

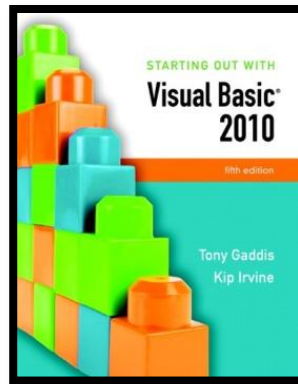


STARTING OUT WITH

Visual Basic® 2010

fifth edition

Tony Gaddis
Kip Irvine



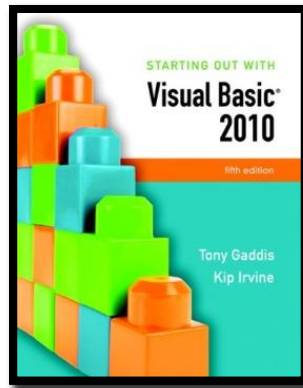
Chapter 3

Variables and Calculations

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.



Section 3.1

GATHERING TEXT INPUT

In this section, we use the TextBox control to gather input the user has typed on the keyboard. We also alter a form's tab order and assign keyboard access keys to controls.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

The TextBox Control

- A text box is a rectangular area on a form that accepts input from a keyboard
- Tutorial 3-1 provides an example in the use of a text box



Using the Text Property in Code

- The TextBox control's Text property can be accessed in code the same way you access other properties
- For Example:
 - The contents of the Text property can be assigned into a Label control's Text property:
 - **lblInfo.Text = txtInput.Text**
 - The contents of the Text property can be displayed in a message box
 - **MessageBox.Show(txtInput.Text)**

Clearing a Text Box

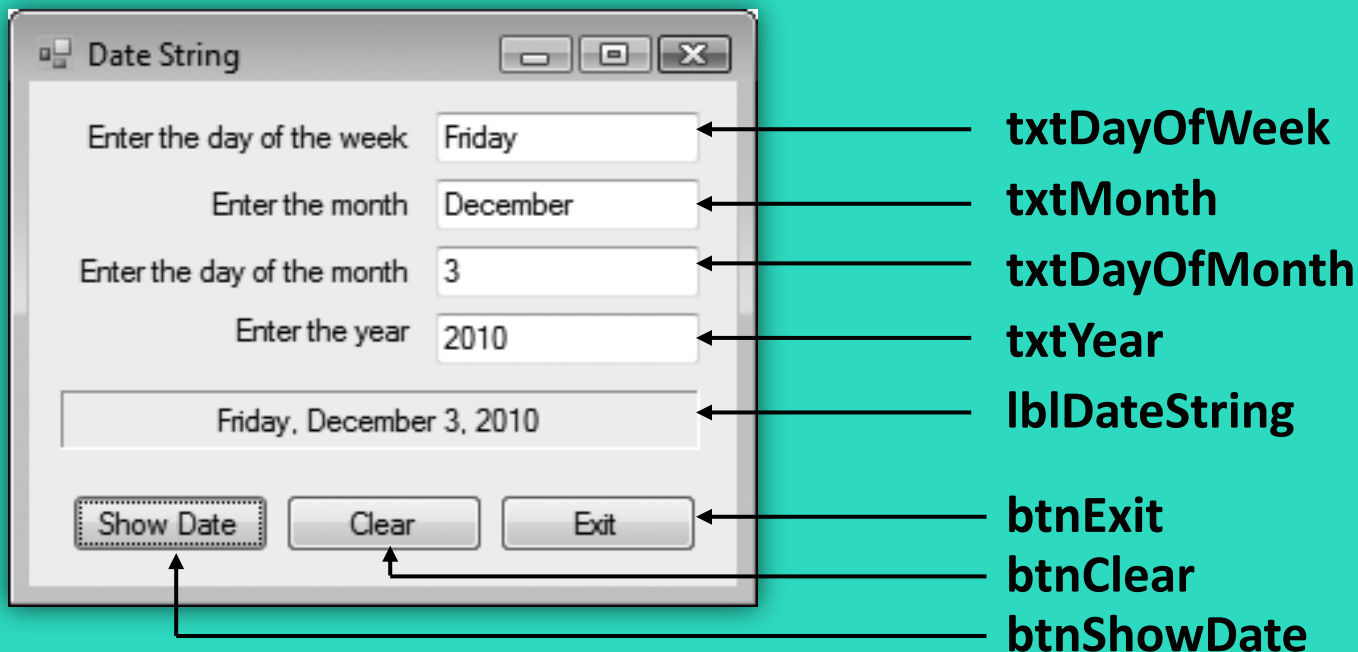
- Can be done with an assignment statement:
 - **txtInput.Text = String.Empty**
 - assigning the predefined constant `String.Empty` replaces whatever text was in `txtInput` with an empty string
- Can also be done with a method:
 - **txtInput.Clear()**
 - `Clear` is a *Method*, not a *Property*
 - Methods are *actions* – as in clearing the text
 - Uses the form *Object.Method*

