Chapter 2

Language Concepts in VB.NET and C#

Chapter 2
Language Concepts in VB.NET and C#

By the end of this chapter, you should see that the difference between the two languages is very minor, and you shouldn't be afraid to use one or the other when you start developing .NET applications.

a. Variables :

In C# and Visual Basic .NET, you use variables to store information.

VB.NET	*C#*	*.NET Type*	*Bytes*	*Value*
Boolean	bool	System.Boolean	4	True or False.
Byte	byte	System.Byte	1	0 to 255.
Char	char	System.Char	2	0 to 65,535.
Date	DateTime	DateTime	8	January 1, 1 to December 31, 9999.
Decimal	decimal	System.Decimal	12	+/- 79,228,162,514,264,337,593,950,335 with no decimal point. +/- 7.9228162512264337593543950335 with 28 places to the right of the decimal point. The smallest nonzero number would be a 1 in the 28th position to the right of the decimal point.

VB.NET	***C#***	***.NET Type***	***Bytes***	***Value***
Double	double	System.Double	8	-1.797693134862231E308 to -4.94065645841247 for negative values to 4.94065645841247 to 1.797693134862231E308 for positive values.
Integer	int	System.Int32	4	-2,147,483,648 to 2,147,483,648.
Long	long	System.Int64	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
Short	short	System.Int16	2	-32,768 to 32,767.
Object	object	System.Object	4	An object can hold any variable type.
Single	float	System.Single	4	-3.402823E38 to -1.401298E-45 for negative values to 1.401298E-45 to 3.402823E38 for positive values.
String	string	System.String	10+2 Lnth	0 to 2 billion Unicode characters.

1. Declaring Variables in VB.NET and C#

VB.NET	C#
using the *Dim* statement Dim intX as Integer	type precedes the variable. int intx;
Scope variable_name as *DataType*	*Scope DataType variable_name*

The *Scope* of the variable defines where the variable can be accessed.

Note C#

C# is a **case-sensitive** language. If you declare variables named intX, INTX, IntX, and intx, you'll have four separate variables. In VB.NET, they all be considered the same variable because VB.NET isn't a case-sensitive language.

Also in C#, all statements are followed by the **semicolon**. So, a semicolon must follow a variable declaration, and anything after the semicolon is another statement .

✍ you can declare multiple variables in a single statement

VB.NET	C#
Dim intX, intY, intX as Integer	int intX, intY, intX;

Note VB.NET

you can specify **multiple variables** on same line with **different data types**.

Dim intX as Integer, Dim strName as string

✍ You can also set the value of a variable when you declare it:

VB.NET	C#
Dim intX As Integer = 5, intY = 10	int intX = 5, intY = 10;

2. Understanding Variable Scope

✍ you can optionally declare variables with an **access modifier**, which specifies where the variable can be accessed by other elements within your application.

VB.NET	C#
○ **Public**— Access is available to any other class or class member in the same project, another referenced project, or another referenced assembly. ○ **Protected**— Access is limited to the containing class or types derived from the containing class. ○ **Friend**— Access is limited to the current project. ○ **Private**— Access is limited to the procedure, class, or structure. ○ **Protected Friend**— Access is limited to derived classes or the same project.	○ **public**— Access is available to any other class or class member in the same project, another referenced project, or another referenced assembly. ○ **protected**— Access is limited to the containing class or types derived from the containing class. ○ **internal**— Access is limited to the current project. ○ **private**— Access is limited to the containing type.

Note VB.NET
At the class level, using Dim is the same as using Public.

If no access modifier, the member is **private**.

Illustrating Code

VB.NET

```
' Public Class variable accessible to
' any object that derives from this class
Public Class Class1

   ' Public to this class and derived classes
   Public strName As String

   ' Private to this class only
   Private intX As Integer

   ' Accessible from any class in project
   Friend intZ As Integer



   ' Private Sub Procedure  available to
   ' Class1, NOT classes derive from Class1
   Private Sub Test()

      ' Private to the procedure Test()
      Dim intX As Integer
   End Sub
   ' Public Sub Procedure available to Class1,
   ' and any classes that derive from Class1
   Public Sub TestPublic()
   End Sub
End Class
```

C#

```
// Public Class variable accessible to
// any object that derives from this class
   public class Class1
   {
     // Public to this class and derived classes
     public string strName;

     // Private to this class only
     private int intX;

     // Accessible from any class in project
     internal int intZ;

     public Class1() { }

    // Private Sub Procedure available to
    //Class1, NOT classes derive from Class1
     private void Test()
     {
       // Private to the procedure Test
       int intX;
     }
     // Public  Procedure available to Class1,
     // and any classes that derive from Class1
     public void TestPublic()
     {       }
}
```

3. Using Constant Variables

A *constant* is an access modifier that enables you to declare a variable whose value doesn't change throughout your application. The best example of a constant is PI.

