Chapter 3 – Performing Operations and Storing the Results

3.1 Variables

Visual Basic .NET allows you to store values in a **variable**.

Creating a variable requires the same specifications as creating an object.

You must allocate the appropriate amount of space and associate a name and data type for it.

To reference the stored value throughout the program you must give variable a valid **variable name**.

You must select from a list of **variable data types** indicating to Visual Basic .NET how much space to allocate and how to process and display the variable.

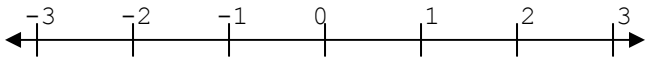
Chapter 3 – Performing Operations and Storing the Results

Variable Data Types

Variables of different data types exist so that you can store different types of values.

Integer is one of the data types available in Visual Studio .NET to represent whole numbers.

Typically you would represent integers are a number following the pattern:



Positive, negative numbers, and the number 0 are all numbers included.

Chapter 3 – Performing Operations and Storing the Results

Selecting the Proper Data Type

Visual Basic .NET provides three data types that store integers.

Which data type is chosen depends on the range of values required by the variable.

Maximum and minimum size of a variable must be taken into account.

Short data type is for values between –32,768 and 32,767.

Integer data type can store a value from –2,147,483,648 to 2,147,483,647.

Long can store a value from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Chapter 3 – Performing Operations and Storing the Results

Selecting the Proper Data Type Continued

You can use the Long data type in all cases to safeguard against choosing the wrong data type.

However, this will waste memory, since Long variable takes twice the space of an Integer variable.

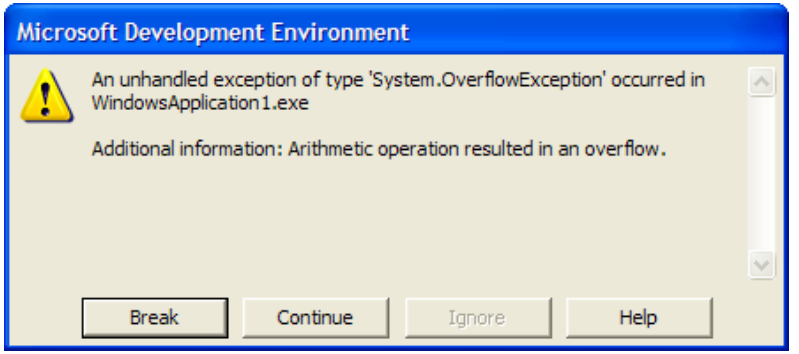
If you choose a variable to be a Short and then set it to a value out of the range of the variable, you will get an execution error (a **run-time error**).

Overflow is the term used to describe when you try to store a value in a variable that is too big.

If you run the following code, you will get an execution error like shown below.

```
Private Sub btnCalculate_Click(…
      Dim shtVariable As Short

      shtVariable = 32767
      shtVariable = shtVariable + 1
End Sub
```



Chapter 3 – Performing Operations and Storing the Results

Other Variable Data Types

Visual Basic .NET provides multiple options for selecting the data type of the variable to use when storing decimal numbers.

Each data type for numerical values has different precisions and storage requirements.

You can select from **Single**, **Double**, or **Decimal** when creating a decimal variable (listed in increasing order of precision and storage requirements).

Visual Basic .NET provides the **String** data type to allow the storage of characters.

You do not need to define how much space is required for Strings because a String's storage requirement is directly related to the length of the string that you wish to store.

A String is specified as a double quote (“), a series of characters, and another double quote.

The **Date** data type allows you to store a date. You need to enclose the date in # signs to use this data type.

Chapter 3 – Performing Operations and Storing the Results

Data Types, Summary

Data Type	Description	Range
`Boolean`	Logical data	True or False
`Date`	Date and time data	January 0100, to December 31, 9999
`Decimal`	Large floating point number	Varies in size depending on the value stored; can hold values much larger or more precise than a Double
`Double`	Large or high precision floating point numbers	Floating point number with up to 14 digits of accuracy
`Integer`	Large integer number	-2,147,483,648 to 2,147,483,647
`Long`	Really large integer numbers	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`Short`	Small integer numbers	-32,768 to 32,767
`Single`	Small floating point numbers	Floating point number with up to 6 digits of accuracy
`String`	Character data	Varies based on the number of characters

Chapter 3 – Performing Operations and Storing the Results

Drill 3.1

In each real-world situation that follows, list the variable data type that would be most appropriate.

1 A variable to store an hourly wage of an employee.

Answer: Decimal

2 A variable to store the average score on an exam that has the lowest possible grade of 0 and highest grade of 100.

Answer: Short

3 A variable to store the sum of 50 test scores on an exam that has the lowest possible grade of 0 and highest grade of 100.

Answer: Short

4 A variable to store the sum of 500 test scores on an exam that has the lowest possible grade of 0 and highest grade of 100.

Answer: Integer

Chapter 3 – Performing Operations and Storing the Results

Drill 3.1 Continued

In each real-world situation that follows, list the variable data type that would be most appropriate.

5 A variable to store the sum of 5,000 test scores on an exam that has the lowest possible grade of 0 and highest grade of 100.

Answer: Integer

6 A variable to store the total number of products ordered. Up to 1 billion orders can be placed, and each order can contain up to three products.

Answer: Long

7 A variable to store the expression “The 76ers are looking great this year!”

Answer: String

8 A variable to store tomorrow’s date.

Answer: Date

Chapter 3 – Performing Operations and Storing the Results

Drill 3.2

If the following code were executed, would an overflow occur? If so, why?

```
Private Sub btnCalculate_Click(...
    Dim shtVariable As Integer

    shtVariable = -32768
    shtVariable = shtVariable + 1
End Sub
```

Answer: An overflow will not occur.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.3

If the following code were executed, would an overflow occur? If so, why?

```
Private Sub btnCalculate_Click(...
    Dim shtVariable As Integer

    shtVariable = 10000
    shtVariable = shtVariable * 3
End Sub
```

Answer: An overflow will not occur.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.4

If the following code were executed, would an overflow occur? If so, why?

```
Private Sub btnCalculate_Click(...
    Dim shtVariable As Integer

    shtVariable = 32767
    shtVariable = shtVariable - 5
    shtVariable = shtVariable + 5
    shtVariable = shtVariable + 1
End Sub
```

Answer: An overflow will occur on the last assignment of shtVariable.

Chapter 3 – Performing Operations and Storing the Results

Variable Names

A variable name in Visual Basic .NET begins with a letter and may be followed by any combination of letters, underscores, or digits. It can be as small as one letter or as large as 255 letters, underscores, and digits combined.

Variable name should be representative of the value that it is storing. More readable variable names will make the program easier to follow.

Visual Basic .NET does not differentiate between two variable names that are identical except for the case of the letters in their names. Therefore, it is not **case sensitive** with regard to variable names.

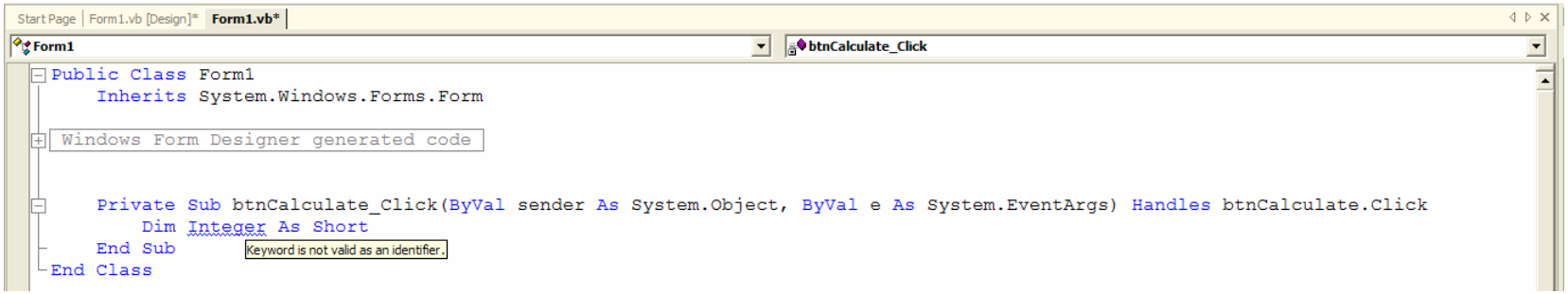
Letters used in variable names can be either lowercase or uppercase. If you refer to the variable with a different capitalization later in the program, Visual Basic .NET will convert the capitalization to the one used earlier in the program.

Chapter 3 – Performing Operations and Storing the Results

Variable Names Continued

Visual Basic .NET will immediately provide feedback if you have violated the rules of declaring a variable: an underline will appear under the violating text.

If you then move the mouse pointer over the underlined text, a pop-up message with an explanation of the error will appear.



Chapter 3 – Performing Operations and Storing the Results

Drill 3.5

Determine which of the following variable names are valid:

1 Maura

2 Ben

3 Maura&Ben

4 Maura_Ben

5 _MauraBen

6 IsThisLegal

7 HowAboutThis?

8 PrivateDancerWasTheNameOfASong

9 Private

Answer: 1, 2, 4, 6 , 8 are valid variable names

Chapter 3 – Performing Operations and Storing the Results

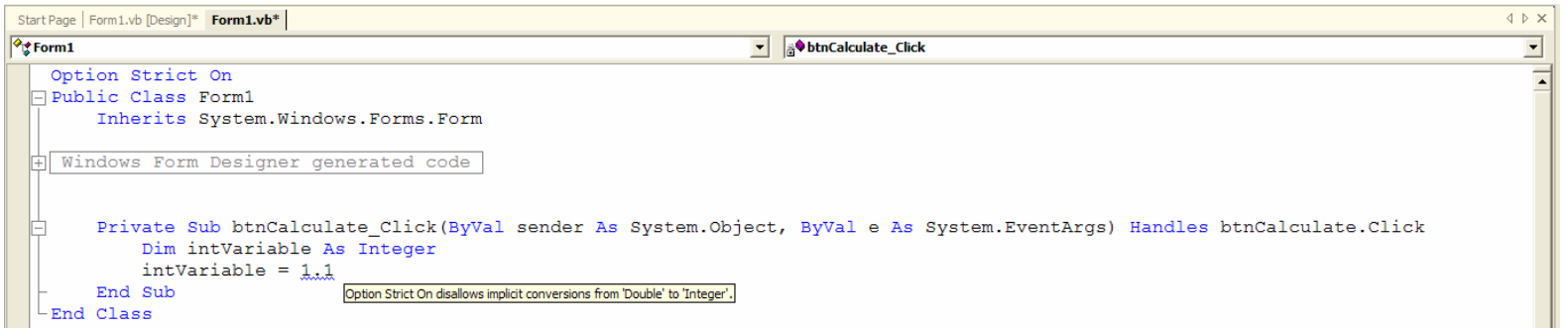
Declaring a Variable

To use a variable in Visual Basic .NET, you must tell the compiler that the variable exists before you actually access it. This is called declaring a variable.

Declaring or **allocating** a variable means that you are indicating to the computer the type of variable that you wish to use as well as the name that you will use to reference it from within the program.

By default, Visual Basic .NET will not allow you to use a variable that you have not declared.