String Concatenation

- Assume the user has entered their name into the TextBox txtName
- Label lblGreeting can say, “Hello” to any name found in the TextBox
 - **lblGreeting.Text = "Hello " & txtName.Text**
 - Appends user name in txtName.Text to “Hello ” and stores result in text property of lblGreeting

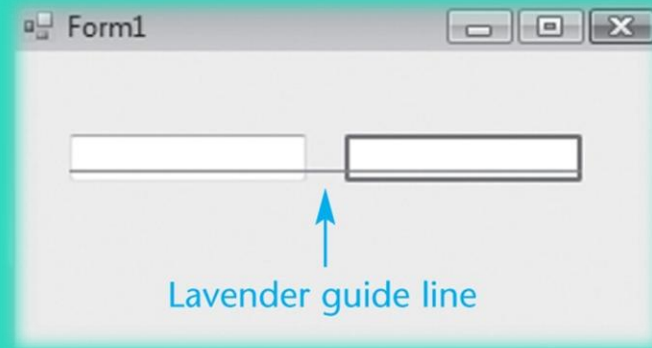
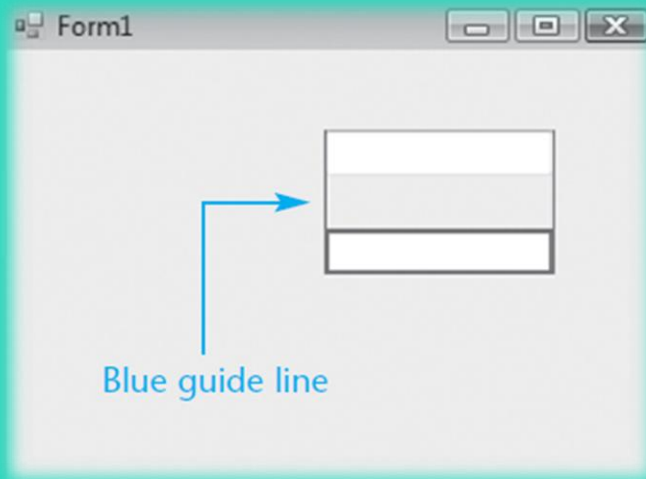
String Concatenation

- Tutorial 3-2 provides another example of how to concatenate strings from text boxes



Aligning Controls in Design Mode

- When dragging a control to a form, it can be aligned with a control already on the form
 - Blue guide lines appear for vertical alignment
 - Lavender guide lines for horizontal alignment



The Focus Method

- For a control to have the **focus** means that it is ready to receive the user's input
- In a running form, one and only one of the controls on the form may have the focus
- Only a control capable of receiving some sort of input may have the focus
- The focus can be set to a control in code using the Focus method:

```
txtUserName.Focus()
```

The Focus Method

- You can tell which control has focus by its characteristics:
 - When a TextBox has focus, it will have a blinking cursor or its text will be highlighted
 - When a button, radio button, or a check box has focus, you'll see a thin dotted line around the control
- Tutorial 3-3 shows an example of the Focus method

Controlling a Form's Tab Order with the TabIndex Property

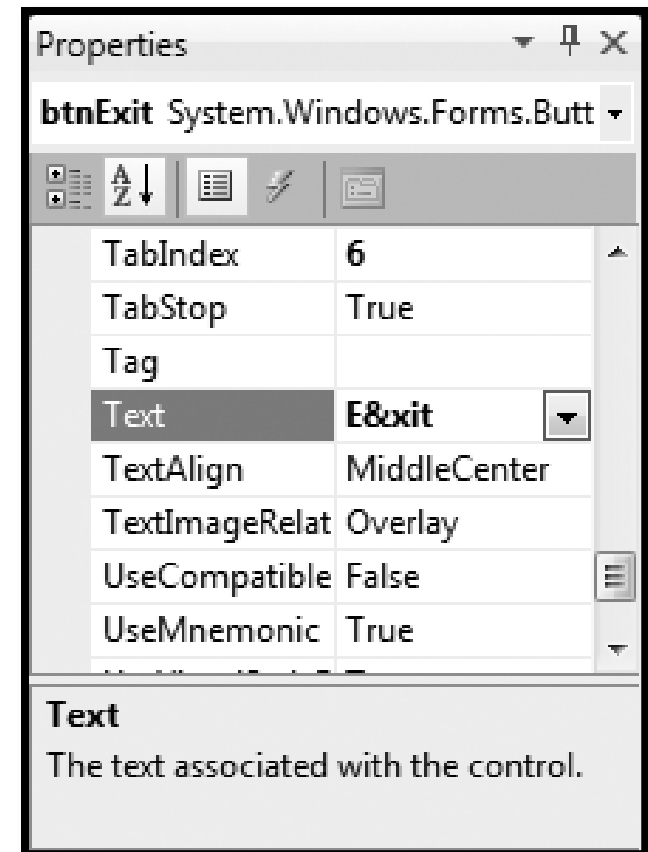
- Tab key steps focus from one control to the next
- This order is set by the `TabIndex` property
- The Tab key causes the focus to jump to the control with the next highest `TabIndex` value
- The `TabIndex` property is best changed with the Tab Order option from the View menu
 - Displays the form in `tab order selection mode`
 - Set a new tab order by clicking the controls in the order you want
 - This sets the numeric `TabIndex` value

