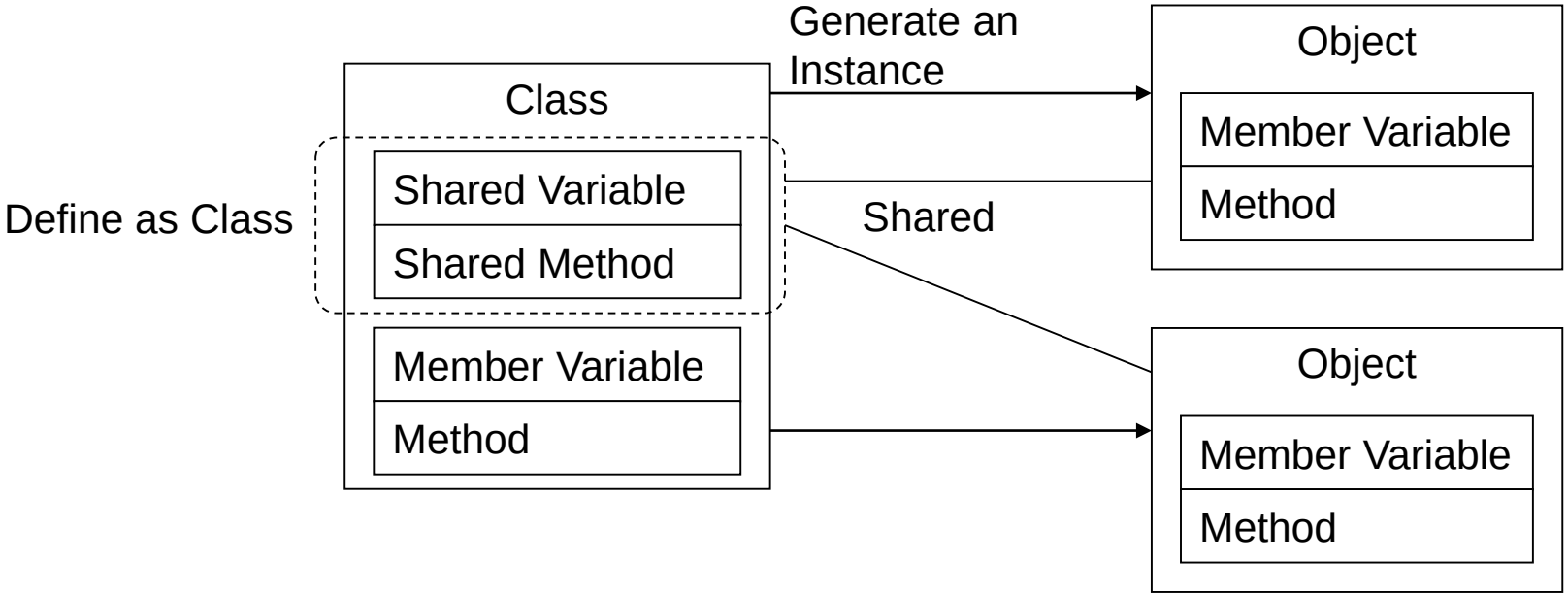
```
Public Class Dog
   Private mName As String

   Public  Property  Name()  As  String
      Get
         Return  mName
      End Get
      Set (ByVal Value As String)
         mName  =  Value.ToUpper
      End Set
   End  Property
End Class
```

```
Dim   pochi  As  Dog
pochi  =  New  Dog()

pochi.Name  =  “Pochi”
```

“Pochi” is set

Shared Attribute and Method

- The attribute or method that is defined for class and not for object
- Assign Shared keyword and define
- Access is possible even without generating an object



Example of System.Math Class

```
Module Module1

    Sub Main()
        Console.WriteLine(Math.PI)
        Console.WriteLine(Math.Abs(-10))
        Console.WriteLine(Math.Max(1, 100))
        Console.WriteLine(Math.Pow(2, 3))
        Console.WriteLine(Math.Sqrt(5))
    End Sub

End Module
```

Write Pi

Return Absolute Value

Return Maximum Value

Return Power

Return Square root

<OUTPUT>

```
3.14159265358979
10
100
8
2.23606797749979
```

Example of Shared Variable and Method

```
Module Module1
    Public Class TestClass
        Public I As Integer
        Public Shared S As Integer
    End Class

    Sub Main()
        Dim a As New TestClass
        Dim b As New TestClass
        a.I = 1
        b.I = 2
        TestClass.S = 3
        Console.WriteLine("a.I = {0} b.I = {1}  TestClass.S = {2}", _
        a.I, b.I, TestClass.S)
    End Sub
End Module
```

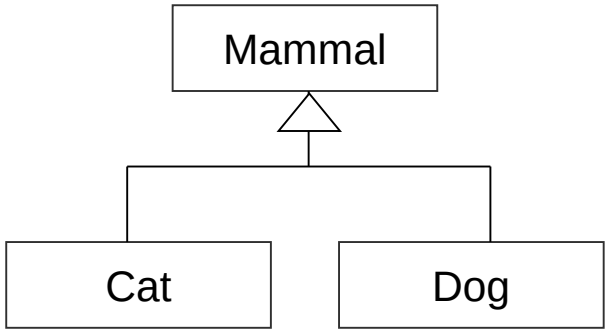
<OUTPUT>

```
a.I = 1  b.I = 2  TestClass.S = 3
```

5.4 Inheritance

- Inheritance is the defining of the newly extended class, with the already existent class as base
- The base class of inheritance is known as superclass, while that which is generated by inheritance is known as subclass
- Subclass inherits the attributes and methods of the superclass
- It is not possible to inherit multiple superclasses
- The object of subclass can be substituted with the superclass type variable