By adding the code `Option Strict On` to the beginning of your module, you can prevent the accidental conversion of one variable data type to another.



Chapter 3 – Performing Operations and Storing the Results

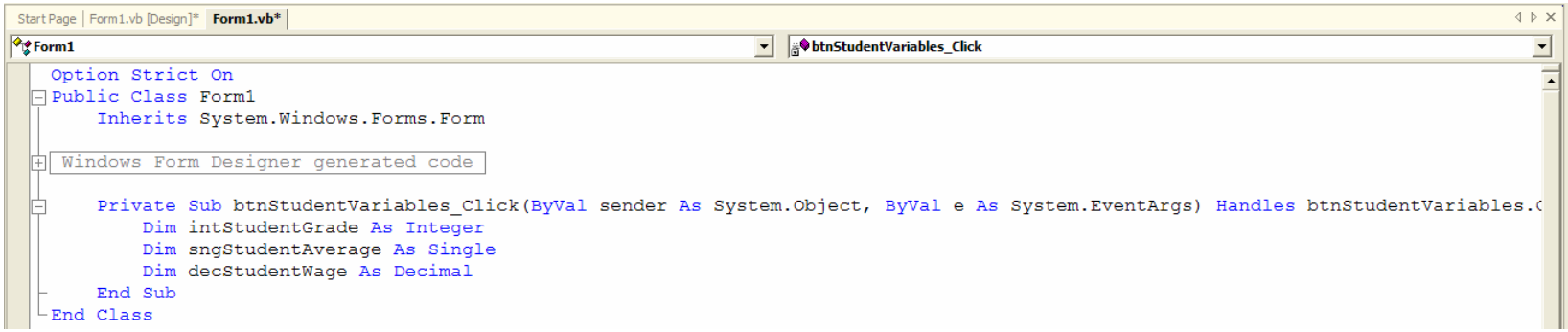
The Dim and Public Keywords

There are two statements you can use to declare a variable.

The degree of visibility that other areas of code can see a variable is known as the **scope** of a variable.

The **Public** keyword is used when you create applications with multiple forms. For now, stick to using the **Dim** keyword when declaring a variable.

You need to first type the word `Dim`, followed by a space, followed by the variable name, followed by the word `As`, followed by the data type of the variable.



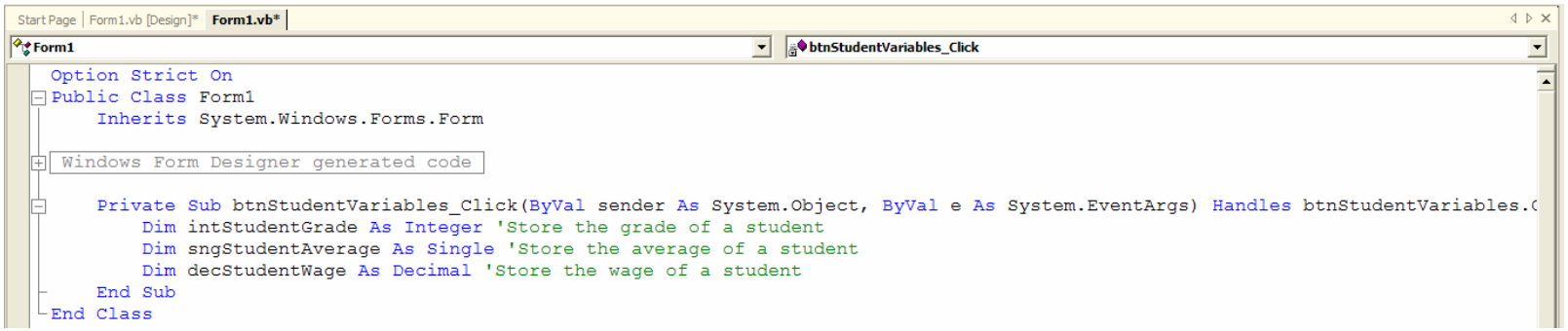
Chapter 3 – Performing Operations and Storing the Results

Adding a Comment

It is always a good idea to add a comment on the same line indicating the purpose of the variable.

You add a comment to a line by typing a single quote and then the comment that you wish to make.

Comments are not part of the actual code but a way of documenting the code so that it is more understandable.



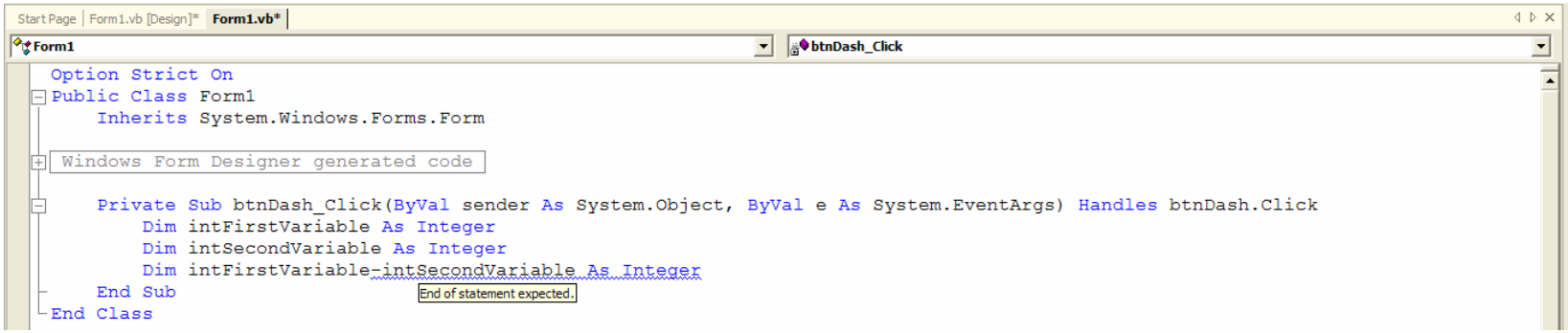
Chapter 3 – Performing Operations and Storing the Results

Ambiguity in Variable Names

It is important when writing computer programs that you do not create ambiguous conditions.

If there is no clear indication of what you intend the computer to accomplish, then the computer will not be able to guess what you intend.

That's why you cannot use a dash in a variable name: there is no way to distinguish between the minus operation and the dash.



Chapter 3 – Performing Operations and Storing the Results

Naming Conventions

It is a good practice to use a naming convention when declaring variables.

All variable names start with a three-letter abbreviation indicating the variable's data type.

After that, a variable should be described in enough detail so that its purpose is self-explanatory.

It is a good standard to capitalize the first letter of each word used in the variable name.

Be consistent with any abbreviation that you might use repeatedly throughout your code.

Data Type	Prefix
Boolean	bln
Date	dte
Decimal	dec
Double	dbl
Integer	int
Long	lng
Short	sht
Single	sng
String	str

Chapter 3 – Performing Operations and Storing the Results

Drill 3.6

What would happen if you tried to write code as follows?

```
Private Sub btnDrill_Click(...
    strDrillValue = "Initial Value"
    Dim strDrillValue As String
    strDrillValue = "What will be the output?"
    MsgBox(strDrillValue)
End Sub
```

Answer: With the default settings, you would have received an error indicating that a reference to `strDrillValue` existed before the declaration of the variable.

Chapter 3 – Performing Operations and Storing the Results

3.2 Simple Operators

Visual Basic .NET allows you to perform all the numerical operations you are familiar with.

A computer uses symbols called **operators** to indicate that an operation is to be performed.

Addition, subtraction, multiplication, and division are supported using operators +, -, *, and / respectively.

The assignment operator is the equals size (=).

The exponent operator ^ is used to raise a number to a power.

The values that operators perform their actions upon are known as **operands**.

Chapter 3 – Performing Operations and Storing the Results

Example of Operator Use

Follow this simple example to see how operators can be used.

Create Form

Step 1: Create a new project.

Step 2: Set the `Name` property of the form to `frmAdd`.

Step 3: Set the `Text` property to `Addition Operator`.

Add Output Label

Step 1: Place a label control in the middle of the form.

Step 2: Set the `Name` property to `lblTotal`.

Step 3: Clear the default text from the `Text` property.

Chapter 3 – Performing Operations and Storing the Results

Example of Operator Use Continued

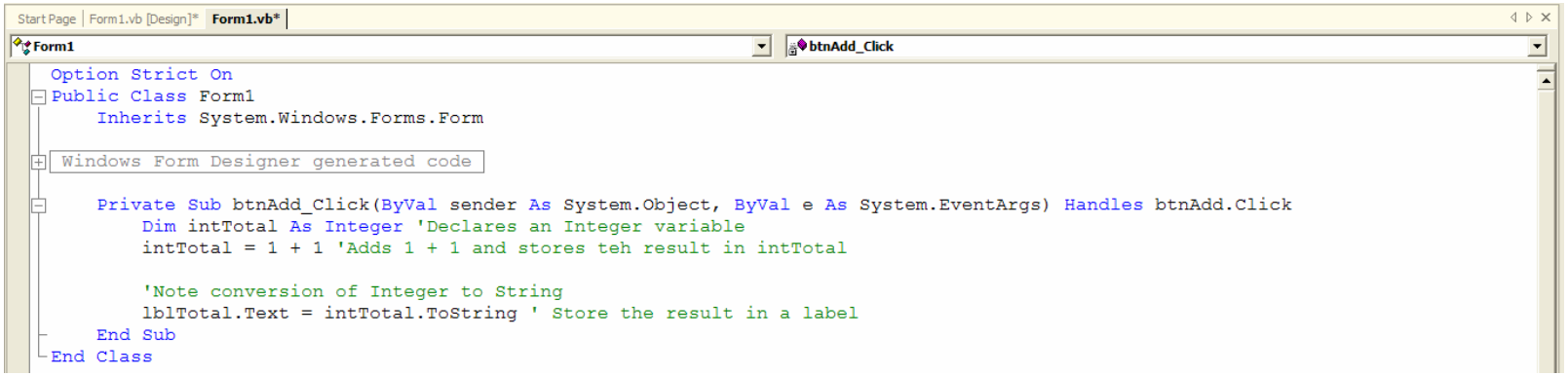
Add Button

Step 1: Place a button control below the `lblTotal` label.

Step 2: Set the `Name` property to `btnAdd`.

Step 3: Set the `Text` property to `Add`.

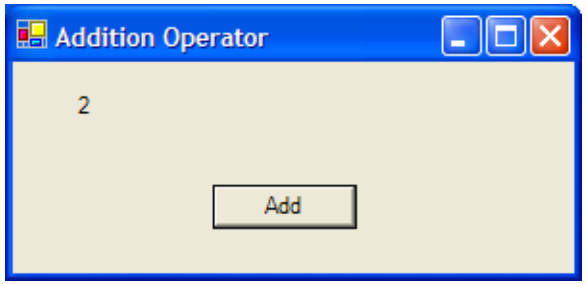
Step 4: Double-click on the `btnAdd` button and add the code shown below.



Chapter 3 – Performing Operations and Storing the Results

Example of Operator Use Continued

The result is shown below.



Chapter 3 – Performing Operations and Storing the Results

Order of Precedence

Expressions may contain not only values and operators but also parentheses.

Parenthesis tell the computer to calculate the operations inside the parentheses before performing the rest of the calculations.

The order in which the operations are performed is referred to as the **order of precedence** of the operations.

