Unit 3 Mastering VB Language

Structure:

3.0 Objectives

After studying this unit, you will be able to:

- list and explain the data types support by VB .NET
- list and explain the operators
- write a program using decision making and looping structure
- discuss the error handling techniques in VB .NET

3.1 Introduction

In the previous unit we discussed the concept of Visual Studio .NET and Microsoft Development Environment. Integrated design environment and integrated debugging environment that helps the program developers to develop the application in a convenient way. Some basic concepts and basic skills required to work with the Development Environment. We also discussed to develop a simple application in VB .NET environment and to run the application.

In this unit we are going to discuss the data types, operators, decision making, looping statements, error handling, techniques supported by VB .NET platform. Also we are introducing the object oriented programming concepts in discussing on classes and objects

3.2 Data Types

A variable has a major role in programming languages. Any holder or

container with a name to hold some value is called variable. These variables are the basic fundamental need for any data processing. Basically the string or the text variables are simple. Whereas the numerical type comes with the different size and precision in order to improve the performance of an application. An earlier version of VB .NET supported the default variable as variant. If the programmer is not mentioning about the type of variable while declaration, it will be considered as variant. This variant type holds any types of data like number or string. It reduces the burden of the programmer to some extend to decide later. .NET does not support the variant type because it requires confirmation between the other languages.

The variant type, Efficient though it often was, had two fatal flaws from the perspective of those who designed VB.NET. First, in some cases, VB had a hard time figuring out which type the variant should change to resulting in an error. Second, the other languages in the .NET universe do not use variants and the .NET philosophy requires conformity between its various languages (at least on the fundamental issues, such as variable typing). Therefore, the variant variable is no longer part of the VB language. It has been banished in VB.NET.

X = 10 and y = 10.3

Here when the value 10 assigned to the variable x, VB understands that this is the integer type. But the y value has the fractional part insist to change the type to floating type so here casting is happening.

```
x = "11"
y = 10
z = x + y
MsgBox (z)
```

From this example you will get the answer as 21, but before doing this calculation the casting process is require converting from string to integer. This interpretation process delays the processing or execution time of an application. Thus the VB .NET is not entertaining the usage of variant type. The easiest and simplest data type is Boolean. It can represent two states that is, True and False. The default value of the Boolean is False. This type can be used where you want to toggle between states of True and False or say On and Off. The syntax for the Boolean data type is:

Dim bool1 As Boolean

Another simple data type is the Integer and its larger size, the Long type. Before VB.NET, the Integer data type was 16 bits large and the Long data type was 32 bits large. Now these types are twice as big as they used to be: Integer is 32 bits large and Long is 64 bits large. If your program needs to use a 16-bit integer, use the new type Short. So if you're translating pre-

.NET VB code, you need to change any As Integer or Cint commands to As Short and Cshort, respectively. Similarly, As Long and CLng now must be changed to As Integer and Cint. In most programming, the Integer is the most common numeric data type. If your non-fractional number is larger or smaller than an integer can hold, make it a Long data type.

Dim abc As Integer
Dim xyz As Long

The other major numeric type is called floating point. It has similar small and large versions called Single and Double, respectively. Use it when your program requires the precision of using fractions:

Dim fract1 As Single, fract2 As Double

VB.NET also has a new Char type, which is an unsigned 16-bit type that is used to store Unicode characters. The new Decimal type is a 96-bit signed integer scaled by a variable power of 10. Table 3.1 listing the data types available ib VB .NET.