Assigning Keyboard Access Keys to Buttons

- Say your form had a button with the text “Exit” on it
- You can allow the user to activate the button using Alt-X instead of a mouse click
- Just change the button text property to “E&xit”
- The character following the '&' (x in this case) is designated as an access key
- Be careful not to use the same access key for two different buttons

'&' Has Special Meaning in a Button

- Note that the '&' in “E&xit” does not display in the button control on the form
- It simply establishes the Alt Key access
- In order to actually display an '&' on a button, it must be entered as "&&“
 - Button text *Save & Exit* is entered as *Save && Exit*

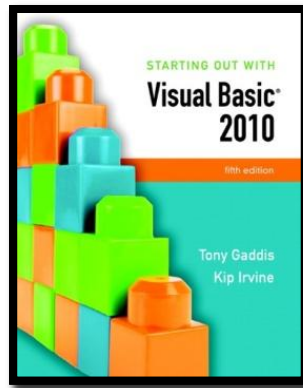


Setting the Accept Button

- The `accept` button is a button that is implicitly activated if the user hits the Enter Key
- The `AcceptButton` Property designates which button on the form will behave in this manner
- The button clicked most frequently on a form is usually assigned as the accept button

Setting the Cancel Button

- The `cancel button` is a button that is implicitly activated if the user hits the Escape Key
- The `CancelButton` Property designates which button on the form will behave in this manner
- Any exit or cancel button on a form is a candidate to become the cancel button
- Tutorial 3-5 provides examples of setting access keys, accept, and cancel buttons



Section 3.2

VARIABLES AND DATA TYPES

Variables hold data that may be manipulated, used to manipulate other data, or remembered for later use.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Why Have Variables?

- A **variable** is a storage location in the computer's memory, used for holding information while the program is running
- The information that is stored in a variable may change, hence the name "variable"

What Can You Do With Variables?

- Copy and store values entered by the user, so they may be manipulated
- Perform arithmetic on values
- Test values to determine that they meet some criterion
- Temporarily hold and manipulate the value of a control property
- Remember information for later use in the program

How to Think About Variables

- You the programmer make up a name for the variable
- Visual Basic associates that name with a location in the computer's RAM
- The value currently associated with the variable is stored in that memory location

Declaring Variables

- A variable declaration is a statement that creates a variable in memory
- The syntax is:

Dim VariableName As DataType

- **Dim** (short for Dimension) is a keyword
 - **VariableName** is the programmer designated name
 - **As** is a keyword
 - **DataType** is one of many possible keywords for the type of value the variable will contain
- Here is an example of a variable declaration:

Dim intLength as Integer

Declaring Multiple Variables

- Several variables may be declared in one statement if they all hold the same type of value

Dim intLength, intWidth, intHeight as Integer

- Or this can be done in 3 separate statements

Dim intLength as Integer

Dim intWidth as Integer

Dim intHeight as Integer

Variable Naming Rules

- The first character of a variable name must be a letter or an underscore
- Subsequent characters may be a letter, underscore, or digit
 - Thus variable names cannot contain spaces or periods (or many other kinds of characters)
- Visual Basic keywords cannot be used as variable names

Variable Naming Conventions

- Naming conventions are a guideline to help improve readability but not required syntax
- A variable name should describe its use
- Each data type has a recommended prefix, in lower case, that begins the variable name
- The 1st letter of each subsequent word in the variable name should be capitalized
 - intHoursWorked - an integer variable
 - strLastName - a string (or text) variable

Setting the Value of a Variable

- An assignment statement is used to set the value of a variable, as in:
 - Assign the value 112 to the variable length
 - **length = 112**
 - Assign the string literal “Good Morning “ followed by the contents of the text box txtName to the variable greeting
 - **greeting = "Good Morning " & txtName.Text**
- An assignment changes only the left operand
- The right operand remains unchanged

Visual Basic Data Types

- Integer types
 - Byte
 - Short
 - Integer
 - Long
- Floating-Point types
 - Single
 - Double
 - Decimal
- Other data types
 - Boolean
 - Char
 - String
 - Date

Integer Data Types

- For values that will always be a whole number
- Usually name a variable starting with a 3 or 4 letter prefix indicating the variable's type

Data Type	Naming Prefix	Description
Byte	byt	Unsigned integer from 0 to 255
Short	shrt	Signed integer from -32,768 to 32,767
Integer	int	Signed integer from -2,147,483,648 to 2,147,483,647
Long	lng	Signed integer from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating-Point Data Types