When reading from left to right, you perform all the operations in the parentheses first, then the exponentiations, then all the multiplications and divisions, and finally all the additions and subtractions.

Operators	Operations
`()`	Parenthesis
`^`	Exponentiation
`* /`	Multiplication and Division
`+ -`	Addition and Subraction

Chapter 3 – Performing Operations and Storing the Results

Converting Data Types

Visual Basic .NET allows you to convert numerical values using a special routine, `ToString`, to convert a numerical value to a String data type.

The routine `ToString` is called a **method**.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.7

The following code snippets are designed to test your order of precedence knowledge. Try working out each example first; then type in the snippet and execute it. Compare your results to the answers found at the end of the chapter.

```
Private Sub btnDrill_Click(...
    lblOutput.Text = (4 + 5 * 6 - 3 / 3 + 6).ToString
End Sub
```

Answer: 39

```
Private Sub btnDrill_Click(...
    lblOutput.Text = ((4 + 5) * 6 - (3 / 3 + 6)).ToString
End Sub
```

Answer: 47

```
Private Sub btnDrill_Click(...
    lblOutput.Text = ((4 + 5) / (1 + 2)).ToString
End Sub
```

Answer: 3

Chapter 3 – Performing Operations and Storing the Results

Drill 3.7 Continued

```
Private Sub btnDrill_Click(...
    lblOutput.Text = (4 * 5 * (3 + 3)).ToString
End Sub
```

Answer: 120

```
Private Sub btnDrill_Click(...
    lblOutput.Text = (2 - 2 / 2 + 2 * 2 - 3).ToString
End Sub
```

Answer: 2

```
Private Sub btnDrill_Click(...
    lblOutput.Text = (2 + 2 * 2 ^ 3 - 3).ToString
End Sub
```

Answer: 15

Chapter 3 – Performing Operations and Storing the Results

Enforcement of Proper Data Type

Visual Basic .NET is not as strict in enforcing the use of proper data types as some other languages are.

Other languages do not produce results of one data type when the calculation is performed on operands of another data type.

Chapter 3 – Performing Operations and Storing the Results

Example: Counter Application

Create an application that acts as a counter.

A counter should start at 0 and increment by 1 each time a button is pressed.

It is also useful to have an additional button that will reset the counter to 0.

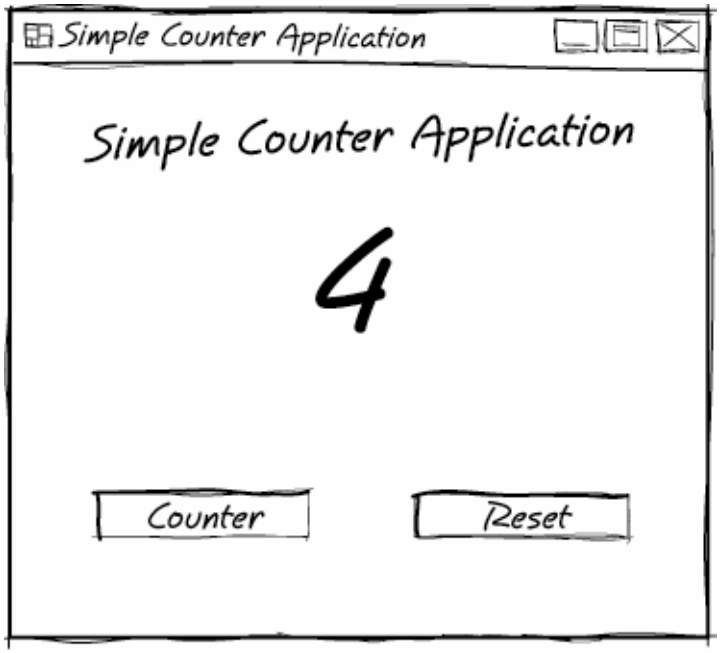


Fig 03-09

Chapter 3 – Performing Operations and Storing the Results

Perform the following steps

Create Form

Step 1: Create a new project.

Step 2: Set the `Name` property of the form to `frmCounter`.

Step 3: Set the `Text` property to `Simple Counter Application`.

Step 4: Right-click on `Form1.vb` in the Solution Explorer and rename the form to `frmCounter.vb`.

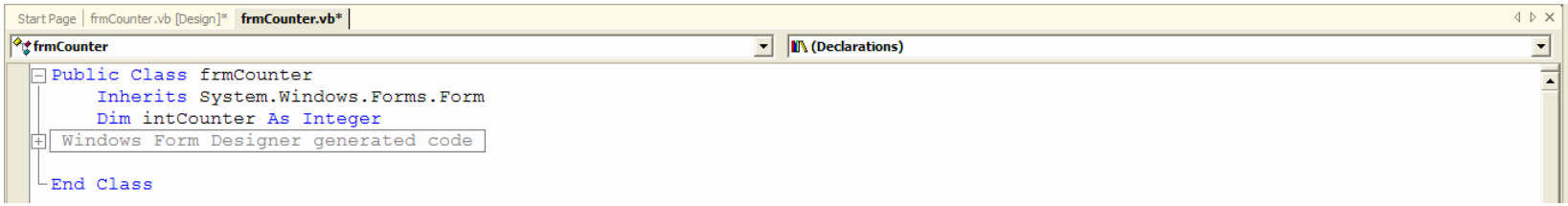
Chapter 3 – Performing Operations and Storing the Results

Add intCounter Variable

Step 1: Right-click on the form.

Step 2: Click on the View Code item in the pop-up menu.

Step 3: Your code should default to the Declarations section. The pull-downs of your code should look like this one below.



Switch back to the object view of the application by clicking on the `frmCounter.bn[Design]*` tab.

Chapter 3 – Performing Operations and Storing the Results

Add Title Label

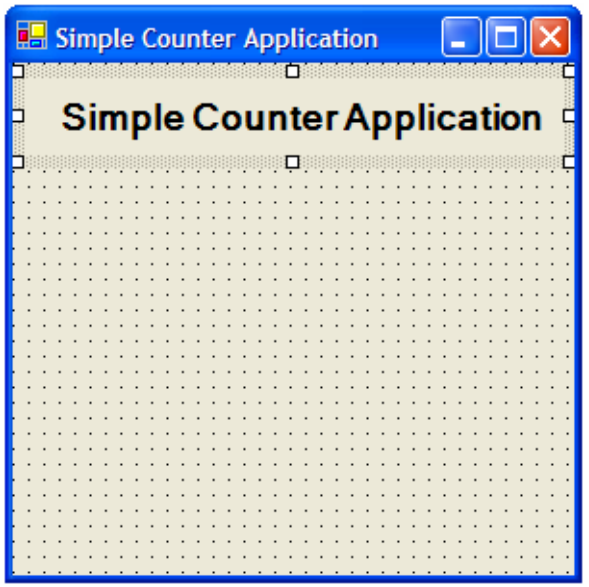
Step 1: Place a label control in the middle of the form.

Step 2: Set the `Name` to `lblTitle`.

Step 3: Set the `Text` to `Simple Counter Application`.

Step 4: Set the `Font Size` property to `14` and the `Font Bold` to `True`.

Step 5: Set the `TextAlign` property to `MiddleCenter`.



Chapter 3 – Performing Operations and Storing the Results

Add Counter Label

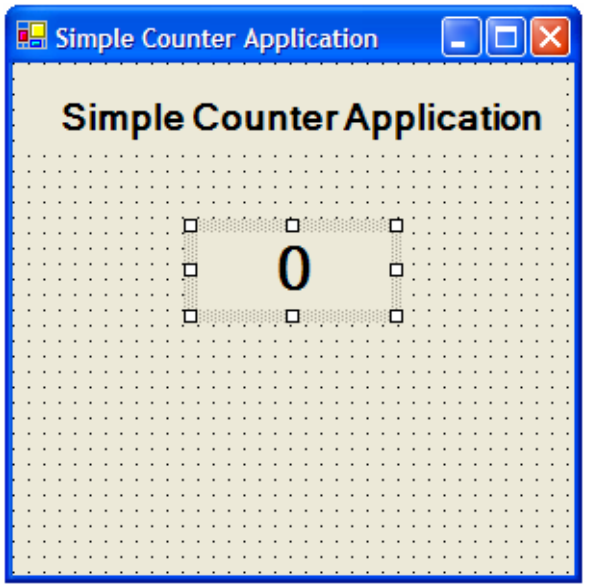
Step 1: Place a label control in the middle of the form.

Step 2: Set the `Name` to `lblCounter`.

Step 3: Set the `Text` to `0`.

Step 4: Set the `Font Size` property to `24` and the `Font Bold` to `True`.

Step 5: Set the `TextAlign` property to `MiddleCenter`.



Chapter 3 – Performing Operations and Storing the Results

Add Counter Button

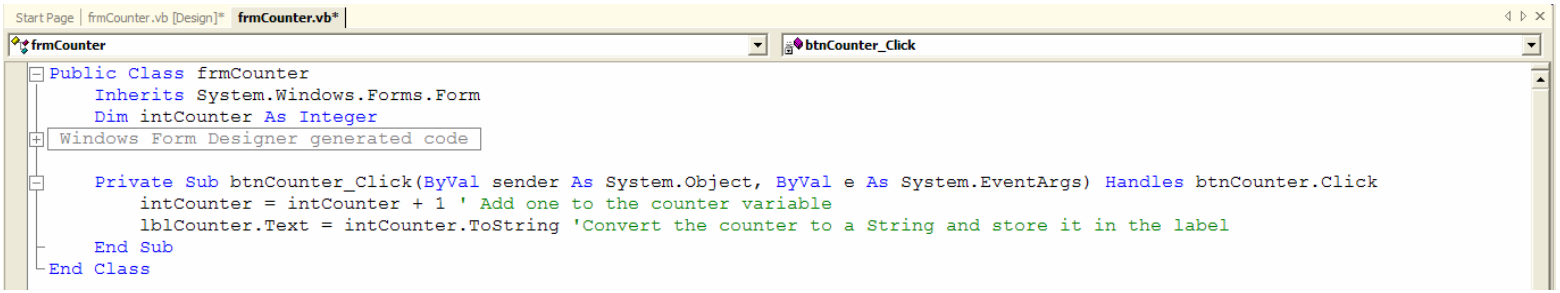
Step 1: Place a button control in the lower-left side of the form.

Step 2: Set the `Name` to `btnCounter`.

Step 3: Set the `Text` to `Counter`.

Step 4: Double-click on the button.

Step 5: Attach the code to add 1 to the counter as shown below.