Table 3.1: Data types in VB .NET

Visual Basic Type	Common Language Runtime Type Structure	Storage Size	Value Range
Boolean	System. Boolean	2 bytes	True or False
Byte	System. Byte	1 byte	Unsigned 0 to 255
Char	System. Char	2 bytes	Unsigned 0 to 65535
Date	System. Date Time	8 bytes	January 1, 0001 to December 31, 9999
Decimal	System. Decimal	16 bytes	+/- 79,228,162,514,264,337,593,543,950,335 with no decimal point; +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Double (double-precision floating-point)	System. Double	8 bytes	-1.79769313486231E+308 to 4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486231E+308 for positive Values
Integer	System.Int32	4 bytes	-2,147,483,648 to 2,147,483,647
Long (long integer)	System.Int64	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Object	System. Object (class)	4 bytes	Any type can be stored in a variable of type Object
Short	System.Int16	2 bytes	-32,768 to 32,767
Single (single-precision floating-point)	System. Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E+38 for positive values
String (variable-length)	System. String (class)	Depends on platform	0 to approximately 2 billion Unicode Characters
User-Defined Type (structure)	(inherits from System. Value Type)	Sum of the sizes of its members	Each member of the structure has a range determined by its data type and independent of the ranges of the other Members

3.3 Operators

Operators are nothing but the symbols which insist the compiler to execute or perform specific logical or mathematical operations. VB .NET has huge number of operators that support to do arithmetic and logical operations. We are now going to discuss the most commonly used operators.

- Arithmetic operators
- Comparison operators
- Logical/Bitwise operators
- Assignment operators

Arithmetic operator

It is a mathematical operation calculated between any two operands. These operators can be used in expressions to perform sequence of calculations. Following table 3.2 shows the various arithmetic operations supported by VB .NET, you can assume here the variable A holds the value 2 and the variable B holds the value 7.

Table 3.2: arithmetic operators

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

Comparison Operator

These operators compare the relationship between given string or numbers and have the value one if the condition is true otherwise 0 for false. String will be generally compared by taking the character one by one from each of the string. The table 3.3 is listing of comparison operators supported by VB.NET language.

Table 3.3: Comparison operator

Operator	Description	Example
==	Two operands values are verified for its equality. If the condition is yes then the result is true.	(A == B) is not true.
<>	Two operands values are verified for its non-equality. If the condition is yes then the result is True	(A <> B) is true.
>	Verify whether the value of left operand is greater than the value of right operand, if yes then the result is true.	(A > B) is not true.
<	Verify whether the value of left operand is less than the value of right operand, if yes then the result is true.	(A < B) is true.
>=	Verify whether the value of left operand is greater than or equal to the value of right operand, if yes then the result is true.	(A >= B) is not true.
<=	Verify whether the value of left operand is lesser than or equal to the value of right operand, if yes then the result is true.	(A <= B) is true.

Apart from these regular operators that are supported by almost all the languages following are the operators supported by VB .NET.

IS operator: Compares two object reference variables and determines if two object references refer to the same object without performing value comparisons. If object1 and object2 both refer to the exact same object instance, result is True; otherwise, result is False.

Is Not Operator – Also compares two object reference variables and determines if two object references refer to different objects. If object1 and object2 both refer to the exact same object instance, result is False; otherwise, result is True.

Like Operator – This operator matches a string against a pattern.

Logical and Bitwise operators

Below are the listed logical operators supported by VB .Net here we need to assume that the variable A has the Boolean value as True and the B has the

value as False. Table 3.4, listing the logical and bitwise operators.

Table 3.4: logical and bitwise operators

Operator	Description	Example
And	It is one of the logical and bitwise AND operator. Here If both the operands are true then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A And B) is False.
Or	It is the logical as well as bitwise OR operator. If any of the two operands is true then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	Not (A And B) is True.
Xor	It is the logical as well as bitwise Logical Exclusive OR operator. It returns True if both expressions are True or both expressions are False; otherwise it returns False. This operator does not perform short-circuiting, it always evaluates both expressions and there is no short-circuiting counterpart of this Operator	A Xor B is True.
And Also	It is the logical AND operator. It works only on Boolean data. It performs short-circuiting.	(A AndAlso B) is False.
Or Else	It is the logical OR operator. It works only on Boolean data. It performs short-circuiting.	(A OrElse B) is True.
Is False	It determines whether given expression is False.	
Is True	It determines whether given expression is True.	