- For values that may have fractional parts
- Single used most frequently
- Double sometimes used in scientific calculations
- Decimal often used in financial calculations

Data Type	Naming Prefix	Description
Single	sng	As large as 10^{38} plus or minus, 7 decimal positions
Double	dbl	As large as 10^{308} plus or minus, 15 decimal positions
Decimal	dec	As large as 10^{29} plus or minus, 29 decimal positions

Other Common Data Types

- Boolean – variable naming prefix is bln
 - Holds 2 possible values, True or False
- Char – variable naming prefix is chr
 - Holds a single character
 - Allows for characters from other languages
- String – variable naming prefix is str
 - Holds a sequence of up to 2 billion characters
- Date – variable naming prefix is dat or dtm
 - Can hold date and/or time information

The String Data Type

- A string literal is enclosed in quotation marks
 - The following code assigns the name Jose Gonzales to the variable strName
Dim strName as string
strName = "Jose Gonzales"
- An empty string literal can be coded as:
 - Two consecutive quotation marks
strName = ""
 - Or by the special identifier String.Empty
strName = String.Empty

The Date Data Type

- Date data type variables can hold the date and time or both
 - You can assign a date literal to a Date variable, as shown here:

```
Dim dtmBirth As Date
```

```
dtmBirth = #5/1/2010#
```

- A date literal is enclosed within # symbols
 - All of the following Date literals are valid:

```
#12/10/2010#
```

```
#8:45:00 PM#
```

```
#10/20/2010 6:30:00 AM#
```

Assigning Text to a Variable

- Tutorial 3-6 provides an example of how the contents of text boxes are assigned to a string variable

' Declare a string variable to hold the full name.

```
Dim strFullName As String
```

' Combine the first and last names

' and copy the result to lblFullName

```
strFullName = txtFirstName.Text & " " & txtLastName.Text
```

```
lblFullName.Text = strFullName
```


Declaring Variables with IntelliSense

- As you enter your program, VB often aids you by offering a list of choices that could be used at that point
- After typing "As" in a variable declaration, VB will offer an alphabetical list of all possible data types
 - Type the first few letters of the data type name
 - IntelliSense box will highlight the matching type
 - Press the Tab key to select highlighted choice
- Or just complete typing the entire data type name

Default Values and Initialization

- When a variable is first created in memory, it is assigned a default value
 - numeric types are given a value of zero
 - Boolean types are given a value of False
 - strings are given a value of Nothing
 - dates default to 12:00:00 AM January 1,1
- Good practice to initialize string variables
 - **Dim strName as String = String.Empty**
 - String with value Nothing causes error if used

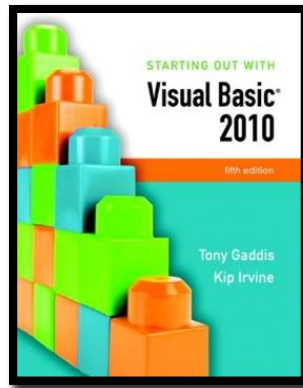
Initialization of Variables

- Can provide a starting or initialization value for any type of variable in a Dim statement
- Usually want to set an initial value unless assigning a value prior to using the variable
- Just append = **value** to the Dim statement where value is the literal to be assigned to the variable

Dim intMonthsPerYear As Integer = 12

Scope and Local Variables

- **Scope** refers to the part of the program where:
 - A variable is visible and
 - May be accessed by program code
- Variables declared within a procedure are called **local variables** and observe these characteristics
 - Scope begins where variable is declared
 - Extends to end of procedure where declared
 - Variable is not visible outside the procedure
- A variable cannot be declared twice in the same procedure



Section 3.3

PERFORMING CALCULATIONS

Visual Basic has powerful arithmetic operators that perform calculations with numeric variables and literals.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Common Arithmetic Operators

- Visual Basic provides operators for the common arithmetic operations:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

Common Arithmetic Operators

- Addition

$\text{dblTotal} = \text{dblPrice} + \text{dblTax}$

- Subtraction

$\text{dblNetPrice} = \text{dblPrice} - \text{dblDiscount}$

- Multiplication

$\text{intArea} = \text{intLength} * \text{intWidth}$

- Division

$\text{dblAverage} = \text{intTotal} / \text{intItems}$

- Exponentiation

$\text{dblCube} = \text{dblSide} ^ 3$

Special Integer Division Operator

- The backslash (\) is used as an integer division operator
- Divides one integer by another
- The result is always an integer, created by discarding any remainder from the division
- If calculating the number of hours in a given number of minutes

intHours = intMinutes \ 60

- With intMinutes equal to 190, this calculation will result in the value 3 assigned to intHours