Chapter 3 – Performing Operations and Storing the Results

Add Reset Button

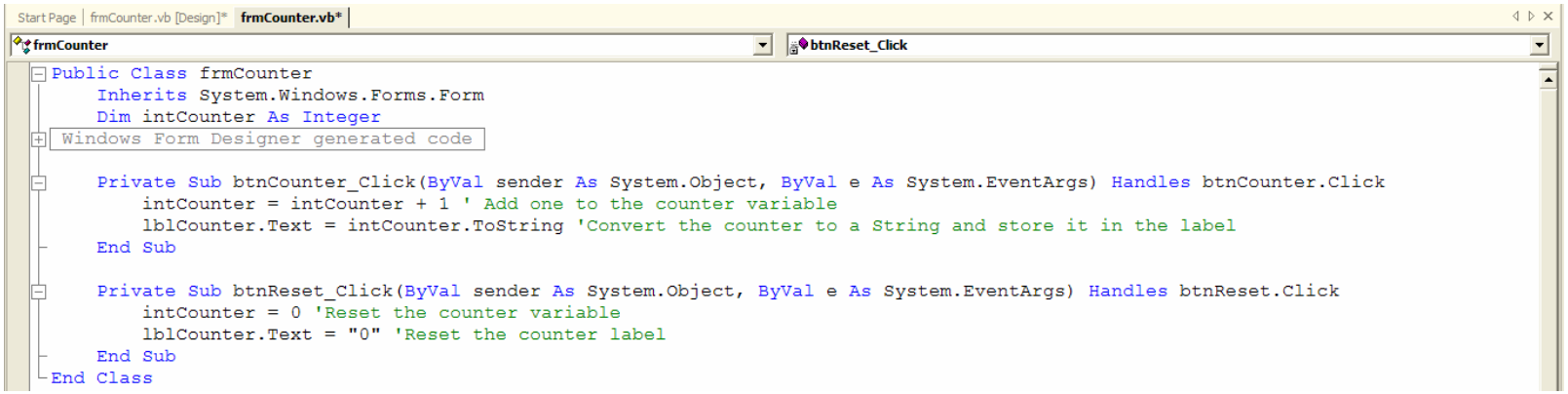
Step 1: Place a button control in the lower-right side of the form.

Step 2: Set the `Name` to `btnReset`.

Step 3: Set the `Text` to `Reset`.

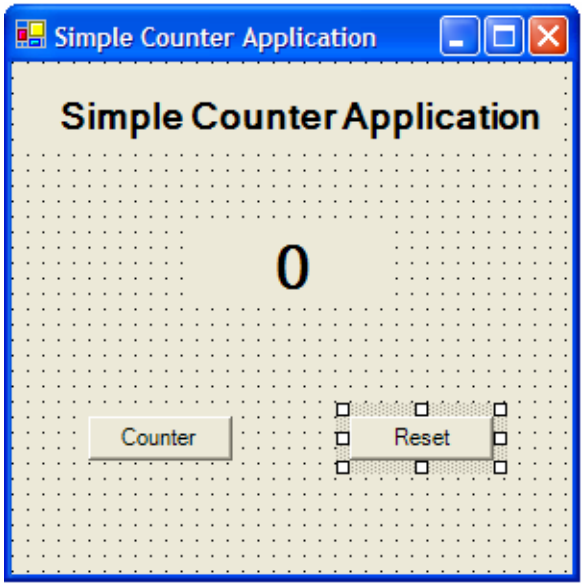
Step 4: Double-click on the button.

Step 5: Attach the code to reset the counter to 0 as shown below.



Chapter 3 – Performing Operations and Storing the Results

In the end your form should look like this:



Run this application to make sure it executes as expected.

Chapter 3 – Performing Operations and Storing the Results

3.3 Local and Global Variables

When a variable is declared within an event, it is only visible within the code for that event. Variables of this nature are known as **local** in scope.

When a variable is declared within the `Declarations` section of a form, it is visible to the entire form and known as **global** in scope.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.8

If the following three `Click` events are coded, what would be the output if the buttons were clicked in the following order: `btnInitialize`, `btnAdd`, and `btnOutput`?

```
Public Class frmDrills
     Inherits System.Windows.Forms.Form

     Private Sub btnInitialize_Click(...
         Dim intDrillValue As Integer
         intDrillValue = 10
     End Sub

     Private Sub btnAdd_Click(...
         intDrillValue =  intDrillValue + 10
     End Sub

     Private Sub btnOutput_Click(...
         MsgBox(intDrillValue.ToString())
     End Sub
End Class
```

Answer: You will receive a build error stating that `intDrillValue` is not declared, and you won't be able to run the program.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.9

If the following three `Click` events are coded, and the variable intDrillValue is declared in each of the events, what would be the output if the buttons were clicked in the following order: `btnInitialize`, `btnAdd`, and `btnOutput`?

```
Public Class frmDrills
     Inherits System.Windows.Forms.Form
     Dim intDrillValue As Integer

     Private Sub btnInitialize_Click(...
         intDrillValue = 10
     End Sub

     Private Sub btnAdd_Click(...
         intDrillValue =  intDrillValue + 10
     End Sub

     Private Sub btnOutput_Click(...
         MsgBox(intDrillValue.ToString())
     End Sub
End Class
```

Answer: When btnInitialize is clicked, intDrillValue becomes 10. When btnAdd is clicked, intDrillValue becomes 20. Finally, then btnOutput is clicked, 20 is displayed.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.10

If the following three `Click` events are coded, and the variable intDrillValue is declared in the `Declarations` section, what would be the output if the buttons were clicked in the following order: `btnInitialize`, `btnAdd`, and `btnOutput`?

```
Public Class frmDrills
     Inherits System.Windows.Forms.Form
     Dim intDrillValue As Integer

     Private Sub btnInitialize_Click(...
         Dim intDrillValue As Integer
         intDrillValue = 10
     End Sub

     Private Sub btnAdd_Click(...
         intDrillValue =  intDrillValue + 10
     End Sub

     Private Sub btnOutput_Click(...
         MsgBox(intDrillValue.ToString())
     End Sub
End Class
```

Answer: The final output of the message box is 10.

Chapter 3 – Performing Operations and Storing the Results

3.4 Constants

Values that will not change during the execution of the program are called **constants**.

By adding a name to associate with the value, your program becomes more readable.

Additionally, with the use of constants, you only have to change the value in a single place.

You wouldn't want to risk the chance that you could inadvertently change a value that shouldn't be changed.

With the constant, you cannot. That is the reason why variables are not used in place of constants.

Chapter 3 – Performing Operations and Storing the Results

Defining a Constant

In order to declare a constant, you type the keyword Const, a space, followed by the name of the constant, followed by a space, followed by an equals sign, followed by the value to set the constant to.

```
Const ConstantName = Value
```

You may also include a data type for the constant and use an expression instead of the value.

```
Const ConstantName As DataType = Value
```

Chapter 3 – Performing Operations and Storing the Results

Example: Sales Tax Calculation

Create a simple application to calculate the sales tax for a purchase. It will use a constant in order to indicate the sales tax percentage. Perform the following steps

Create Form

Step 1: Create a new project called Sales Tax Calculation.

Step 2: Set the `Name` property of the form to `frmSalesTax`.

Step 3: Set the `Text` property to `Sales Tax Calculation`.

Step 4: Right-click on `Form1.vb` in the Solution Explorer and rename the form to `frmSalesTax.vb`.

Chapter 3 – Performing Operations and Storing the Results

Add Title Label

Step 1: Click on the label control in the Control toolbox.

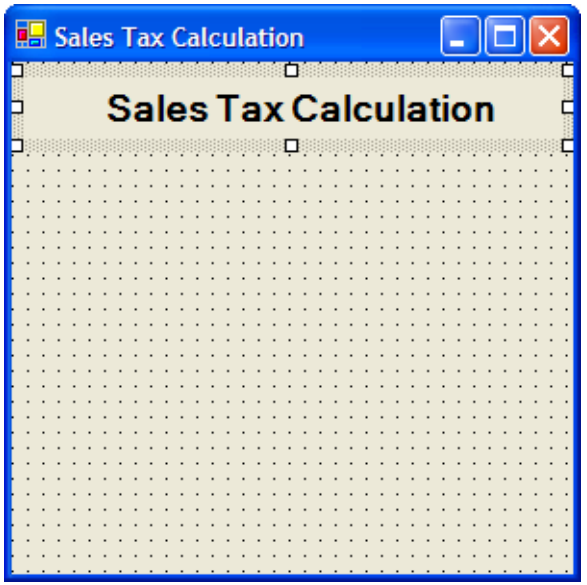
Step 2: Draw a label control on the form.

Step 3: Set the `Name` property of the label to `lblTitle`.

Step 4: Change the `Text` property to `Sales Tax Calculation`.

Step 5: Change the `Font Size` property to `14` and the `Font Bold` to `True`.

Step 6: Change the `TextAlign` property to `MiddleCenter`.



Chapter 3 – Performing Operations and Storing the Results

Add Purchase Price Label

Step 1: Place a label control on the form.

Step 2: Change the `Name` property to `lblPurchasePrice`.

Step 3: Change the `Text` property to `Purchase Price`.

Step 4: Change the `Font Size` property to `14` and the `Font Bold` property to `True`.

Add Sales Tax Label

Step 1: Place a label control on the form.

Step 2: Change the `Name` property to `lblSalesTax`.

Step 3: Change the `Text` property to `Sales Tax`.

Step 4: Change the `Font Size` property to `14` and the `Font Bold` property to `True`.

Chapter 3 – Performing Operations and Storing the Results

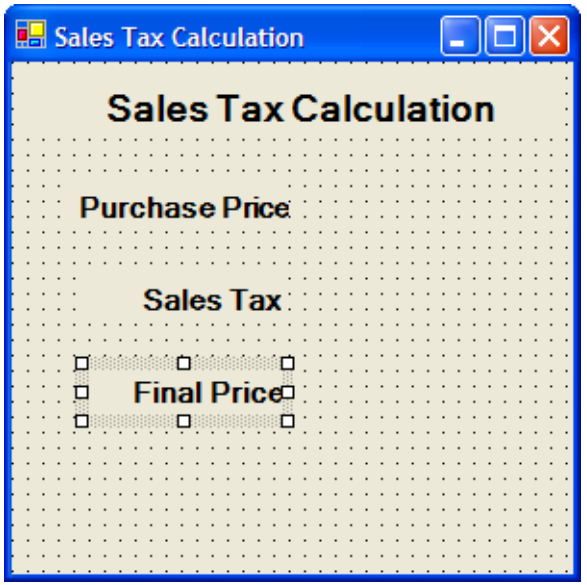
Add Final Price Label

Step 1: Place a label control on the form.

Step 2: Change the `Name` property to `lblFinalPrice`.

Step 3: Change the `Text` property to `Final Price`.

Step 4: Change the `Font Size` property to `14` and the `Font Bold` property to `True`.



Chapter 3 – Performing Operations and Storing the Results

Add Purchase Price Text Box

Step 1: Place a text box control on the form.

Step 2: Change the `Name` property to `txtPurchasePrice`.

Step 3: Erase the value in the `Text` property.

Add Sales Tax Text Box

Step 1: Place a text box control on the form.

Step 2: Change the `Name` property to `txtSalesTax`.

Step 3: Erase the value in the `Text` property.

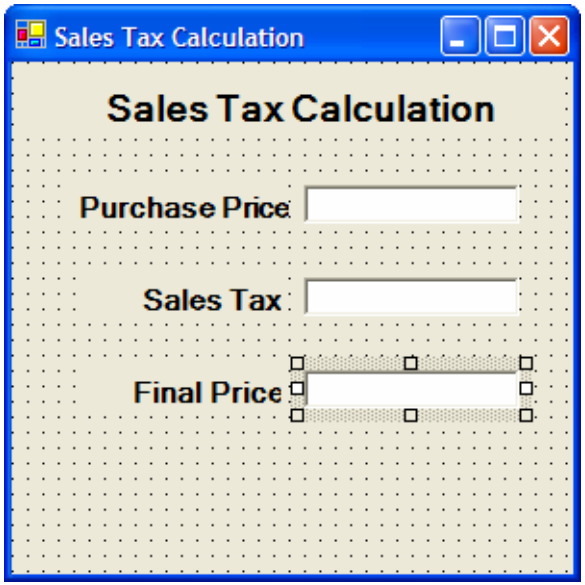
Chapter 3 – Performing Operations and Storing the Results

Add Final Price Text Box

Step 1: Place a text box control on the form.