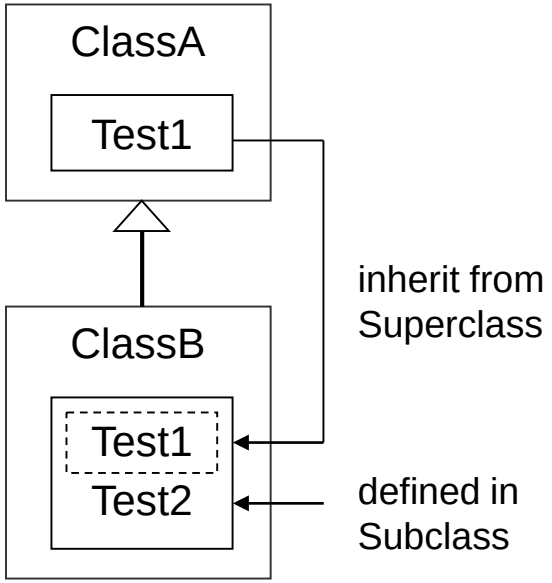
Example: Cat, Dog inherit Mammal




```
Dim  a,  b,  c  As  Mammal
a  =  New  Mammal()
b  =  New  Cat()
c  =  New  Dog()
```

Inheritance Example

```
Module Module1
    Public Class ClassA
        Public Sub Test1()
            Console.WriteLine("Test1 called")
        End Sub
    End Class

    Public Class ClassB
        Inherits ClassA
        Public Sub Test2()
            Console.WriteLine("Test2 called")
        End Sub
    End Class

    Sub Main()
        Dim a As New ClassA
        Dim b As New ClassB
        a.Test1()
        b.Test1()
        b.Test2()
    End Sub
End Module
```

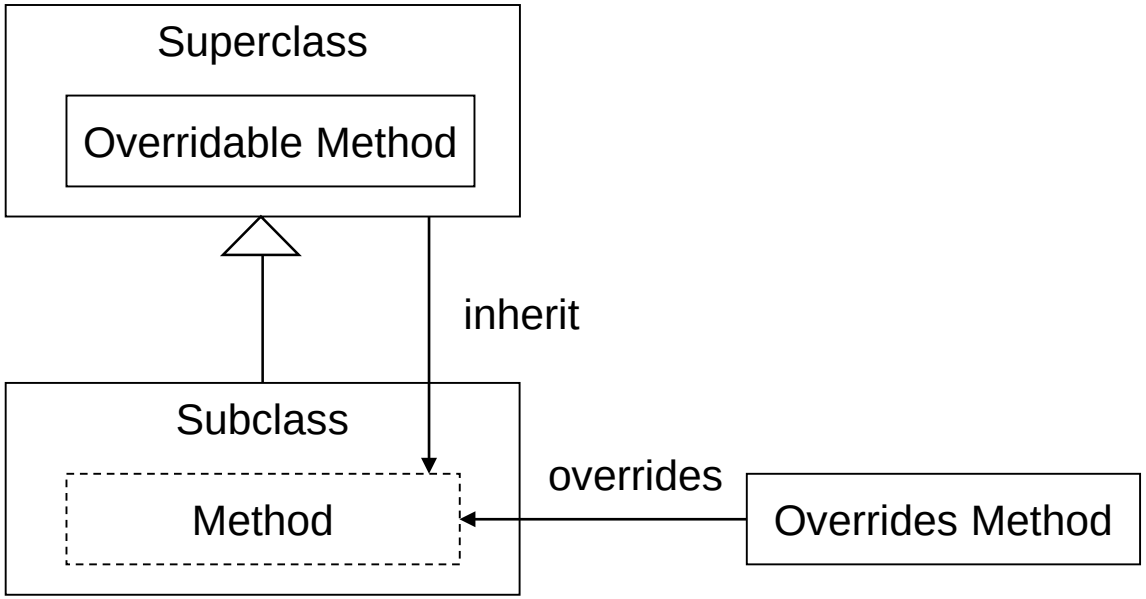


<OUTPUT>

```
Test1 called
Test1 called
Test2 called
```

Override

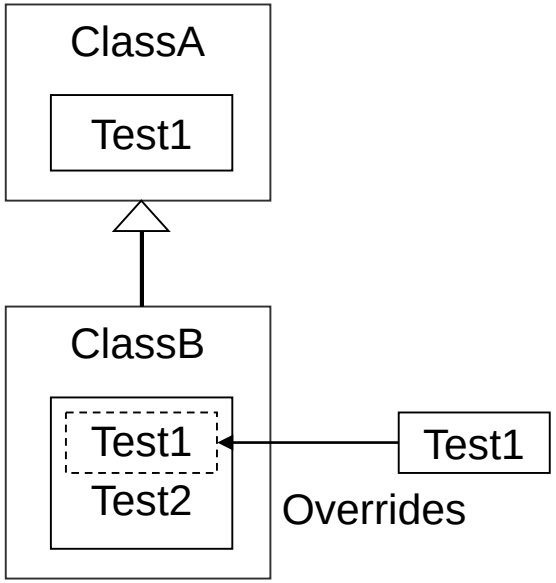
- Override is the re-definition of method that has been inherited on the subclass side
- Define using **Overrides** keyword, and perform only for the method that is defined as **Overridable** in superclass
- When calling a method of superclass, which is the base for Override, define as MyBase.Method()



Override Example (1/2)

The above mentioned ClassA, ClassB are modified below

```
Module Module1
    Public Class ClassA
        Public Overridable Sub Test1()
            Console.WriteLine("Test1 called")
        End Sub
    End Class

    Public Class ClassB
        Inherits ClassA
        Public Overrides Sub Test1()
            Console.WriteLine("Test1 of ClassB called")
        End Sub
        Public Sub Test2()
            Console.WriteLine("Test2 called")
        End Sub
    End Class

    Sub Main()
        Dim a As New ClassA
        Dim b As New ClassB
        a.Test1()
        b.Test1()
        b.Test2()
    End Sub
End Module
```