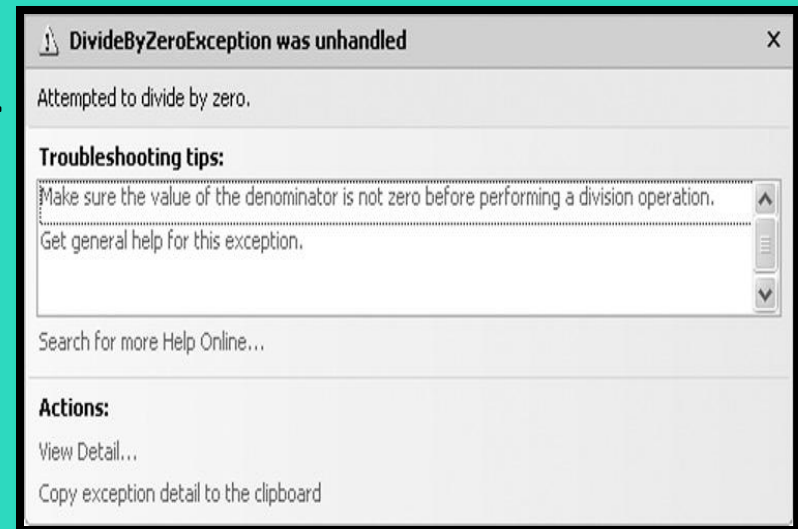
Modulus (MOD) Operator

- This operator can be used in place of the backslash operator to give the remainder of a division operation

intRemainder = 17 MOD 3 ' result is 2

dblRemainder = 17.5 MOD 3 ' result is 2.5

- Use of the \ or MOD operator to perform integer division by zero causes a **DivideByZeroException** runtime error



Retrieving the Current Date/Time

- A series of keywords yields the current date, current time, or both

Description	Keyword	Example
Date & Time	Now	dtmCurrent=Now
Time only	TimeOfDay	dtmCurrTime=TimeOfDay
Date only	Today	dtmCurrDate=Today

- Variables `datCurrent`, `datCurrTime`, and `datCurrDate` must be declared as `Date` data types

Combined Assignment Operators

- Often need to change the value in a variable and assign the result back to that variable
 - For example: **intValue = intValue - 5**
 - Subtracts 5 from the value stored in intValue
- Other examples:
 - **x = x + 4** Adds 4 to x
 - **x = x - 3** Subtracts 3 from x
 - **x = x * 10** Multiplies x by 10
- VB provides for this common need with **combined assignment operators**

Combined Assignment Operators

These special assignment operators provide an easy means to perform these common operations:

Operator	Usage	Equivalent to	Effect
+=	x += 2	x = x + 2	Add to
-=	x -= 5	x = x - 5	Subtract from
*=	x *= 10	x = x * 10	Multiply by
/=	x /= y	x = x / y	Divide by
\=	x \= y	x = x \ y	Int Divide by
&=	name &= last	name = name & last	Concatenate

Arithmetic Operator Precedence

- Operator precedence tells us the order in which operations are performed
- From highest to lowest precedence:
 - Exponentiation (^)
 - Multiplicative (* and /)
 - Integer Division (\)
 - Modulus (**MOD**)
 - Additive (+ and -)
- Where precedence is the same, operations occur from left to right

Operator Precedence Examples

The result is very different when the divide by 2 operation is moved from the end of the calculation to the middle.

$$\begin{array}{r} 6 * 2^3 + 4 / 2 \\ \downarrow \\ 6 * 8 + 4 / 2 \\ \downarrow \\ 48 + 4 / 2 \\ \downarrow \\ 48 + 2 \\ \downarrow \\ 50 \end{array}$$

$$\begin{array}{r} 6 / 2 * 2^3 + 4 \\ \downarrow \\ 6 / 2 * 8 + 4 \\ \downarrow \\ 3 * 8 + 4 \\ \downarrow \\ 24 + 4 \\ \downarrow \\ 28 \end{array}$$

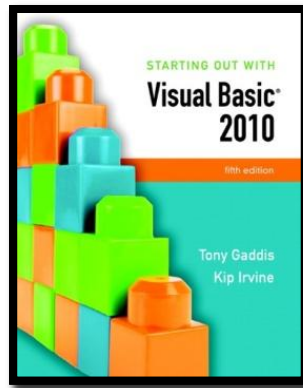
Grouping with Parentheses

- Parentheses () can be used to force selected parts of an expression to be evaluated before others
 - Assume we're computing the average of 3 numbers
 - **dblAvg = int1 + int2 + int3 / 3** ' incorrect
 - int3 / 3 is evaluated first
 - That result is added to int1 and int2
- Use parentheses to control order of operations
 - **dblAvg = (int1 + int2 + int3) / 3** ' correct
 - int1 + int2 + int3 is evaluated first
 - That result is divided by 3
- When in doubt, use parentheses!

Converting Mathematical Expressions to Programming Statements

- In algebra, the mathematical expression $2xy$ describes the value 2 times x times y .
- Visual Basic requires an operator for any mathematical operation.