Bit Shift operator

Bitwise operations will be executed on the bits to perform bit operation, truth table for &, | and ^ are listed below.

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

Assignment Operator

Table 3.5 listing the assignment operators available in VB.NET

Table 3.5: Assignment operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side Operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left Operand	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left Operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand(floating point division)	C /= A is equivalent to C = C / A
\=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand (Integer division)	C \= A is equivalent to C = C \A
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	C^=A is equivalent to C = C ^ A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.	Str1 &= Str2 is same as Str1 = Str1 & Str2

Other Functions supported by VB .NET

- **Address Of**: This function returns the address value of a given procedure.

  Example: Add Handler Button1.click, Addreess Of Button1_click

- **Await:** Until the awaited task completes it suspend the execution of procedure.

  Example:

Dim result as res1 = Await AsyncMethodThatReturnsResult() Await AsyncMethod()

- **Get Type:** Returns the Type object for the given object like methods, events, properties etc.

  Example: MsgBox(GetType(Integer).ToString())

- **Function Expression**: Declares code and parameter of the function that has lambda expression.

```
Dim add5 = Function(num As
Integer) num + 10
Console.WriteLine(add5(10))
```

3.4 Decision Making and Loops Structures

Decision making statements are required for any programming language to support the programmer to decide the block of statements needs to be executed when the condition is true and the other blocks will be executed when it is false. Following are the list of conditional statements supported by VB.NET

- If ….. Then statement
- If……Then…Else statement
- Nested if statements
- Select Case statement
- Nested Select Case statement

Now we are going to discuss the if construct with an example that gives an idea about the if Then and the if Then Else statements

```
Sub Main()
Dim a As Integer = 100
If (a < 20) Then
Console.WriteLine("a is less than 20")
Else
Console.WriteLine("a is not less than 20")
End If
Console.WriteLine("value of a is : {0}", a)
Console.ReadLine()
End Sub
```

In the above example the variable a is declared and assigned value with 100. The if construct checks for the value of a is less than 20 if the condition is True it prints “a is less than 20”. But as per this example, the condition is false so it enters into the else portion and prints “a is not less than 20”. Further it comes out from the construct and prints the original value of a and stops the process.

We will see one more program below that depicts the nested ifconstruct. Sub Main()

```
Dim a As Integer = 100
Dim b As Integer =
200 If (a = 100)
Then
If (b = 200) Then
Console.WriteLine("Value of a is 100 and b is 200")
End If
End If
Console.WriteLine("Exact value of a is
:{0}", a) Console.WriteLine("Exact value of
b is : {0}", b) Console.ReadLine()
End Sub
```

In the above program a and b are the two integer variables declared locally. Here, the if constructs followed by the variable declaration check for the value of a is 100 and the result is a Boolean type. If the result is True then the control will enter in to one more if construct called the nested if. If the
nested if boolean condition results True it prints the statement on the console. If the first construct result or the inner/nested if result is false then the statement come out from the if construct and prints the original value of a and b variables.

Select Case

This construct allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.

Select [Case]
expression [Case
expressionlist

[statements
]] [Case
Else

[elsestatements
]] End Select

expression: is an expression that must evaluate to any of the elementary data type in VB.Net, i.e., Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort.

expression list: List of expression clauses representing match values for expression. Multiple expression clauses are separated by commas.

statements: statements following Case that run if the select expression matches any clause in expression list.

else statements: statements following Case Else that run if the select expression does not match any clause in the expression list of any of the Case statements.

Loop statements

Generally the program starts executing statements sequentially but there are situation where the programmer wanted to execute the set of statements for repeated number of times. So the languages supports by providing various loops construct to support the complex situation. Here loop statements execute either a single or group of statements for a specified number of times. Following are the various loop construct supported by VB .NET.