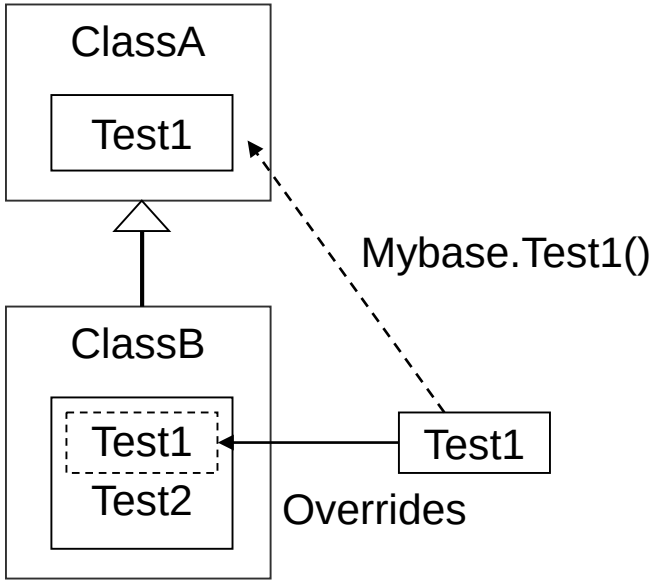
<OUTPUT>

```
Test1 called
Test1 of ClassB called
Test2 called
```

Override Example (2/2)

When performing Override, and when calling the source method

```
Module Module1
    Public Class ClassA
        Public Overridable Sub Test1()
            Console.WriteLine("Test1 called")
        End Sub
    End Class

    Public Class ClassB
        Inherits ClassA
        Public Overrides Sub Test1()
            Mybase.Test1()
            Console.WriteLine("Test1 of ClassB called")
        End Sub
        Public Sub Test2()
            Console.WriteLine("Test2 called")
        End Sub
    End Class

    Sub Main()
        Dim a As New ClassA
        Dim b As New ClassB
        a.Test1()  <---------- (1)
        b.Test1()  <---------- (2)
        b.Test2()  <---------- (3)
    End Sub
End Module
```



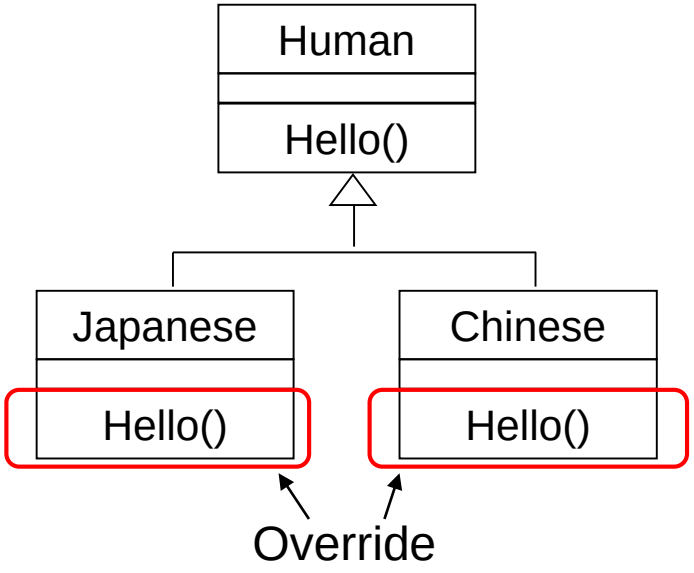
<OUTPUT>

```
(1) Test1 called
    Test1 called
(2) Test1 of ClassB called
(3) Test2 called
```

5.5 Polymorphism

Polymorphism refers to differing operations that are performed for each object even if the same method is called

For example, in the class group that has the under mentioned inheritance structure, the following code is run




```
Sub Main()
    Dim man(2) As Human
    Dim i As Integer
    man(0) = New Human
    man(1) = New Japanese
    man(2) = New Chinese

    For i = 0 To 2
        man(i).Hello()
    Next
End Sub
```

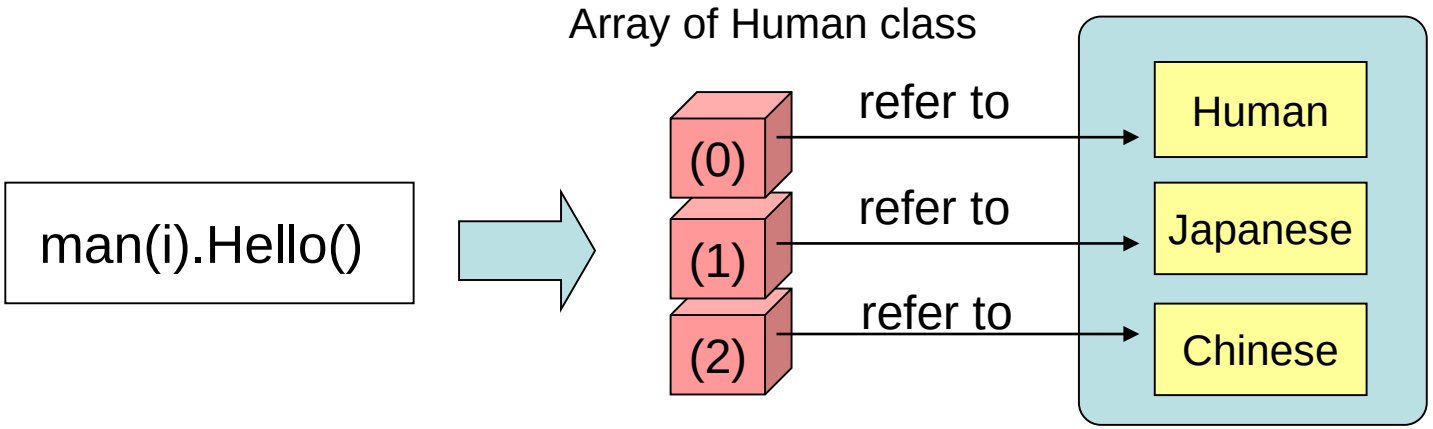
Polymorphism Example (1/2)

```
Public Class Human
    Public Overridable Sub Hello()
        Console.WriteLine("Hello")
    End Sub
End Class

Public Class Japanese
        Inherits Human
    Public Overrides Sub Hello()
        Console.WriteLine("Konnichiwa")
    End Sub
End Class

Public Class Chinese
        Inherits Human
    Public Overrides Sub Hello()
        Console.WriteLine("Nihao")
    End Sub
End Class
```