Step 2: Change the `Name` property to `txtFinalPrice`.

Step 3: Erase the value in `Text` property.



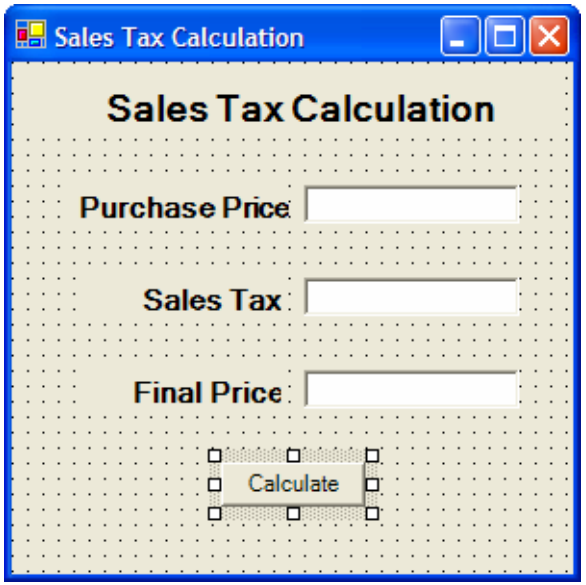
Chapter 3 – Performing Operations and Storing the Results

Add Calculation Button

Step 1: Place a button control on the form.

Step 2: Set the `Name` property of the control to `btnCalculate`.

Step 3: Change the `Text` property to `Calculate`.



Chapter 3 – Performing Operations and Storing the Results

Add Code to the Button

Step 1: Double-click on the `btnCalculate` button.

Step 2: Type the declaration to define a constant called `decSalesTaxRate` as a `Decimal` data type and set it equal to `0.06`.

Step 3: Declare three variables: `decSalesTaxAmount`, `decFinalPrice`, and `decPurchasePrice`.

Step 4: Convert the value stored in the `txtPurchasePrice` text box to a numerical value and store it in the `decPurchasePrice` variable.

Step 5: Calculate the `decSalesTaxAmount` by multiplying the `decSalesTaxRate` by `decPurchasePrice`.

Step 6: Calculate the `decFinalPrice` by adding the amount stored in the `decPurchasePrice` and `decSalesTaxAmount`.

Step 7: Store the `decSalesTaxAmount` in the `txtSalesTax` text box.

Step 8: Store the `decFinalPrice` in the `txtFinalPrice` text box.

Chapter 3 – Performing Operations and Storing the Results

Add Code to the Button Continued

Start Page | frmSalesTax.vb [Design]* | frmSalesTax.vb*

frmSalesTax | btnCalculate_Click

```
Public Class frmSalesTax
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCalculate.Click
        Const decSalesTaxRate As Decimal = 0.06 'Sales Tax Rate

        'Variables to store temporary values
        Dim decSalesTaxAmount As Decimal
        Dim decFinalPrice As Decimal
        Dim decPurchasePrice As Decimal

        'Convert text in TextBox to a numerical value
        decPurchasePrice = Val(txtPurchasePrice.Text)

        'Calculate the Amount of the Sales Tax
        decSalesTaxAmount = decSalesTaxRate * decPurchasePrice

        'Calculate the total price with the sales tax included
        decFinalPrice = decPurchasePrice + decSalesTaxAmount

        txtSalesTax.Text = decSalesTaxAmount.ToString
        txtFinalPrice.Text = decFinalPrice.ToString
    End Sub
End Class
```

Chapter 3 – Performing Operations and Storing the Results

3.5 Complex Operators

Visual Basic .NET provides several complex operators.

You will often wish to perform such mathematical operations as adding a number to, subtracting a number from, or multiplying a number by an existing variable and store the result back in the same variable.

Refer to this table for a list of operators for that purpose:

Operation	Long Way of Writing the Statement	Short Way of Writing the Statement
Addition	`intVar = intVar + 1`	`intVar += 1`
Subtraction	`intVar = intVar – 1`	`intVar -= 1`
Division	`intVar = intVar / 1`	`intVar /= 1`
Multiplication	`intVar = intVar * 1`	`intVar *= 1`
String concatenation	`strVar = strVar & "New Text"`	`strVar &= "New Text"`

Chapter 3 – Performing Operations and Storing the Results

Drill 3.11

What is the output if the `btnOperators' Click` event is executed?

```
Private Sub btnOperators_Click(...
    Dim intDrillValue As Integer

    intDrillValue = 10
    intDrillValue += 5

    MsgBox(intDrillValue.ToString)
End Sub
```

Answer: The final value of intDrillValue is 15 and it is output in a message box.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.12

What is the output if the `btnOperators' Click` event is executed?

```
Private Sub btnOperators_Click(...
    Dim intDrillValue As Integer

    intDrillValue = 1
    intDrillValue *= 5
    intDrillValue += 5

    MsgBox(intDrillValue.ToString)
End Sub
```

Answer: The final value of intDrillValue is 10 and it is output in a message box.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.13

What is the output if the `btnOperators' Click` event is executed?

```
Private Sub btnOperators_Click(...
    Dim strDrillValue As String

    strDrillValue = "This "
    strDrillValue &= "and "
    strDrillValue &= "that"

    MsgBox(strDrillValue)
End Sub
```

Answer: "This and that" is output in the message box.

Chapter 3 – Performing Operations and Storing the Results

Drill 3.14

What is the output if the `btnOperators' Click` event is executed?

```
Private Sub btnOperators_Click(...
    Dim strDrillValue As String

    strDrillValue = "This "
    strDrillValue = "and "
    strDrillValue = "that"

    MsgBox(strDrillValue)
End Sub
```

Answer: "that" is output in the message box.

Chapter 3 – Performing Operations and Storing the Results

3.6 Using the Debugger

As your programs become more complex, you will need more sophisticated ways of determining the source of errors.

You must learn how to use the **Debugger**.

You will use the previous example and step through its execution.

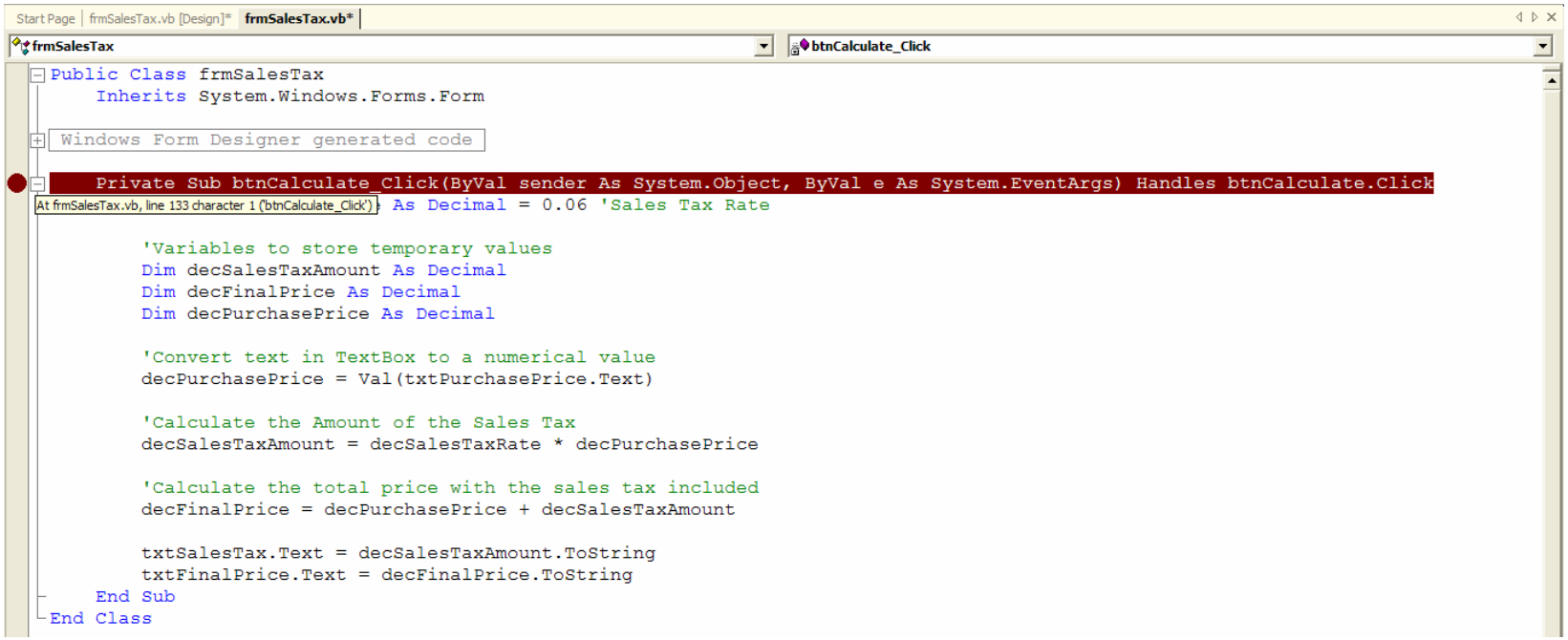
You will set a breakpoint at the start of the code you wrote.

A **breakpoint** is a signal to the Debugger to stop the execution of the application and wait for further instructions on how to continue executing the application

Chapter 3 – Performing Operations and Storing the Results

Start the Debugger

Step 1: A breakpoint is set by clicking to the left of the code you wish to be set as the breakpoint. In the figure below the breakpoint is set to the beginning of the `Click` event code for the `btnCalculate` button.

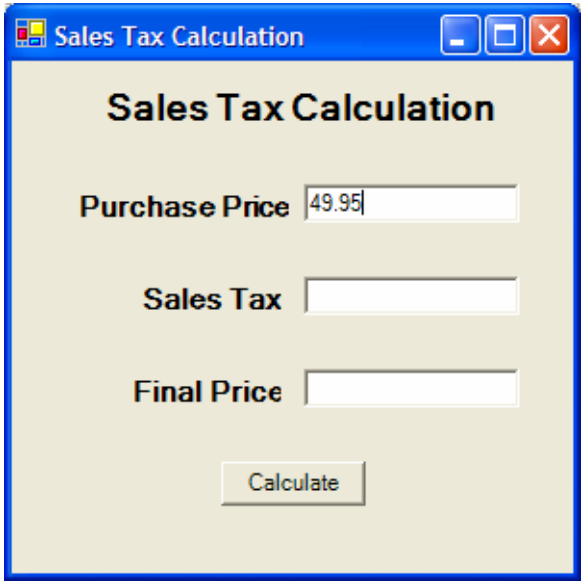


Chapter 3 – Performing Operations and Storing the Results

Execute the Application

Step 2: Start running the application in the normal manner by clicking on the `Start` button or hitting the `<F5>` key.