Mathematical Expression	Operation	Visual Basic Equivalent
$6B$	6 times B	$6 * B$
$(3)(12)$	3 times 12	$3 * 12$
$4xy$	4 times x times y	$4 * x * y$



Section 3.4

MIXING DIFFERENT DATA TYPES

When you assign a value of one data type to a variable of another data type, Visual Basic attempts to convert the value being assigned to the data type of the receiving variable.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Implicit Type Conversions

- A value of one data type can be assigned to a variable of a different type
 - An implicit type conversion is an attempt to convert to the receiving variable's data type
- A widening conversion suffers no loss of data
 - Converting an integer to a double
 - **Dim dblVal As Double = 5**
- A narrowing conversion may lose data
 - Converting a decimal to an integer
 - **Dim intNum As Integer = 12.2 ' intNum becomes 12**

Option Strict

- Option Strict is a VB configuration setting
- Only widening conversions are allowed when Option Strict is set to On
 - An integer can be assigned to a decimal
 - A decimal cannot be assigned to an integer
 - A single can be assigned to a double
 - A double cannot be assigned to a single
- Option Strict On is recommended to help catch errors

Type Conversion Runtime Errors

- Consider the statement
Dim intCount As Integer = "abc123"
- This is a narrowing conversion
- With Option Strict On, statement will not compile
- With Option Strict Off, statement compiles but
 - String "abc123" will not convert to an integer
 - A runtime error called a type mismatch occurs when this statement is executed

Literals

Type	Description	Example
Boolean Keywords	True and False	True
Byte	Decimal digits between 0 and 255	200
Char	Character surrounded by double quotes followed by lowercase C	"A"c
Date	Date and/or time representation enclosed in #	#4/17/10 #
Decimal	Digits with decimal point followed by D or @	+32.0D
Double	Digits with decimal point followed by optional R	3.5R
Integer	Decimal digits followed by optional letter I	-3054I
Long	Decimal digits followed by the letter L	40000L
Short	Decimal digits followed by the letter S	12345S
Single	Digits with decimal point followed by letter F or !	26.4F
String	Characters surrounded by double quotes	"ABC123"

Named Constants

- Programs often need to use given values
 - For example: **dblTotal *= 1.06**
 - Adds 6% sales tax to an order total
- Two problems with this approach
 - The reason for multiplying `dblTotal` by `1.06` isn't always obvious
 - If sales tax rate changes, must find and change every occurrence of `.06` or `1.06`
- Use of **named constants** resolves both these issues

Named Constants

- Can declare a variable whose value is set at declaration and cannot be changed later:

Const dblSALES_TAX_RATE As Double = 1.06

- Looks like a normal declaration except:
 - Const used instead of Dim
 - An initialization value is required
 - By convention, entire name capitalized with underscore characters to separate words
- The objective of our code is now clearer

Const dblSALES_TAX_RATE As Double = 1.06

dblTotal *= dblSALES_TAX_RATE

Explicit Type Conversions

- A function performs some predetermined operation and provides a single output



- VB provides a set of functions that permit narrowing conversions with Option Strict On
- These functions will accept a constant, variable name, or arithmetic expression
- The function returns the converted value

Explicit Type Conversions

- The following narrowing conversions require an explicit type conversion
 - Double to Single
 - Single to Integer
 - Long to Integer
- Boolean, Date, Object, String, and numeric types represent different sorts of values and require conversion functions as well

Explicit Type Conversion Examples

- Rounding can be done with the `CInt` function
 - `intCount = CInt(12.4)` ' intCount value is 12**
 - `intCount = CInt(12.5)` ' intCount value is 13**
- `CStr` converts an integer value to a string
 - `Dim strText As String = CStr(26)`**
- `CDec` converts a string to a double
 - `Dim dblPay As Double = CDbI("$1,500")`**
- `CDate` converts a string to a date
 - `Dim datHired As Date = CDate("2/14/2012")`**

Commonly Used Conversion Functions

Here are some commonly used conversion functions:

Function	Description
Cint (<i>expression</i>)	Converts <i>expression</i> to an integer
Cdbl (<i>expression</i>)	Converts <i>expression</i> to a double
Cdate (<i>expression</i>)	Converts <i>expression</i> to a date
Cdec (<i>expression</i>)	Converts <i>expression</i> to a decimal
Cstr (<i>expression</i>)	Converts <i>expression</i> to a string

A Full List of Conversion Functions

- There are conversion functions for each data type:

CBool (*expression*)

CByte (*expression*)

CChar (*expression*)

CDate (*expression*)

CDbl (*expression*)

CDec (*expression*)

CInt (*expression*)

CLng (*expression*)

CObj (*expression*)

CShort (*expression*)

CSng (*expression*)

CStr (*expression*)