Polymorphism Example (2/2)

<OUTPUT>

Hello	Hello of Human class is called
Konnichiwa	Hello of Japanese class is called
Nihao	Hello of Chinese class is called



In this case, method is called not based on array type, but on object type that is actually referred to by each element of array

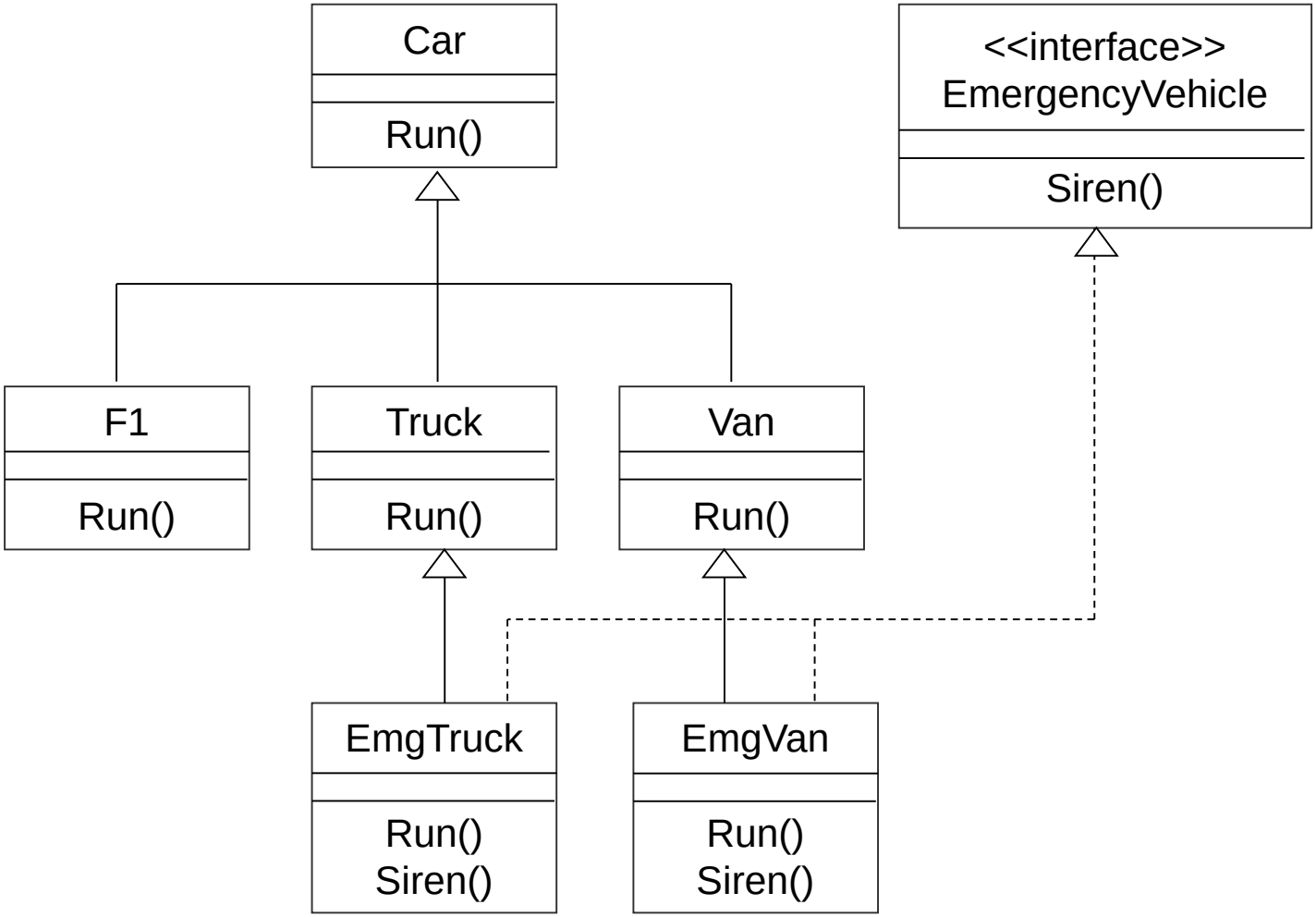
5.6 Interface

- Interface does not contain method implementation but only definition
- The class that implements interface has to define the method that contains the interface

```
[Accessibility]  Interface  InterfaceName
    [MethodDecls]
End Interface
```

Accessibility	Access modifiers such as Public, Private etc.
InterfaceName	Name of interface
MethodDecls	Interface method, property declaration

Interface Example (1/3)