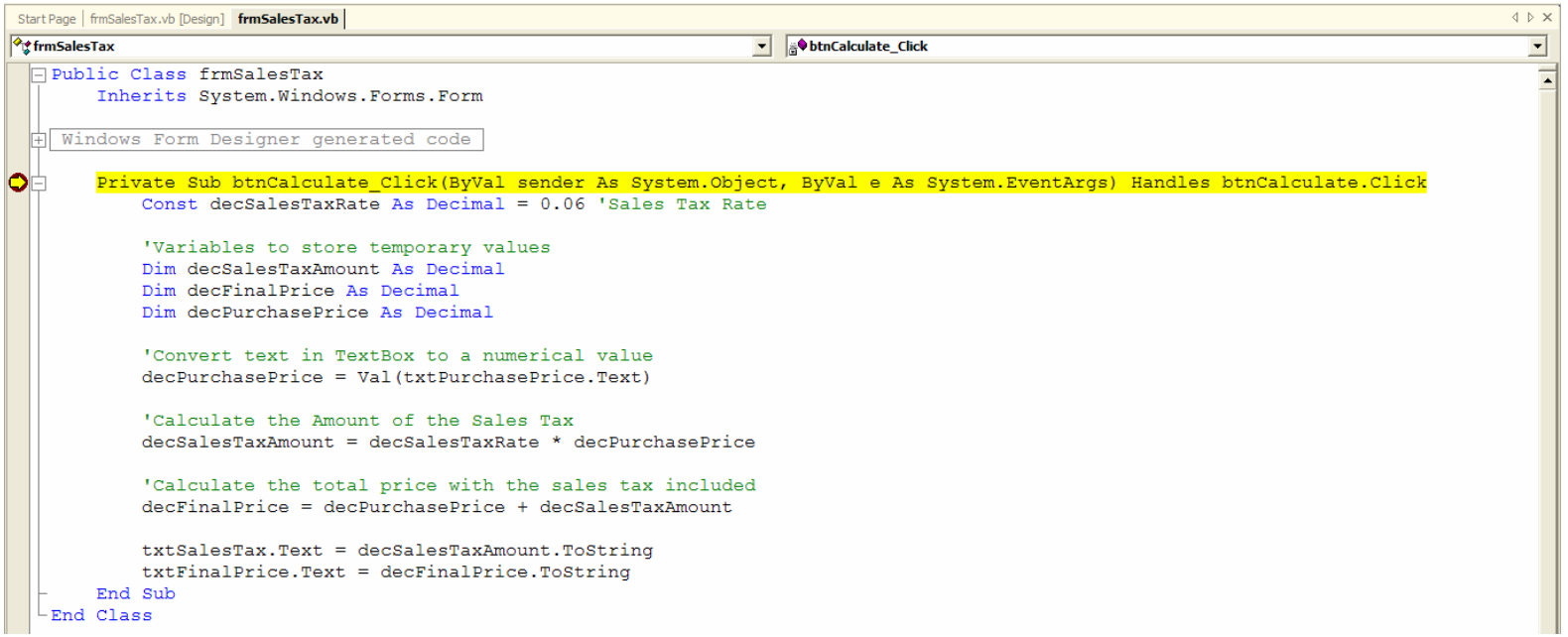
Step 3: Enter a value for the purchase price, `49.95`. Do not enter the dollar sign.



Chapter 3 – Performing Operations and Storing the Results

Stepping Into Code

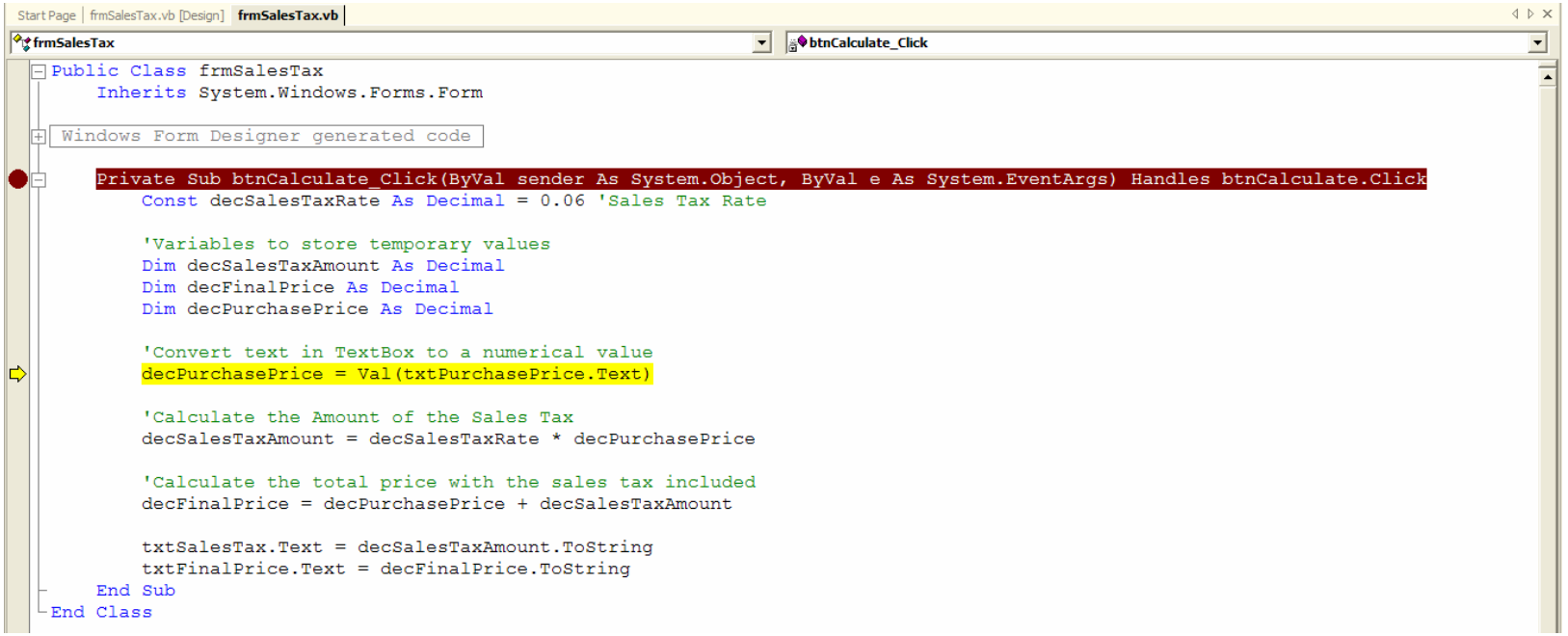
Step 4: Click on the `btnCalculate` button. Notice that instead of executing the code, the actual code is displayed. You are now in the Visual Basic .NET Debugger. The yellow highlighting indicated what line you are about to execute. You can step through the `Click` event line by line by clicking on the `Debug` menu item and then clicking on `Step Into` or you can press the `<F11>` key.



Chapter 3 – Performing Operations and Storing the Results

Stepping Over Code

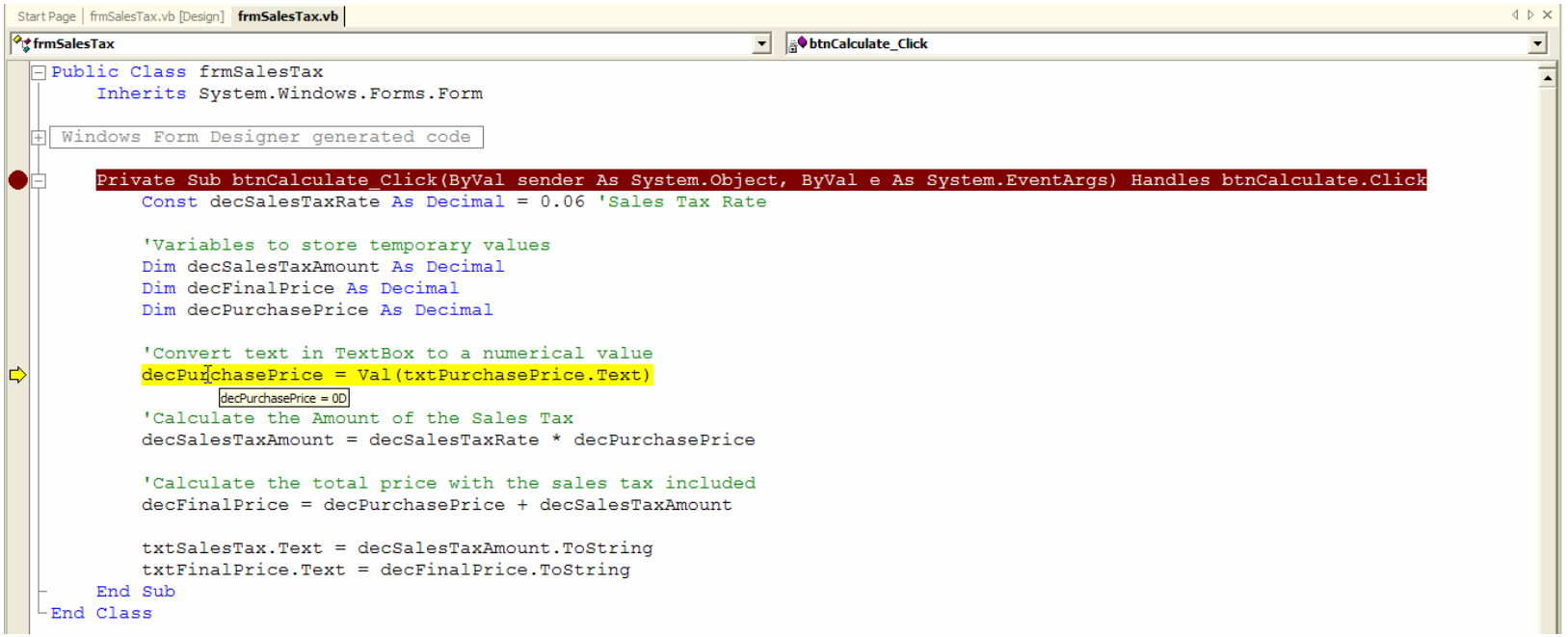
Step 5: Unlike in Step 4, to move to the next line of code, use the <F10> key instead of <F11>. The <F10> key will step over code instead of stepping into it. By stepping over code you will prevent the accidental entry into additional code that may complicate the tracing of the application. Press the <F10> key once. Notice how the yellow line skips over the declarations and is over the first line of code to be executed. You cannot trace the declaration of variables.



Chapter 3 – Performing Operations and Storing the Results

Displaying the Initial Value

Step 6: If you move the mouse pointer over objects and variables and then pause, the object's or variable's value will be displayed in a mini pop-up window.