VB.NET

```
Const myPI As Decimal = 3.4799897
Const name As String= “MOHAMMED EL-DOSUKY”
```

C#

```
const decimal myPI = 3.4799897;
const string name=”MOHAMMED EL-DOSUKY”;
```

4. Using static (in c#) / Shared(in VB.NET) Variables

static / Shared variable's scope is public or private to the class that it's in, but isn't affected by instances of the class.

VB.NET

```
Public Class StaticTest


  Public Shared globalCounter As Integer

  Public Shared Function Increment() As Integer
    globalCounter += 1
    Return globalCounter
  End Function
End Class
```

C#

```
public class StaticTest
{
  public StaticTest ()   {   }
   public static int globalCounter;

  public static int Increment()
  {
    return ++globalCounter;
  }
}
```

Now declare an instance of StaticTest, as this VB.NET code demonstrates:

```
Dim x As New StaticTest()
Dim c As Integer = x.Increment()
MessageBox.Show(c.ToString)

Dim y As New StaticTest()
c = y.Increment()
MessageBox.Show(c.ToString)
```

Each separate instance of the StaticTest class retains the value of the globalCounter variable. It's shared across all instances of the class.

5. Using the New (in VB.NET) / new (in c#) Keyword

- used for creating new object instances and setting their members to default values
- you must normally create an instance of a class before you attempt to access any of its members.

VB.NET	C#
Dim x as Class1 = new Class1 ' also the declaration Dim x as new Class1	Class1 x = new Class1();

- After you've declared X as a new instance of the class Class1, you have access to the members of Class1 using the dot notation.

VB.NET	C#

' Set the variable
x.firstName = "El-Dosuky"
 	
' Retrieve the variable
MessageBox.Show(x.firstName)
	// Set the variable
	x.firstName = " El-Dosuky ";
	
	// Retrieve the variable
	MessageBox.Show(x.firstName);

b. Operators : arithmetic, assignment, bitwise, comparison, concatenation, logical.

1.Arithmetic Operators

VB.NET	***C#***	***Description***
+ - * /	+ - * /	Addition, Subtraction, Multiplication, Division
Mod	%	Modulo
^ \	N/A	Exponentiation, Integer division
N/A	++ --	Unary addition , subtraction

C#

```
private void testOperators()
{
  int X = 50;
  int Y = 10;

  // Addition - returns 60
  MessageBox.Show((X + Y).ToString());

  // Subtraction - returns 40
  MessageBox.Show((X - Y).ToString());

  // Multiplication - returns 500
  MessageBox.Show((X * Y).ToString());

  // Division - returns 5
  MessageBox.Show((X / Y).ToString());

  // Modulo - returns 0
  MessageBox.Show((X % Y).ToString());

}
```

VB.NET

```
Private Sub testOperators()
  Dim X As Integer = 50
  Dim Y As Integer = 10

  ' Addition - returns 60
  MessageBox.Show((X + Y).ToString())

  ' Subtraction - returns 40
  MessageBox.Show((X - Y).ToString())

  ' Multiplication - returns 500
  MessageBox.Show((X * Y).ToString())

  ' Division - returns 5
  MessageBox.Show((X / Y).ToString())

  ' Modulo - returns 0
  MessageBox.Show((X Mod Y).ToString())

  ' Modulo - returns 10
  MessageBox.Show((Y Mod X).ToString())

  ' Exponent - returns 9.76562E+16
  MessageBox.Show((X ^ Y).ToString())
End Sub
```

2. Assignment Operators

take a value from the right side and assign it to the value on the left.

VB.NET	***C#***	***Description***
=	=	Equals assignment
+=	+=	Addition/concatenation assignment
-=	-=	Subtraction assignment
*=	*=	Multiplication assignment

VB.NET	*C#*	*Description*
/=	/=	Division assignment
\=	/=	Integer division
^=	N/A	Exponentiation assignment
&=	+=	String concatenation assignment
N/A	%=	Modulus assignment
N/A	<<=	Left shift assignment
N/A	>>=	Right shift assignment
N/A	&=	Bitwise AND assignment
N/A	^=	Bitwise exclusive OR assignment
N/A	|=	Bitwise inclusive OR assignment