Interface Example (2/3)

```
Module Module1
    Public Class Car
        Public Overridable Sub Run()
        End Sub
    End Class

    Public Class F1
        Inherits Car
        Public Overrides Sub Run()
            Console.WriteLine("F1 is running")
        End Sub
    End Class

    Public Class Truck
        Inherits Car
        Public Overrides Sub Run()
            Console.WriteLine("Truck is running")
        End Sub
    End Class

    Public Class Van
        Inherits Car
        Public Overrides Sub Run()
            Console.WriteLine("Van is running")
        End Sub
    End Class

    Public Interface EmergencyVehicle
        Sub Siren()
    End Interface
```

“” BE CONTINUED “”

Interface Example (3/3)

```
Public Class EmgTruck
    Inherits Truck
    Implements EmergencyVehicle
    Public Sub Siren() Implements EmergencyVehicle.Siren
        Console.WriteLine("Trcuk is blowing a siren")
    End Sub
End Class

Public Class EmgVan
    Inherits Van
    Implements EmergencyVehicle
    Public Sub Siren() Implements EmergencyVehicle.Siren
        Console.WriteLine("Van is blowing a siren")
    End Sub
End Class

Sub Main()
    Dim etruck As New EmgTruck
    Dim evan As New EmgVan

    etruck.Run()
    etruck.Siren()
    evan.Run()
    evan.Siren()
End Sub

End Module
```

<<OUTPUT>>

```
Truck is running
Trcuk is blowing a siren
Van is running
Van is blowing a siren
```

6. Exception and Debugging Tool

6.1 Errors

There are 2 types of Errors

- Syntax Error

  These are errors due to mistakes in the syntax of the program, and are found when performing Build

- Logical Error

The syntax is correct but they occur when the contents differ from the intended flow of the program. To detect a logical error, test the program automatically or manually, and it is necessary to check as to where the mistake occurs in the program

→ Use the debug function of Visual Studio.NET

6.2 Exception

Exception refers to error states or unexpected operations that the running program encounters

- Generally, an exception is caused when all types of errors occur

```
Module Module1
  Sub Main()
      Dim a As Integer
      Dim b As Integer = 0
      a = 10 \ b
  End Sub
End Module
```

DivideByZeroException occurs

Example of exception occurrence
The DivideByZeroException occurs when dividing by 0

Typical Exception

- Main reasons for exception occurrence
  - Numeric value overflow
  - Divide by 0
  - Insufficient memory
  - Index that is out of Array range
  - Failure in DB connectivity etc.

- Exception Class
  - Exception class is the class inherited from System.Exception class of .NET Framework Class Library(FCL)
  - Generally, a name such as ABCException is assigned
  - Eg: DivideByZeroException, NullReferenceException, FileNotFoundException, ArgumentOutOfRangeException etc.
  - It is also possible to create an Exception class unique to the user

Try ... Catch Statements

```
Try
      TryBlockStatements
[Catch  ex  As  Exception]
      [CatchBlockStatements]
[Finally]
       [FinallyBlockStatements]
 End  Try
```

TryBlockStatements	Write the code for when exception can occur
CatchBlockStatements	Write the code that is run when an exception occurs
Exception	The exception class that is caught
FinallyBlockStetaments	It is run after Try block or Catch block are executed. It is run regardless of whether an exception occurs or not

Try ... Catch Statements Example

- Enclose the parts where exception can occur with Try...Catch, and describe the process to be run when an exception occurs in Catch block

```
Sub Main()
    Dim a As Integer
    Dim b As Integer = 0
    a = 10\b
End Sub
```

Parts where exception can occur

```
Sub Main()
    Dim a As Integer
    Dim b As Integer = 0
    Try
        a = 10\b
    Catch ex As DivideByZeroException
        Console.WriteLine("Can't divide by zero")
    End Try
End Sub
```

Enclose with Try...Catch

Main Properties of Exception

Property	Description
Message	Message that shows the cause
Source	Assembly name of the exception occurrence source
StackTrace	Stack Trace till exception occurrence
TargetSite	Method information of the exception occurrence source (MethodBase class)
HelpLink	URL link for help information
InnerException	When exception is nested, it is the Instance of the exception type that is thrown just before

Exception Property Example

Output the exception properties information, that is specified in Catch block

```
Try
   a  =  Int32.Parse("abcde")
Catch  ex  As  Exception
   Console.WriteLine(ex.Message)
   Console.WriteLine(ex.StackTrace)
End Try
```

Output Exception information

<OUTPUT>

```
Input string was not in a correct format.
  at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
  at System.Int32.Parse(String s)
  at ExceptionSample2.Module1.Main() in C:\ExceptionSample2\Module1.vb:line 7
```

Filtering

Catch block through its multiple descriptions can catch specific exception

The exception class specified in the Catch statement, or its sub class are processed

```
Sub Main(ByVal Arg() As String)
    Try
        Console.WriteLine(CType(Arg(0), Integer) \ CType(Arg(1), Integer))
    Catch ex As ArithmeticException
        Console.WriteLine("ArithmeticException occurred")
    Catch ex As Exception
        Console.WriteLine("Exception occurred")
    End Try
End Sub
```

<<OUTPUT>>

```
> filter.exe  5  0
ArithmeticException occurred
> filter.exe  5  abc
Exception occurred
```

Filtering note

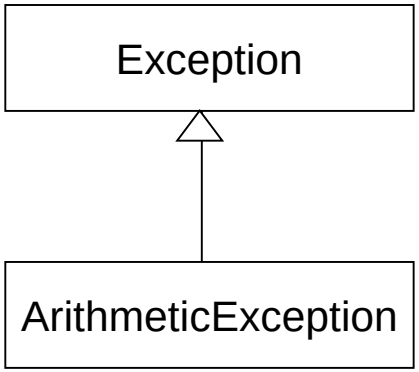
When specifying multiple Catch blocks, it is necessary to start specifying the hierarchical structure from the lower levels of the Exception class

A correct usage example

```
Try
    ••••••
Catch  ex  As ArithmeticException   } (1)
    ••••••
Catch  ex  As  Exception            } (2)
    ••••••
```