- Do Loop
- For …. Next
- For Each…Next
- While…..End While
- With….End with
- Nested Loops

Do Loop

This loop executes the single or group of statements until the Boolean condition is True or it becomes True, also the loop can be terminated at any point of time using Exit Do statement. Below are the two construct for Do loop with entry level and exit level condition.

Do { While | Until }
condition Statements

Exit Do
Loop
Do

Statements
Exit Do
Loop { While | Until } condition

Now we will see one example for Do Loop using exit condition. This program starts the loop by printing the variable “a” with the initial values as 10, and continues until the value of a reaches 20. Here the condition checking is at the exit level so first time the loop will be executed and checks for condition to proceed further.

```
Sub Main()
    Dim a As Integer =
```

```
    10 Do
       Console.WriteLine("value of a: {0}",
       a) a = a + 1
    Loop Until (a = 20)
    Console.ReadLine()
  End Sub
```

Output : Value of a:11

-

-

Value of a: 19

For Next

This loop executes for single or group of statements for a specified number of times, here the loop index counts the number of iterations. This loop can be terminated at any point of time using Exit For statement.

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

For Each loop is used exclusively for accessing and manipulating all elements in an array or in VB.Net collection.

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

You can see the example below, For Each loop statement that reads the array value one by one and prints the same. This loop continues to work until it reads entire items from an array.

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

Output: 1, 2, 5, 7, 9

While End While

It is yet another loop executes single or group of statements until the given condition is true. Main key point with while loop is that it starts with the

condition of checking the conditional statement, if the condition results as false the loop statements will be skipped and the statement after the loop body will be executed.

```
While condition [
   statements ]

   [ Exit While ] [
   statements ]
End While
```

Example:

```
        Sub Main()
          Dim a As Integer = 10
          While a < 20
             Console.WriteLine("value of a: {0}",
             a) a = a + 1
          End While
          Console.ReadLine()
        End Sub
      Output: Value of a:10
      -
      -         Value of a:19
```

In the above example the variable a is assigned with the value of 10. While loop starts with the conditional testing of whether the value of a is less than 20. As per this example the condition is True so it starts execute the content of the loop. It prints the value of a and then the value of a is incremented with one. When the value of the variable a reaches twenty, then the condition becomes false and it comes out of the loop and end the process.

With End With

If you take the *With End with* construct it is not exactly a looping construct. It does the execution of series of statements that repeatedly refers to a single structure or object.

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

```
        Public Class student
          Public Property Name As String
          Public Property course As String
          Public Property semester As String
        End Class
        Sub Main()
          Dim stud As New student
          With stud
```

```
        .Name = "Mr.X"
        .course = "MCA"
        .Semester =
    "Five" End With
    With stud
        Console.WriteLine(.Name)
        Console.WriteLine(.course)
        Console.WriteLine(.semester)
    End With
    Console.ReadLine()
End Sub
```

You can observe in the above program group of data is created using the same property and it is been referred by the With construct.

Nested Loops

As we discussed earlier nested are, building of loop construct inside the loop. VB .NET supports this concept with all the loops like Do, While For etc. Below you can find syntaxes pertaining to nested loops.

```
For counter1 [ As datatype1 ] = start1 To end1 [ Step step1 ] For
    counter2 [ As datatype2 ] = start2 To end2 [ Step step2 ]
    ...
    Next [ counter2 ]
Next [ counter 1]
While condition1
    While condition2
    ...
    End While
End While
```

3.5 Error Handling

VB .NET supports two types of error handling through which we can avoid the termination of program during execution. The broad categories are

- Unstructured error handling
- Structured error handling

Generally error happens during the run time also called as exceptions generated due to unexpected or abnormal condition during execution of code.

Unstructured error handling is executed with the help of On Error statement, generally placed at starting of the block of the code in order to handle all the errors which can be raised due to unexpected situation. All VB .NET errors can be handled using Microsoft. Visual Basic. Information. Err namespace. When the procedure is called the handler will be set with

Nothing. Better to have single On Error statement is one procedure more than disable other previous handlers defined in that procedure.