C#

```
private void testOperators()
{
  int X = 50;
  int Y = 10;

  MessageBox.Show((X += Y).ToString()); // returns 60

  MessageBox.Show((X -= Y).ToString());// returns 50

  MessageBox.Show((X *= Y).ToString());//returns 500

    MessageBox.Show((X /= Y).ToString());// returns 50

   MessageBox.Show((X = Y).ToString());// returns 10
}
```

VB.NET

```
Private Sub testOperators()
  Dim X As Integer = 50
  Dim Y As Integer = 10
  X += Y  :   MessageBox.Show(X)   '  returns 60
  X -= Y  :   MessageBox.Show(X)   '  returns 50
  X *= Y  :   MessageBox.Show(X)   '  returns 500
  X /= Y  :   MessageBox.Show(X)    ' returns 50
  MessageBox.Show((X = Y).ToString()) ' returns FALSE
End Sub
```

3. Comparison Operators

VB.NET	***C#***	***Description***
<	<	Less than
<=	<=	Less than or equal to
>	>	Greater than
>=	>=	Greater than or equal to
=	==	Equal to
<>	!=	Not equal to
Is	==	Compare two objects
TypeOf	is	Compare object reference types
=	==	Compare two strings
&	+	Concatenate Strings
AndAlso	&&	Short-circuited Boolean AND
OrElse	||	Short-circuited Boolean OR
And	&&	Logical AND
Or	||	Logical OR
Not	!	Logical NOT

C#

```
private void testOperators()
{
  int X = 50;
  int Y = 10;
  if (X < Y)
  {
   //This never shows, X is NOT less then Y
    MessageBox.Show("X greater than Y");
  }

  string s1 = "Dosuky";
  string s2 = " dosuky";
  if (s1 == s2)
  {
    // This never shows, Dosuky ≠ dosuky
    MessageBox.Show("s1 = s2");
  }
  if (s1 == "Dosuky" || s2 == "dosuky")
  {
    // This shows, s1 or s2 = Dosuky
    MessageBox.Show("s1 or s2 = Dosuky");
  }
}
```

c. Decision Structures

Decision structures are program elements that control the flow of your application.

Type	***VB.NET***	***C#***
Selection	Select Case	switch
Decision	If...Then	if...else
Looping (condition-	While	do

Type	*VB.NET*	*C#*
checked-first) Looping (condition-checked-last)	Do , Do Until Loop While , Loop Until	while
Looping structure for collections	For For Each	for foreach

1. Select Case (in VB.NET) and switch in (c#)

in C#, syntax of switch:

```
switch (expression)
{
  case constant-expression:
    statement
    jump-statement
  [default:
    statement
    jump-statement]
}
```

Valid **jump statements** in C#

- **break**— Terminates the closest loop or conditional statement in which it appears.
- **continue**— Passes control to the next **iteration** of the enclosing iteration statement.
- **default**— If the expression being evaluated in a switch statement doesn't match any of the corresponding case constant expressions, the default label is used.
- **goto**— Transfers control to a label within the looping or switch statement.
- **return**— Terminates execution of the executing method and passes control back to the caller.

```
string strName = "El-Dosuky";
switch(strName)
{
  case "Merna":
  case "Mona":
    // ….
    break;
  case "El-Dosuky":
   //……
    break;
  default:
    // ….
    break;
}
```

In VB.NET , syntax of Select Case is very complex format.
Here we give an example :

```
Dim intX As Integer = 50
Select Case intX
  Case Is > 51
    ' ….
  Case 43 Or 57 Or 98
    ' ….
  Case 5 to 9
    ' ….
  Case Else
End Select
```

For valid jump statements in VB.NET, say **Exit X**
Where x is Sub| Function| For| ….
You can also use **return** or **goto**.

2. If...Then Statements

VB.NET	C#

If *condition* [Then]
  [*statements*]
[ElseIf *elseifcondition* [Then]
  [*elseifstatements*]]
[Else
  [*elsestatements*]]
End If
	if (*expression*)
	  statement1
	[else
	  statement2]

VB.NET

```
Dim intX As Integer = 10
If intX > 11 Then
  MessageBox.Show("intX > 11")
ElseIf intX < 5 Then
  MessageBox.Show("intX < 5")
Else
  MessageBox.Show("Fallback code block")
End If
```

C#

```
int intX = 10;
if (intX > 11)
  MessageBox.Show("intX > 11");
else if (intX < 5)
  MessageBox.Show("intX < 5");
else
  MessageBox.Show("Fallback code block");
```

3. Looping Structures

While (in VB.NET) and *while*(in C#) → execute while a condition is true. *Do...Loop* (VB.NET) and *do* (C#) → execute until a condition is satisfied. *For...Next* (VB.NET) and *for* (C#) → execute a specified number of times. *For...Each* (VB.NET) and *foreach* (C#)→ execute for each item in a collection.