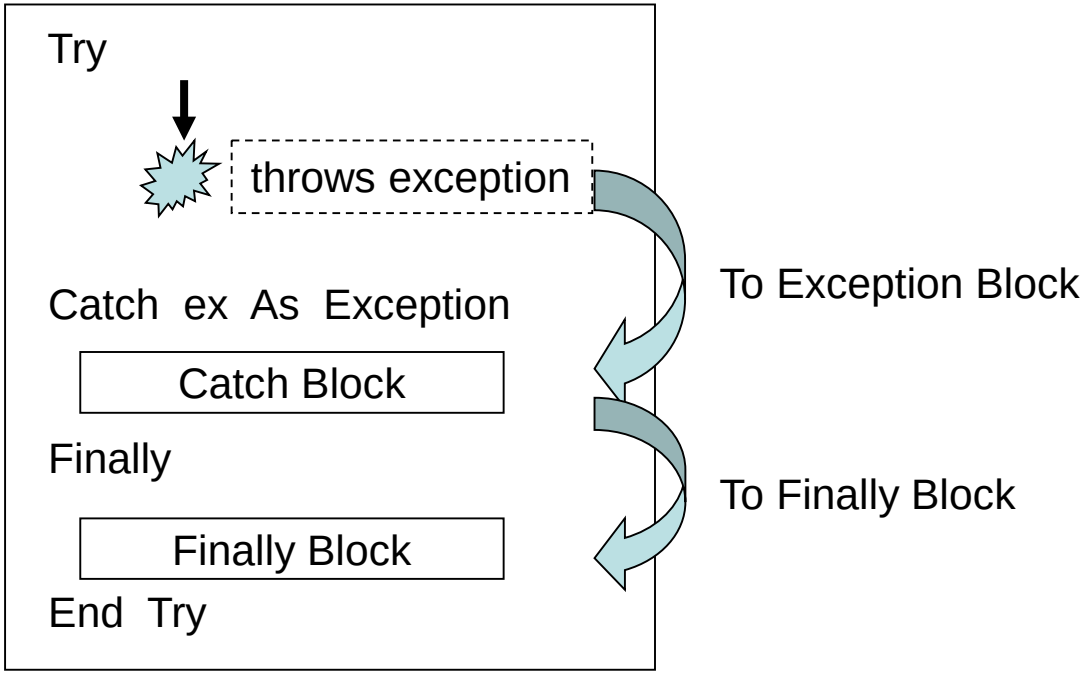
An incorrect usage example

```
Try
    ••••••
Catch  ex  As Exception             } (3)
    ••••••
Catch  ex  As  ArithmeticException  } (4)
    ••••••
```



Finally Block

- When Try statement is processed, Finally block is definitely run regardless of whether exceptions have occurred or not
- Even when an exception occurs, define the process that has to be run
  Example: File Close, Disconnecting DB connection etc.



Throw Exception

- With Throw keyword, an exception can be thrown explicitly
- Even an exception that has been caught can be Thrown again

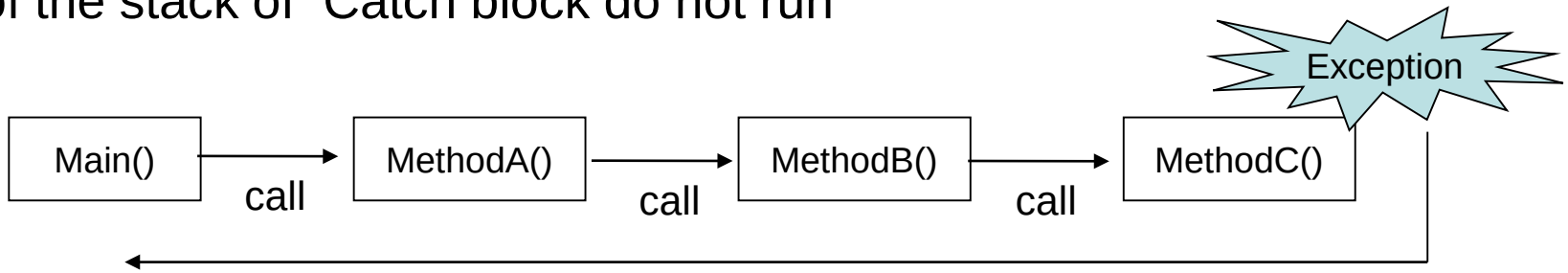
```
 Try
      Throw  New ApplicationException(“Error has occurred!!!”)
Catch  ex  As  Exception
      Console.WriteLine(ex.Message)
      Throw ex
 End  Try
```

<OUTPUT>

```
Error has occurred!!!
```

Catch Block Search

- When an exception occurs, CLR finds the Catch block that has to process the exception
- Search function traces Call stack from the method where exception occurred
- When it is not caught till the end of stack, it is treated as “Exception that cannot be handled”
- When the exception is caught at the lower levels of the stack, the higher levels of the stack of Catch block do not run



When it is not Caught till the end, it is an Unhandled Exception

Trace Call stack till the Catch block that can catch the occurred exception appears

6.3 Debugging Tool

- In VS.NET, various functions have been integrated for debugging
  - Stepping
  - Breakpoint
  - Output Window
  - Quick Watch Dialog Box
  - Watch Window
  - Local Window
  - Exception Catching

  Other than the above, many functions has been provided in VS.NET for the purpose of debugging