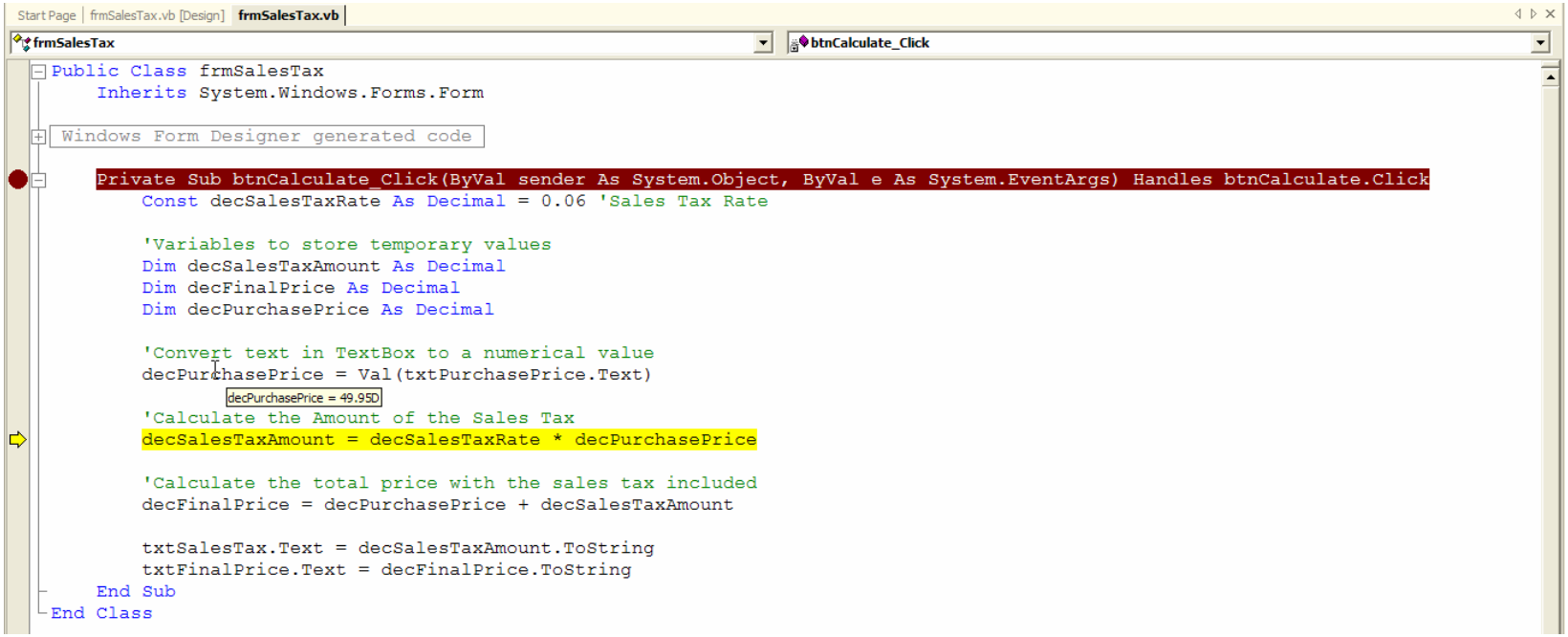


Chapter 3 – Performing Operations and Storing the Results

Displaying Value

Step 7: Press the `<F10>` key one more time. The yellow highlighting is now over the next line of code that is about to be executed.

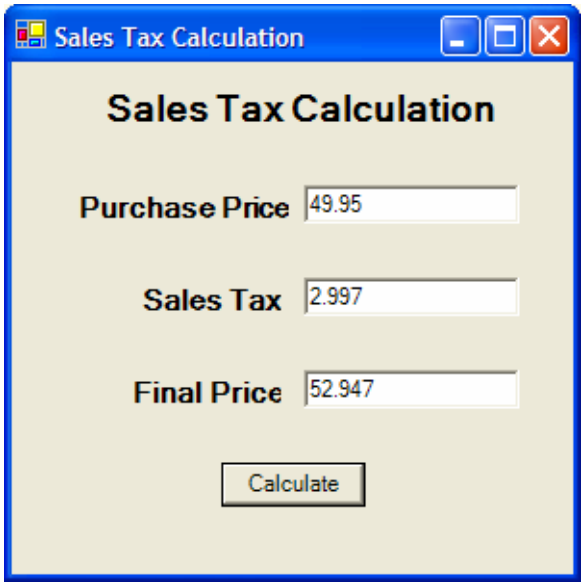
Step 8: Place your mouse pointer over the variable `decPurchasePrice to see its value`.



Chapter 3 – Performing Operations and Storing the Results

Resume Execution

Step 9: You can either press the <F5> key once or press <F10> a few more times and the rest of the code is executed.



Chapter 3 – Performing Operations and Storing the Results

3.7 Case Study

Problem Description

This case study will modify the one from Chapter 3 and add the computational functionality that was missing. You need to add a `btnCalculate` button that will automatically calculate a person's weekly pay as well as the total payroll cost.

A person's weekly pay is calculated by multiplying the number of hours a person worked by a fixed hourly rate of $9.50/hour.

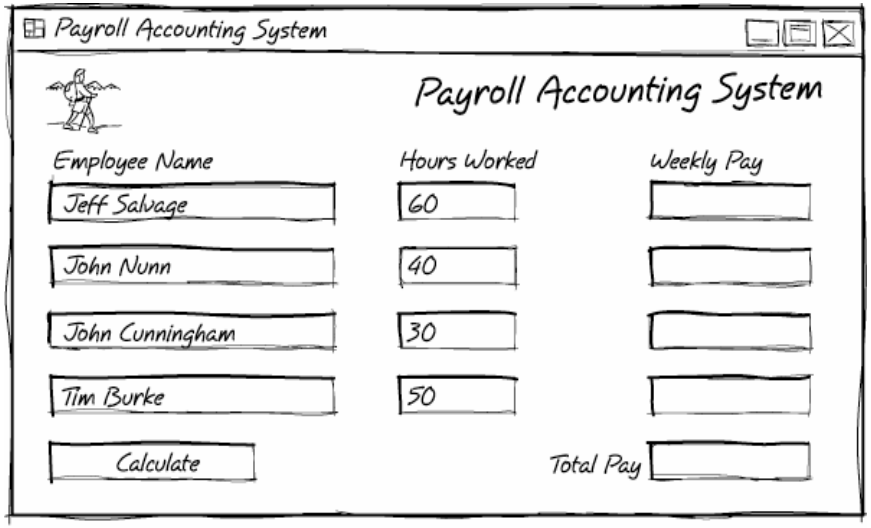


Fig 03-29

Chapter 3 – Performing Operations and Storing the Results

Problem Description Continued

After the btnCalculate button is clicked, the sample output for input shown previously should be like this:

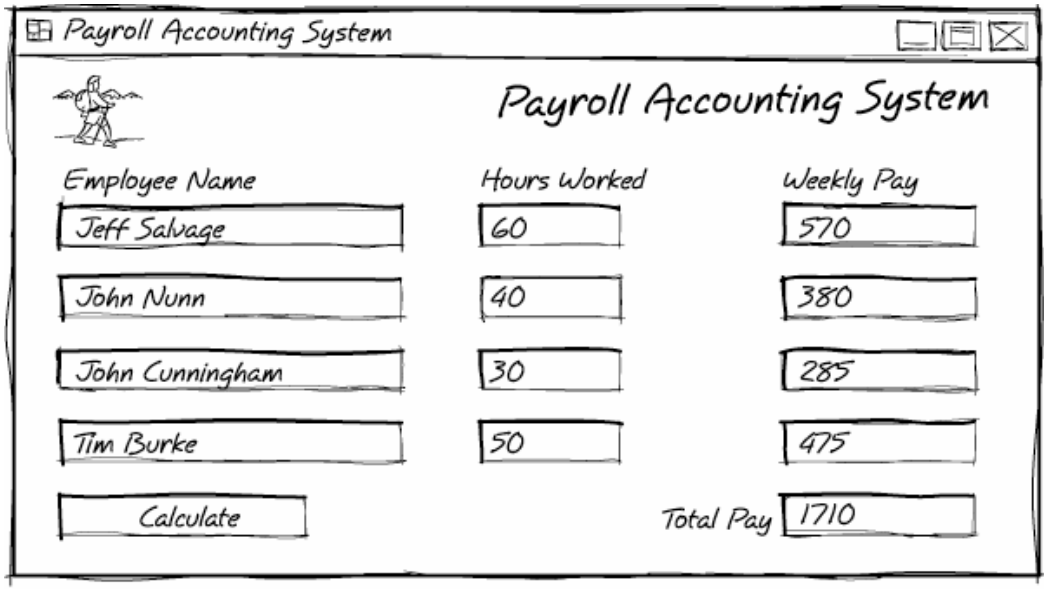


Fig 03-30

Chapter 3 – Performing Operations and Storing the Results

Problem Discussion

The solution to the problem will entail two basic steps. You must compute the values for weekly and total pay, and then you must assign those values to the text box controls for display.

Problem Solution

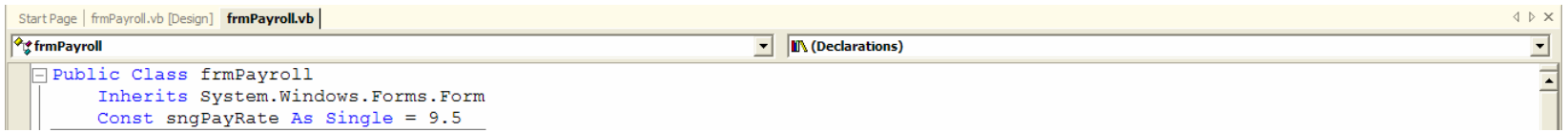
You need to add a button control, `btnCalculate`. This control will perform the calculations necessary for the application.

The use of constant in this solution is desirable to indicate the pay rate.

To set the constant, perform the following steps:

Step 1: Right-click the mouse and click on `View Code` in the pop-up menu.

Step 2: Type "`Const sngPayRate As Single = 9.5`".



Chapter 3 – Performing Operations and Storing the Results

Place a Button

Step 1: Click on the `frmPayroll.vb [Design]` tab.

Step 2: Click on the button control in the toolbox.

Step 3: Draw a button control on the form.

Step 4: Change the `Name` property to `btnCalculate`.

Step 5: Change the `Text` property to `Calculate`.

Step 6: Add the code for the button.