Illustrating Code

VB.NET	C#
' Use a while loop	**// Use a while loop**
Dim intX As Integer = 1	int intX = 1;
While intX < 6	while (intX < 6)
intX += 1	{
End While	intX++;
	}
' Use a do loop until	**// Use a do loop until**
Dim intX, intY As Integer	int intX, intY;
Do	do {
intX = intY + 1	intX = intY++;
Loop While intY < 10	} while(intY < 10);
' Use a for loop	**// Use a for loop**
Dim intX As Integer	for (int intX = 1; intX <= 5; intX++)
For intX = 1 To 5	Console.WriteLine(intX);
Console.WriteLine(intX)	
Next intX	

C# hint for(; ;){} = while(true){} = infinite loop.

VB.NET hint for I = 0 to 10 step 2 : … : next

4.Some useful instructions (IIf in VB.NET or ?: in c#)

VB.NET IIF	Conditional assignment C# ?:
The *conditional operator (?:)* is related closely to the if/else structure. It takes three operands. The operands and the ?: form a *conditional expression.* The first operand is a *condition*, the second is the value for the conditional expression if the condition evaluates to true and the third is the value for the conditional expression if the condition evaluates to false. E.g. , return (deg >= 50) ? "Pass" : "Fail" ;	IIf() is a built-in function that evaluates the first argument, which is a logical expression. If the expression is True, the IIf() function returns the second argument. If the expression is False, the function returns the third argument. Min = IIf(a < b, a, b)

5. Data-Type Conversion Functions

✍ In VB.NET

Function	Converts Its Argument To
CBool	Boolean
CByte	Byte
CChar	Unicode character
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer (4-byte integer, Int32)
CLng	Long (8-byte integer, Int64)
CObj	Object
CShort	Short (2-byte integer, Int16)
CSng	Single
CStr	String
CType	Specific type

For example

```
Dim A As Integer
Dim B As Double
B = CDbl(A)
```

Also in VB.NET, use **Val**() to parse a string and return the first piece forming a double number.

In c#, we use **casting**.

```
int x = (int) y;
// Or
Class1 x = y as Class1;
```

But both c# and VB.NET make use of the class **Convert**. Try use it . Also, both make use of **int.parse() , double.parse**(), …. Try use them.

Debuging

Debug class is found in the **System.Diagnostics** namespace.
Debug is used to determine the runtime state of your code .
Using **Debug.Writeline**, the data is sent to the **Output** window by default. You can add a listener and direct the output to an event log or a text file. Code written using **Debug** shows up in the build only if current build configuration is **Debug**. Not in **Release**.

Properties of the Debug Class

Property	***Description***
AutoFlush	Gets/sets a value indicating if Flush should be called after every write
IndentLevel	Gets or sets the indent level
IndentSize	Gets or sets the number of spaces in an indent
Listeners	Gets the collection of listeners that is monitoring the debug output

Methods of the Debug Class

Method	***Description***
Flush	Flushes output buffer and causes buffered data to write to the Listeners collection
Indent	Increases the current IndentLevel by one
Unindent	Decreases the current IndentLevel by one
Write	Writes information about the debug to the trace listeners in the Listeners collection
WriteIf	Writes information to trace listeners in the Listeners collection if a condition is true
WriteLine	Writes information and adds a linefeed character to the end of the output
WriteLineIf	Writes information if a condition is true , adds a linefeed character at end of output

```
private void button3_Click(object sender, System.EventArgs e)  {
int x = 5;
System.IO.Stream fOut = System.IO.File.Create(@"C:\csConsoleOut.txt");

System.Diagnostics.TextWriterTraceListener textListener = new
      System.Diagnostics.TextWriterTraceListener(fOut);

System.Diagnostics.Debug.Listeners.Add(textListener);

System.Diagnostics.Debug.WriteLine("This is going to a text file");
System.Diagnostics.Debug.Indent();
System.Diagnostics.Debug.WriteLineIf(x == 7, "X = 7");
System.Diagnostics.Debug.WriteLineIf(x == 5, "X = 5");

textListener.Flush();
}
```

When you run the project, the default Output window displays the following:

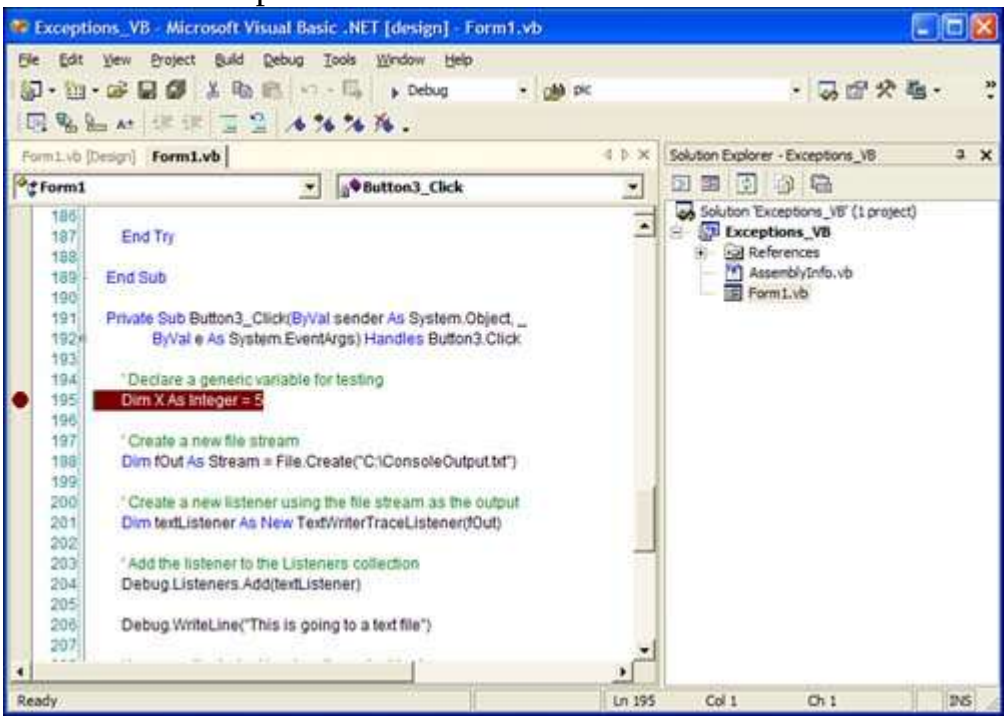
This is going to a text file

X = 5

Using Breakpoints to Debug Code

Breakpoints can be set in the following five ways:

- click on the margin to the left of the line where the breakpoint should be set
- Place the cursor on the line where the breakpoint should occur and press F9
- Select New Breakpoint from the Debug menu
- Right-click the line of code and select New Breakpoint or Insert Breakpoint .
- Place the cursor on the line where the breakpoint should occur and press Ctrl+B .



To remove breakpoints, you can

- click on the margin to the left of the line where the breakpoint is currently set
- Place the cursor on the line where the breakpoint occurs and press F9
- Right-click the line of code and click Remove Breakpoint from the context menu
- Select Clear All Breakpoints from the Debug menu

To disable/enable breakpoints, you can

- Right-click the line of code where the breakpoint should be disabled (enabled) and select Disable (Enable) Breakpoint from the contextual menu
- Select Disable (Enable) All Breakpoints from the Debug menu

If you haven't set a breakpoint, but still want to enter break mode, you can

- Press Ctrl+Break at any time while the application is running
- Select Break All from the Debug menu

Using the Command Window

The command window has two modes: command mode (default) and immediate. With command window you can evaluate an expression, execute a statement, or print any variable's value in the current scope. To change between command mode an immediate mode, type **cmd** or **immed** at the command prompt and press the Enter key.

	Caption	***Command Prompt***
Command Mode	Command Window	>
Immediate Mode	Command Window - Immediate	None

write varName or expr then press Enter

Using the This/Me Window

The This/Me window enables you to view and modify data members of the object.

The window is named This in C# and Me in Visual Basic.

By double-clicking in the value cell of the data member, you can edit the current value of that member and it's changed for the existing execution scope.

Using the Autos Window

The Autos window is similar to the This/Me window .

The difference is that the Autos window displays variables used in the current statement and the previous statement for C++ and C#, or the current statement and the three statements on either side of the current statement for Visual Basic.

Stepping Through Code

	Keyboard Layouts	***Shortcut***	***Debug Menu***	***Context Menu***
Continue	F5		Continue	
Break All	Ctrl + Alt + Break		Break All	
Stop Debugging	Shift+F5		Stop Debugging	
Restart	Ctrl+Shift+F5		Restart	
Detach All			Detach All	
Step Into	F11		Step Into	
Step Over	F10		Step Over	
Step Out	Shift+F11		Step Out	
Run to Cursor				Run to Cursor