Stepping

To debug, select the following items of the [Debug]Menu, and start

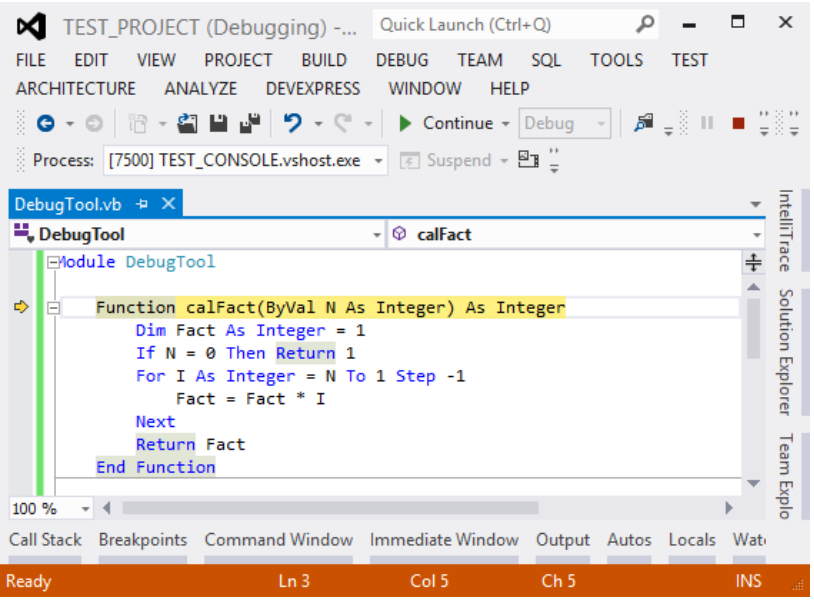
- Start

  Start debugging, and when Breakpoint is set, stop over there

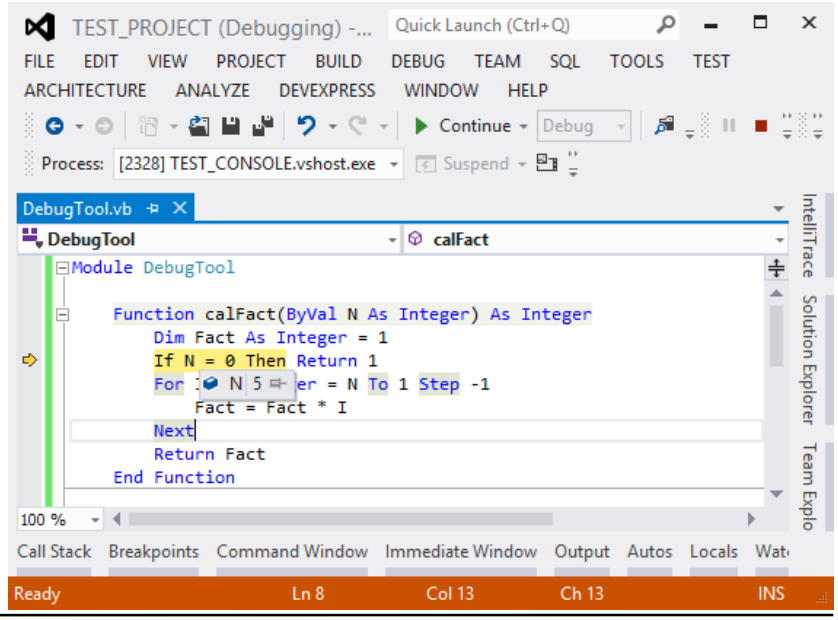
- Step Into, Step Over

  Start program stepping

Select [Debug] – [Step Into]



Step Into

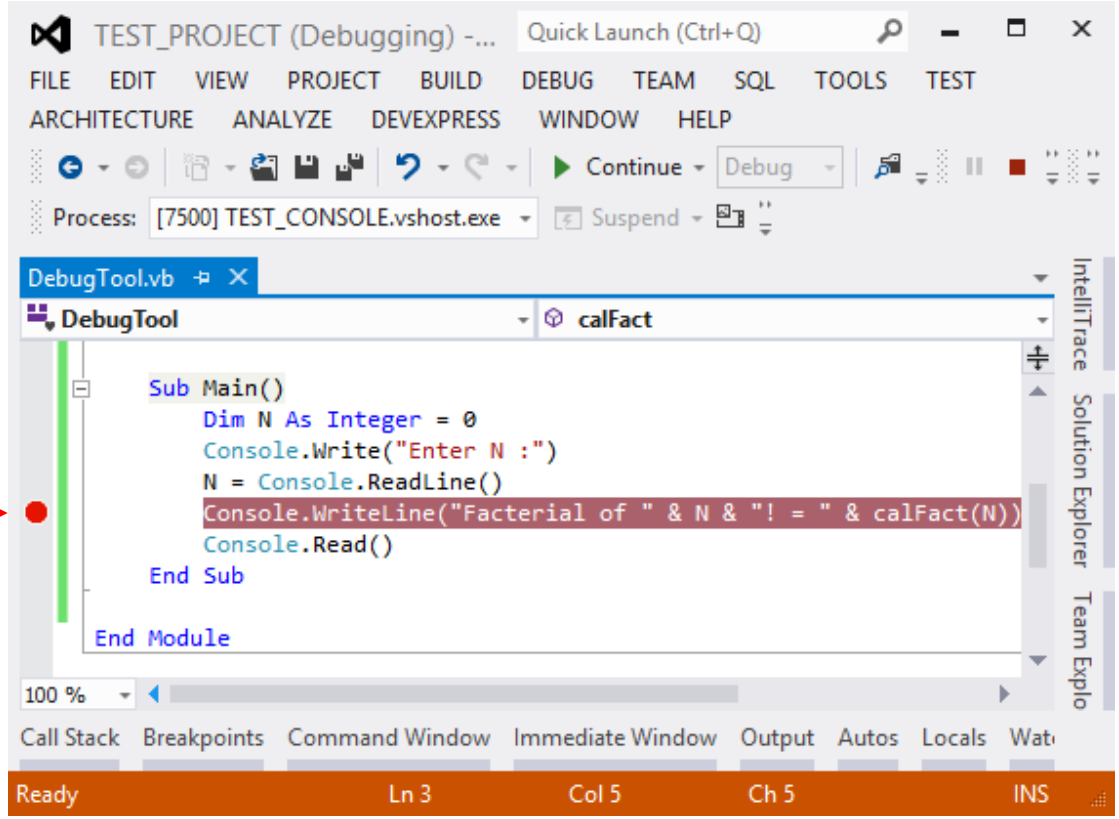


Breakpoint

When Breakpoint is set, it is possible to stop execution at a specific location in the program