On Error statement is used to either, enable, disable or specify to branch to the location.

On Error {Go To [line | 0 |-1] | Resume Next}

Here Go To line is used to enable the error handling routine, located at the starting of the line, it may be either label or number. When the specified line number or label does not occur in the procedure it raises the compiler error. To avoid the un expected behavior when no error occur we can place Exit Sub, Exit property or Exit Function just near the line number or label.

GoTo 0: Disables the enabled error handler that is defined within the current procedure and resets it to Nothing.

GoTo -1: Disables the enabled exception that is defined within the current procedure and resets it to Nothing.

Resume Next: Moves the control of execution to the statement that follows immediately after the statement that caused the run-time error to occur, and continues the execution from this point forward. This is the preferred form to use to access objects, rather than using the On Error GoTo statement.
Below you can find the situation where and how these error techniques can be handled.

```
'Generate an error if the user cancels.:
dlgOpenFile.CancelError = True
'Ignore errors for now :
On Error Resume Next
'Present the dialog :
dlgOpenFile.ShowOpen
'See if there was an error:
If Err.Number = cdlCancel Then
'The user canceled. Do
nothing: Exit Sub
'Unknown error. Take more action.
ElseIf Err.Number <> 0 Then
  End If
'Resume normal error
handling: On Error GoTo 0
```

Structured error handling helps the programmers to handle the unexpected errors efficiently and provide support to the programmer to develop application efficiently and for easier maintenance for the same. Structures error handling used the Try.. Catch…Finally statement to handle the errors. The code that is suspected for error generation during execution can be put

inside this block. If this block produces an error during execution It tries to match with the error in the catch block. If the matches found, the control will be transferred to the initial line of the catch block. The Finally, exist as a last statement in the Try catch Finally block will be executed immaterial of errors found or not. If there this no error match with the catch block the Finally statement execute to propagate outer statements.

The example gives you an idea about the role of Try…Catch…Finally statements.

```
Public Sub TryExample()
  Dim x1 As Integer = 5
  Dim y1 As Integer =
  0 Try
    X1 /= y1  ' Lead to "Divide by Zero" error.
  Catch ex As Exception When y = 0   ' To Catch error.
    MsgBox(ex.toString)
  Finally
    Beep()  ' sound after processing of error.
  End Try
End Sub
```

3.6 Classes and Objects

Class is defined as “A container for data and code. The data within the class can be accessed with properties. The code is referred to as methods”.

Object is defined as “An instance of a class in memory. An instance is created using a Dim statement and the New keyword”

The definition of class will gives you a blueprint of a data type. Actually whenever you define a class doesn’t mean that you are defining a data. It indicates the content of the object of the class and various operations that can be performed unit this objects. The data members and the functions to perform operations on these data are the members of the class. Objects are nothing but the instance of a class. You can see below the class construct that can be defined in VB .NET language. It starts with the key word called class and followed by the name of the class. Inside the body list of members will be declared and ends with the End Class statement.

```
 [ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable
] [ Partial ] _
Class name [ ( Of typelist ) ]
   [ Inherits classname ]
   [ Implements interfacenames ]
   [ statements ]
End Class
```

attribute list: attributes list of the table all the square brackets in the syntax indicates, it is optional.

Access modifier: Has three types of access specifier like private, protected and public defines the scope of the class member.

Shadows: Declared and hidden as overload member in the name class.

Must Inherit: Indicates you cannot create object for this class only this class can be used for inheritance.
Partial: specify the fractional definition of the class.

Inherits: Indicates which class is inherited.

Implements: used to specify the interface of the class inherited from

In VB .NET environment if you observe, whenever you create a form it comes with Public Class Form1 in right top of the window. The form what you have started itself a class. Form1 can be name of the class. Whenever you add a control to the form, you are adding members in to the form class. When you initiate or start the form VB does the instantiation it is converting in to an object.