Invalid Conversions

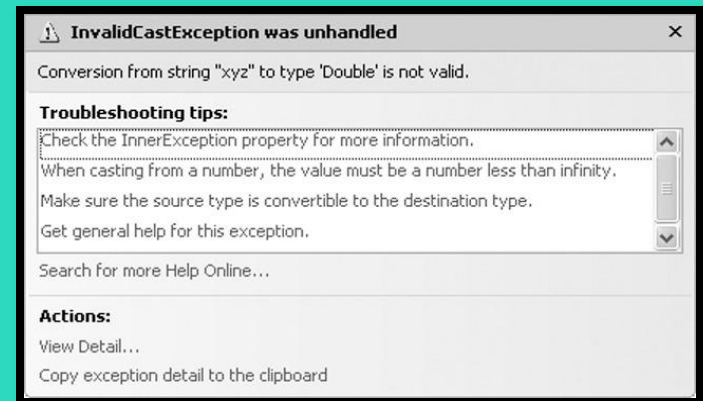
- Conversion functions can fail

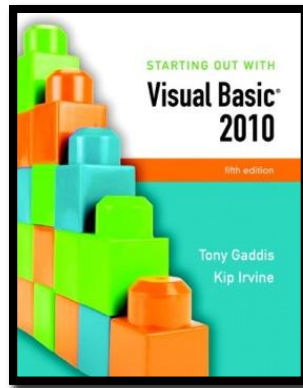
```
Dim dblSalary As Double = CDbI("xyz")
```

```
Dim datHired As Date = CDate("5/35/2011")
```

- String "xyz" can't be converted to a number
- There's no day 35 in the month of May

- Failed conversions cause a runtime error called an invalid cast exception





Section 3.5

FORMATTING NUMBERS AND DATES

Users of computer programs generally like to see numbers and dates displayed in an attractive, easy to read format. Numbers greater than 999, for instance, should usually be displayed with commas and decimal points. The value 123456.78 would normally be displayed as “123,456.78”.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

The ToString Method

- Converts the contents of a variable as a string
- Every VB data type has a ToString method
- Uses the form VariableName.ToString
 - Value in VariableName is converted to a string
- For example:
 - Dim number As Integer = 123**
 - lblNumber.text = number.ToString**
 - Converts integer 123 to string "123"
 - Then assigns the string to the text property of the lblNumber control

ToString Method with Format String

- Can pass a format string to the ToString method
- Indicates how you want to format the string
- For example
 - Dim dblSample As Double**
 - Dim strResult As String**
 - dblSample = 1234.5**
 - strResult = dblSample.ToString("c")**
- The value "c" is a format string
- Converts 1234.5 to currency format \$1,234.50

Types of Format Strings

Format String	Description
N or n	Number format includes commas and displays 2 digits to the right of the decimal
F or f	Fixed point format 2 digits to the right of the decimal but no commas
E or e	Exponential format displays values in scientific notation with a single digit to the left of the decimal point. The exponent is marked by the letter e, and the exponent has a leading + or - sign.
C or c	Currency format includes dollar sign, commas, and 2 digits to the right of the decimal
P or p	Percent format multiplies number by 100 and displays with a trailing space and percent sign

Specifying Decimal Precision

- Can add an integer to the format string to indicate number of digits to display after the decimal point
- Rounding occurs when displaying fewer decimal positions than the number contains as in the 2nd line

Number Value	Format String	ToString() Value
12.3	n3	12.300
12.348	n2	12.35
1234567.1	n	1,234,567.10
123456.0	f2	123456.00
123456.0	e3	1.235e+005
.234	p	23.40%
-1234567.8	c	(\$1,234,567.80)

Specifying Integer Leading Zeros

- Can specify a minimum width when displaying an integer value
- Leading zeros are inserted to meet the minimum width if needed

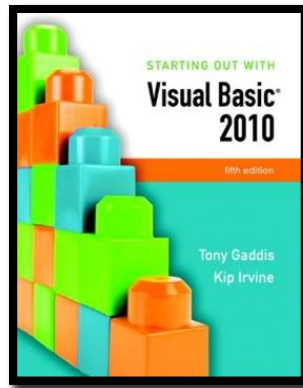
Number Value	Format String	ToString() Value
23	D	23
23	D4	0023
1	D2	01

Formatting Dates and Times

- The ToString method can format a Date or DateTime value in a variety of ways
- If the date is 8/10/2010 and the time is 3:22 PM

Format String	Description	ToString() Value
d	Short Date	"8/10/2010"
D	Long Date	"Tuesday, August 10, 2010"
t	Short Time	"3:22 PM"
T	Long Time	"3:22:00 PM"
F	Long Date & Time	"Tuesday August 10, 2010 3:22:00 PM"

- Tutorial 3-8 provides an opportunity to work with number formatting concepts



Section 3.6

CLASS-LEVEL VARIABLES

Class-level variables are accessible to all procedures in a class.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Class-Level Variables

- A variable declared inside a class but outside any procedure is a **class-level variable**
 - Scope is throughout all procedures of the class
- Take care when using class-level variables:
 - Tracking down logic errors can be time consuming because many statements can access the variable
 - Make sure not to upset the accuracy of variables that are used in multiple procedures
 - Because all statement can access the variables, you must be aware of every statement that has access

Class-Level Constants

- A class-level constant is a named constant declared with the **Const** keyword, at the class level
- Class-level constants cannot be changed during runtime
 - eliminates many of the potential hazards that are associated with the use of class-level variables
 - generally more acceptable to use than class-level variables

Class-Level Declarations

Public Class Form1

' Begin after class declaration.