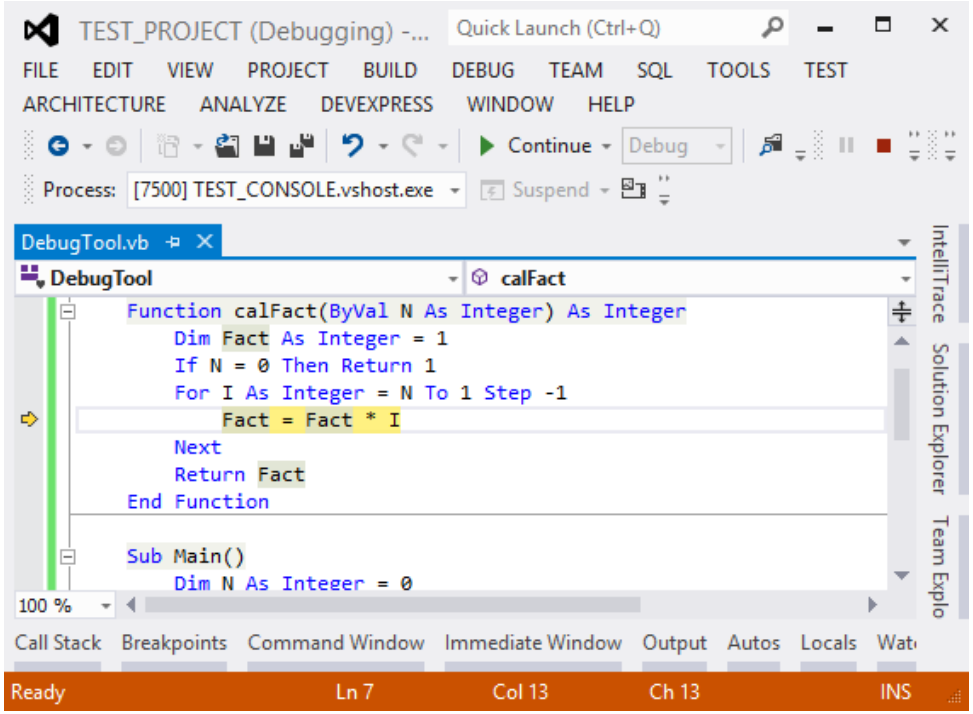
- Breakpoint is set in the source code screen

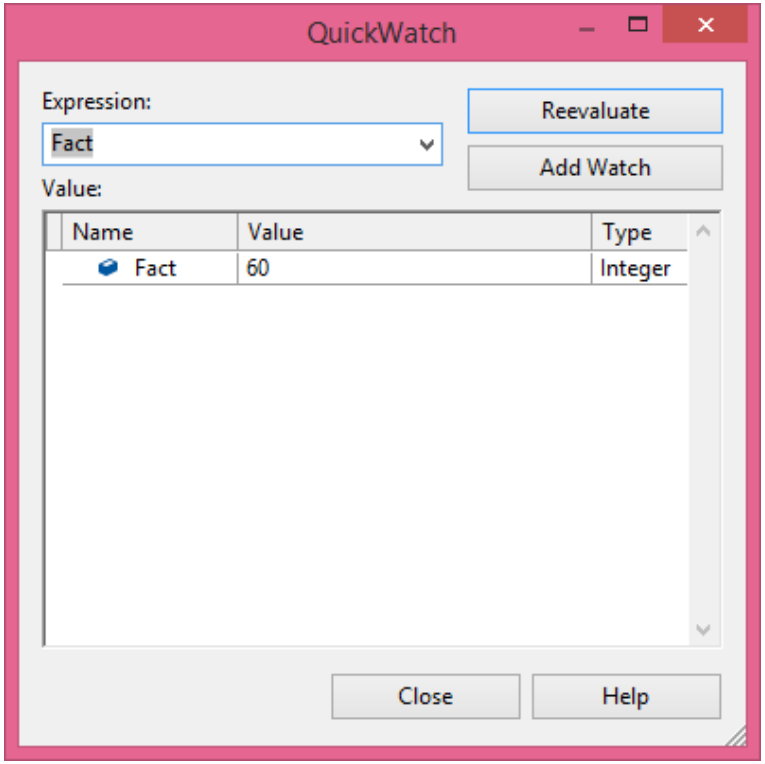
Click



Quick Watch Dialog Box

- The result of variable values and expressions can be verified easily
- It is also possible to edit variable values
- Stop executing at Breakpoint etc., right click the value of the variable or expression to be referred to, and select [Quick Watch]





Watch Window

- Inherit the value of the specified variable or expression, and display
- It is also possible to change the values
- Stop executing at Breakpoint etc., and select [Window] - [Watch] – [Watch1] ~ [Watch4] from the [Debug]Menu

Watch 1

Name	Value	Type
Fact	60	Integer
I	2	Integer
Fact * I	120	Integer

Autos | Locals | Watch 1

Other Windows

- Locals Window
  - Display the values of local variables of the current context
  - Switch context between the Debug Location Tool Bar, or Call Stack Window, Thread Window
- Call Stack Window

  Display the Call stack method name and the type and value of argument