3.1 Creating and Running a Simple Application

As we discussed already the IDE panes consists of three areas, project pan provides details about the project which was recently used or opened. Getting started pane gives tips for quick application development. VB Express headlines pane provides the recent updates and releases about the VB versions.

To start with the new application development you need to click on file from the menu bar and select the new project. The new project dialog box will appear as depicted in figure 2.9.

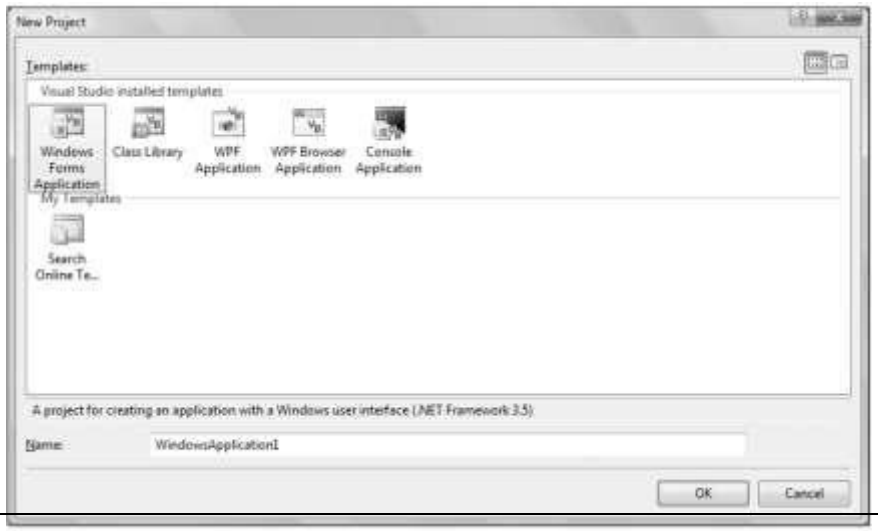

Fig. 2.9: new project dialog window

You can see five templates before you; here we are going to select the Windows Forms application since we are developing windows application. By default you get the name as windowsApplication1 you can also change the name according to your application nature and select ok to continue. As we discussed already the solution explorer will list all the forms and project of the current application. Now you will get the window with the new form as appear in the figure 2.10. Here the application title is specified as My First Program and the project name is given as My project1. Under that you can see a form1 is listed through which we are going to develop our application you can rename this form by changing the name of the form through the property window say “Multiplication”.

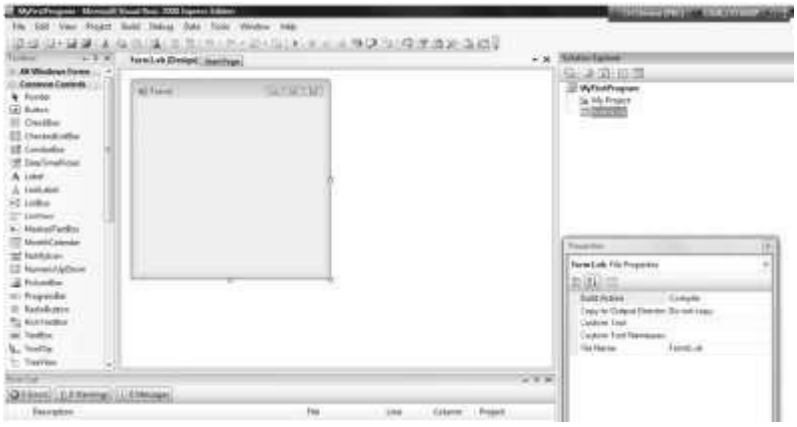

Fig. 2.10: IDE with new form

Now we can design our first application to multiply two numbers when you click on the button which is available on the form. First we will see how to add a button in the form, towards the left hand side in the new form window, the Toolbox window appears under the common control you can see the control called Button. To add this button to the form either you can drag and drop the control to the form or select and double click on the form to put the control in the form change its default name to multiply using the property window. These dragged and dropped controls can be resized later and can located anywhere in the form according to developer wish. For this application we also need three more TextBox controls, the same you can bring these TexBox controls to the form. Two TextBox controls are to accept input and one more to display the calculated result.