Chapter 3 – Performing Operations and Storing the Results

Button Code

```
Private Sub btnButton_Click(...
    'Temporary Variables to Store Calculations
    Dim decTotalPay As Decimal
    Dim decWeeklyPay As Decimal

    'First Person's Calculations
    'Compute weekly pay of 1st person
    decWeeklyPay = Val(txtHours1.Text) * sngPayRate
    'Convert weekly pay to String and output
    txtWeeklyPay1.Text = decWeeklyPay.ToString
    'Initialize total pay to first person's weekly pay
    decTotalPay = decWeeklyPay

    'Second Person's Calculations
    'Compute weekly pay of 2nd person
    decWeeklyPay = Val(txtHours2.Text) * sngPayRate
    'Convert weekly pay to String and output
    txtWeeklyPay2.Text = decWeeklyPay.ToString
    'Add to total pay 2nd person's pay
    decTotalPay += decWeeklyPay
```

Chapter 3 – Performing Operations and Storing the Results

Button Code Continued

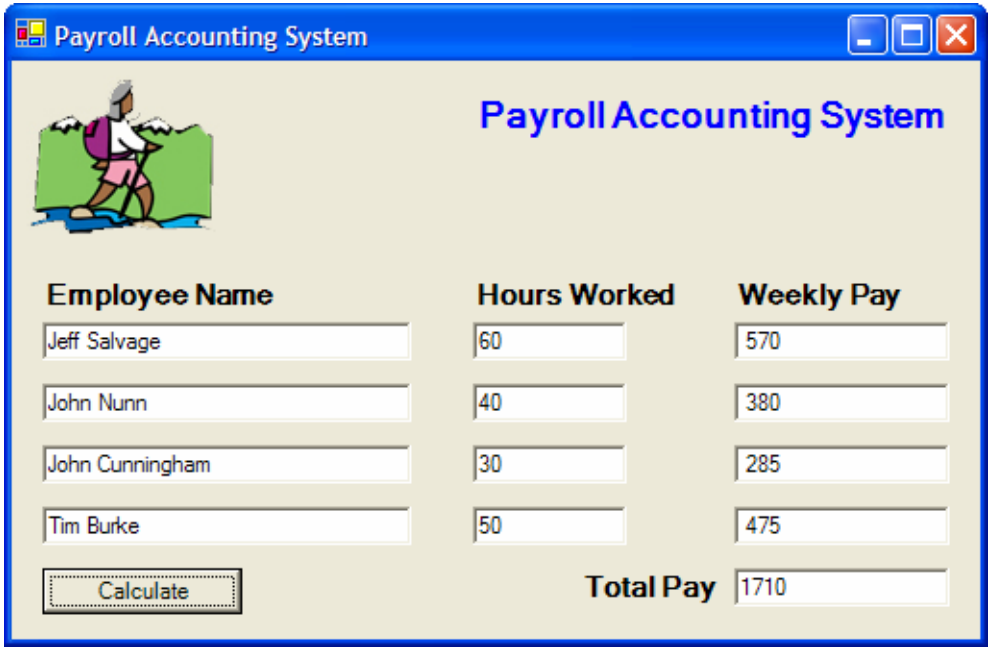
```
    'Third Person's Calculations
    'Compute weekly pay of 3rd person
    decWeeklyPay = Val(txtHours3.Text) * sngPayRate
    'Convert weekly pay to String and output
    txtWeeklyPay3.Text = decWeeklyPay.ToString
    'Add to total pay 3rd person's pay
    decTotalPay += decWeeklyPay

    'Fourth Person's Calculations
    'Compute weekly pay of fourth person
    decWeeklyPay = Val(txtHours4.Text) * sngPayRate
    'Convert weekly pay to String and output
    txtWeeklyPay4.Text = decWeeklyPay.ToString
    'Add to total pay 4th person's pay
    decTotalPay += decWeeklyPay

    'Convert Total Pay to a string and copy to TextBox
    txtTotalPay.Text = decTotalPay.ToString
End Sub
```

Chapter 3 – Performing Operations and Storing the Results

Final Result



Chapter 3 – Performing Operations and Storing the Results

Converting Data Types

When a value is stored in a text box, it is stored as a `String`. In order to perform mathematical calculations, you need to convert it to a numerical value.

Built-in function `Val` will return the numerical value for the `String` placed within the parentheses.

To place the value back into the text box, you use the `ToString` method, which converts a numerical value to a `String` data type.

Chapter 3 – Performing Operations and Storing the Results

Coach's Corner

Using MsgBox

You can use a **message box** to display information to the user of the application but not take up space on your form for it.

The syntax of a simple call to a message box is illustrated below:

```
MsgBox ("Message")
```

`MsgBox` is the function that indicates to Visual Basic .NET that you wish a message to be displayed in a small window. Small windows like the one used for a message box are commonly referred to as a **dialog box**.

To display a message, place any text you want displayed within a series of double quotes.

Chapter 3 – Performing Operations and Storing the Results

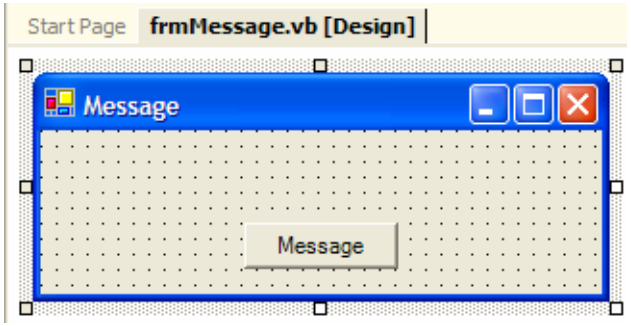
Using MsgBox Continued

Follow these steps to create a message box:

Step 1: Add a button to a blank form.

Step 2: Change the `Name` property of the button to `btnMessage`.

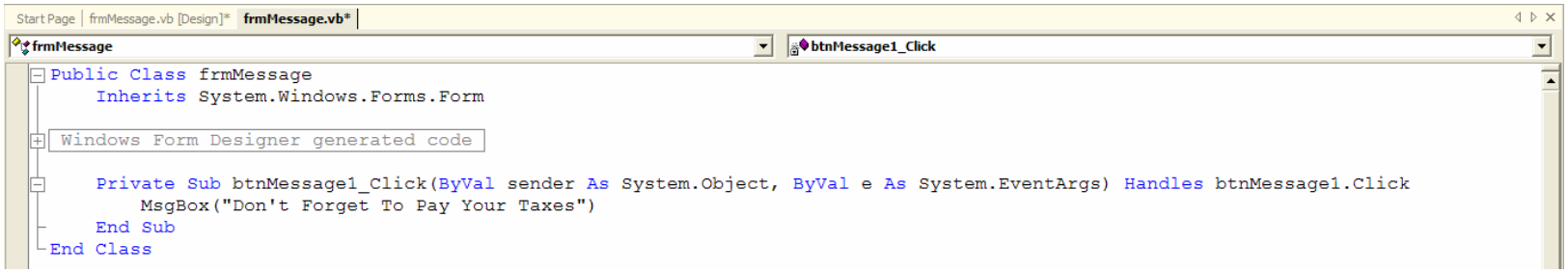
Step 3: Change the `Text` property of the button to `Message`.



Chapter 3 – Performing Operations and Storing the Results

Using MsgBox Continued

Step 4: Assign the code `MsgBox("Don't Forget to Pay Your Taxes")` to the button.



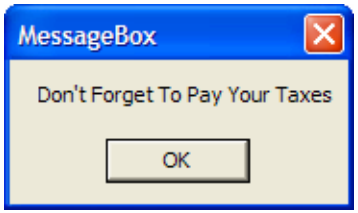
Chapter 3 – Performing Operations and Storing the Results

Using MsgBox Continued

Step 5: Run the application.

Step 6: Click on the button.

Step 7: The message box appears.



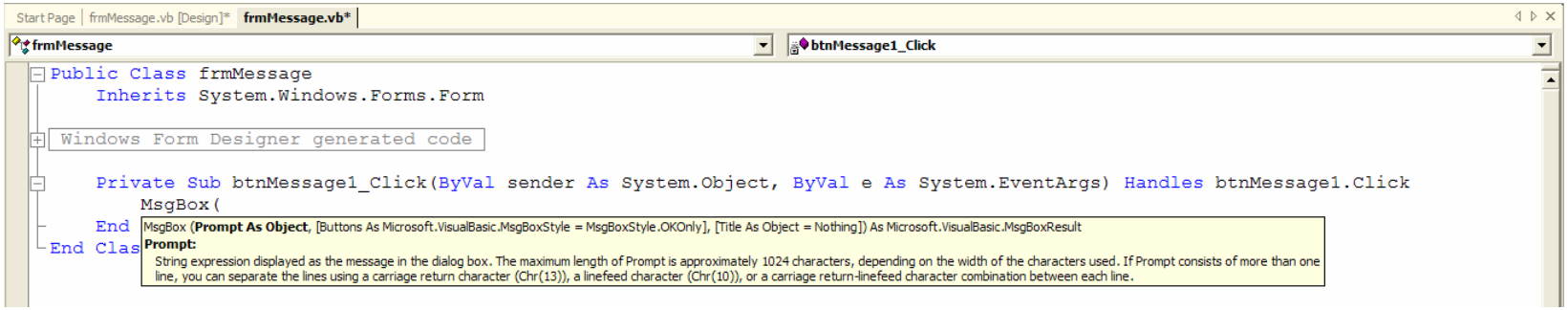
Chapter 3 – Performing Operations and Storing the Results

More Complex Message Boxes

When you typed `MsgBox` and then a space, additional information appeared. This information serves as a guide for you to provide additional options to the `MsgBox` command.

The first option is for the `Prompt`. The `Prompt` is the value that you typed to be displayed.

The other items are optional ways of further modifying the message box displayed. By selecting different values, you can change the way your message box is displayed.



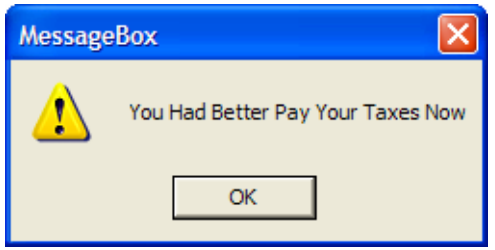
Chapter 3 – Performing Operations and Storing the Results

Examples of Using MsgBox

Here are two examples of code followed by the resulting message box.

```
MsgBox ("Division By Zero Occurred", MsgBoxStyle.Critical)
```


```
MsgBox ("You Had Better Pay Your Taxes Now", MsgBoxStyle.Exclamation)
```

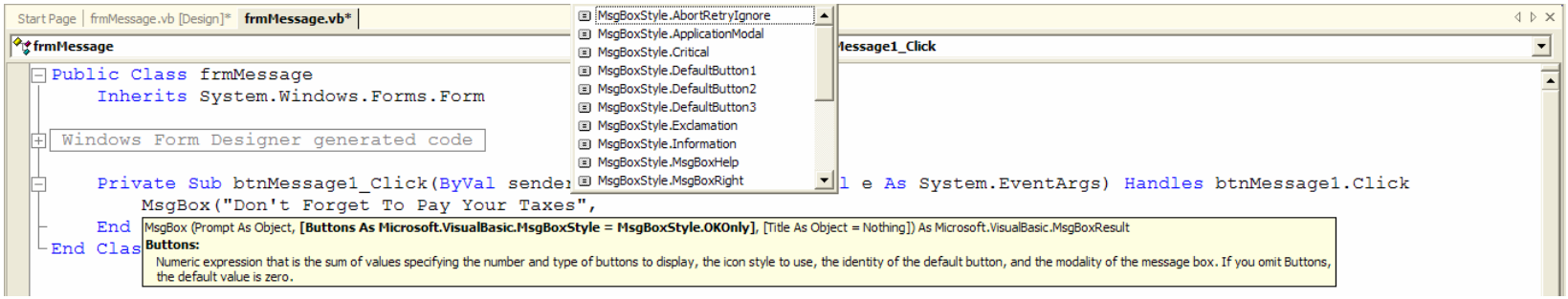


Chapter 3 – Performing Operations and Storing the Results

More Complex Message Boxes Continued

You should explore more options by checking the help system or by exploring the type ahead feature.

If you have typed your commands up to and including the comma after the prompt, a list appears that you may choose from. Most options are self-explanatory.



Chapter 3 – Performing Operations and Storing the Results

Adding a Title to the Message Box

By adding an additional parameter to the MsgBox statement, you can specify a title for the title bar of the message box so that the message box does not use the title of the project as the title of the window.

```
MsgBox ("Don't Forget To Pay Your Taxes", MsgBoxStyle.OkOnly, "Tax Message")
```