Now we will see how to add the coding portion in the application. Here the event decided for calculating or multiplying two numbers are “Click on button”. To go to the code window either we can do double click on the button control or through solution explorer you can reach the code window. If you double click you can see the below procedure appears automatically.

Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As

System.EventArgs) Handles Button1.Click

End Sub

Within this block we need to type the coding or the business logic to do the calculation.

```
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As

System.EventArgs) Handles
        Button1.Click Dim a, b, c as single
        a= TextBox1.Text
        b=TextBox2.Text
        c=a*b
        TextBox3.Text=c

End Sub
```

Running VB .NET application

Once the coding part is over we need to execute or run the program to see the output. We can run the VB. NET application in two ways either by pressing the F5 button or by selecting the run option from the toolbar. For the above specified coding the result will appear as shown in figure 2.11.

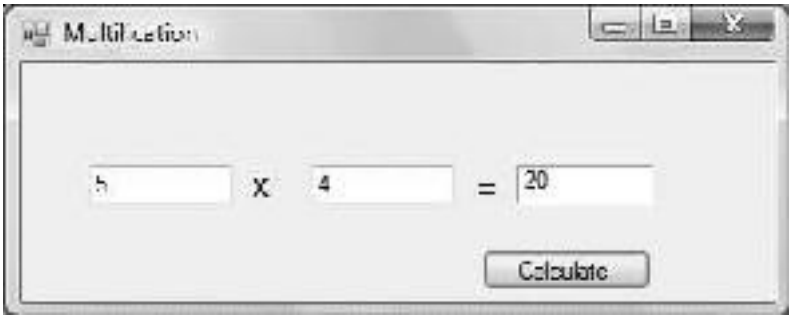

Fig. 2.11: Application output

Now we will see the coding part of this application here three variables are declared as single to accept numbers. The variable “a” is going to get the value which is received by the TextBox during the run time. TextBox1.Text is to take the value from the control will be assigned to the variable “a”. Likewise the variable “b” will receive the value from the control TextBox2. The next statement “c=a*b” will multiply the values of a and b will be stored in c. Finally we are moving the calculated value that is “c” to the TextBox3 control. This application can be executed as we discussed earlier and assume in this example user has given two input as 5 and 4. The resultant value 20 is stored in the TextBox3 and displayed, this happens when you click on the button multiply since the calculation part is written in the button click event.

3.7 Summary

- Each variable in VB .NET has a specific type that defines the size and layout of the variable in memory.
- Among all the data types Boolean data type is considered as a simple one.
- Operators are the symbols which insist the compiler to execute or perform specific logical or mathematical operations.
- AddressOf, Await, GetType and Function Expression are the special data types supported in VB .NET
- VB .NET supports two types of error handling through which we can avoid the termination of program during execution are structured and unstructured error handling techniques.
- Class is defined as “A container for data and code. The data within the class can be accessed with properties. The code is referred to as methods”.
- Object is defined as “An instance of a class in memory. An instance is created using a Dim statement and the New keyword”

3.8 Questions and Exercises

1. List and explain the list of data types supported by VB .NET.
2. Explain the comparison and logical operators available in VB .NET.
3. Discuss the decision making statements with example.
4. List and explain the looping statements.
5. Discuss in detail the error handling techniques.
6. Brief about classes and objects.

3.9 Suggested Readings:

- http://www.tutorialspoint.com/vb.net/vb.net_operators.htm
- http://support.microsoft.com/kb/311326
- http://www.vb-helper.com/err_sample_text.html
- http://msdn.microsoft.com/en-us/library/fk6t46tz(v=vs.71).aspx
- NOTE