' Declare a class-level constant.

```
Dim Const intValue As Integer = 0
```

' Declare a class-level variable.

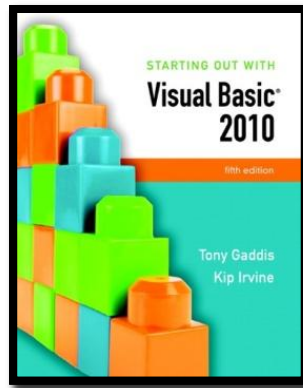
```
Dim intValue As Integer
```

' End before procedure declarations.

```
Private Sub Procedure()
```

```
End Sub
```

```
End Class
```

Section 3.7

EXCEPTION HANDLING

A well-engineered program should report errors and try to continue. Or, it should explain why it cannot continue, and then shut down. In this section, you learn how to recover gracefully from errors, using a technique known as exception handling.

Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Runtime Errors

- We've shown two possible runtime errors
 - DivideByZeroException
 - InvalidCastException
 - There are many others
- Runtime errors occur for many reasons
- A runtime error results when:
 - Visual Basic throws an exception
 - And it is an unhandled exception
- Exception handling allows a program to fail gracefully and recover if possible

Handling Exceptions

- Visual Basic provides an exception handler
- The Try-Catch statement:
 - Try**
 - ' Try block statements...
 - Catch**
 - ' Catch block statements...
 - End Try**
- The **try block** contains program statements that might throw an exception
- The **catch block** contains statements to execute if an exception is thrown

Exception Handling Example

Try

' Get the user's input and convert it to a Decimal.

`decSalary = CDec(txtSalary.Text)`

' Display the user's salary.

`MessageBox.Show("Your salary is " & decSalary.ToString("c"))`

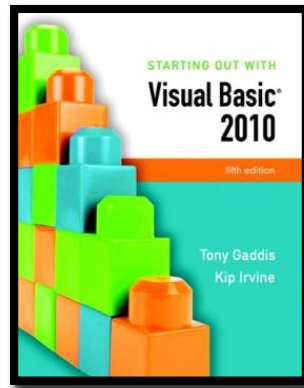
Catch

' Display an error message.

`MessageBox.Show("Please try again, and enter a number.")`

End Try

- If **CDec** throws a cast exception, the try block catches it, jumps to and executes the catch block which displays the error message



Section 3.8

GROUP BOXES

The GroupBox control is a container that is used to group other controls together.

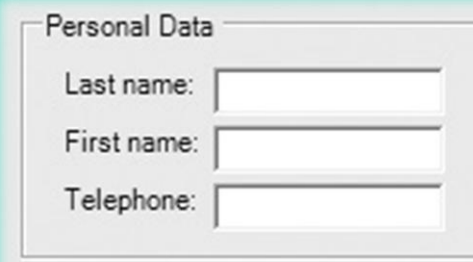
Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

The GroupBox Control

- A **GroupBox** creates a grouping of controls
 - Controls are enclosed in a box with a title
 - It's apparent the controls within the GroupBox are related in some way
 - Controls in a GroupBox have their own tab order
 - Moving a GroupBox moves its controls with it
 - Removing a GroupBox also removes all controls within it



Personal Data

Last name:

First name:

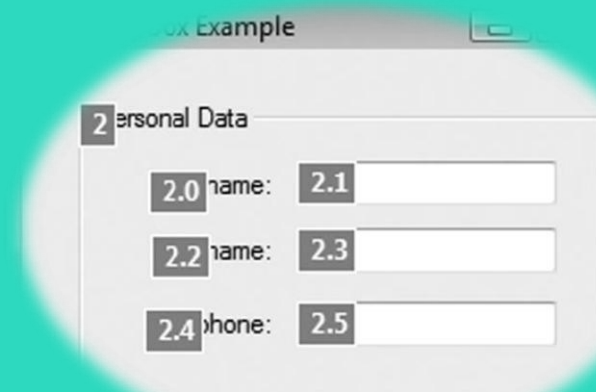
Telephone:

Placing Controls Within a Group Box

- Must create the GroupBox first
- Then select the GroupBox control and
 - Double-click the tool from the ToolBox to place the control in the group
 - or**
 - Click and drag the control from the ToolBox to the GroupBox
- To move an existing control to a GroupBox
 - Select the control and cut it from the form
 - Select the group and paste the control into it

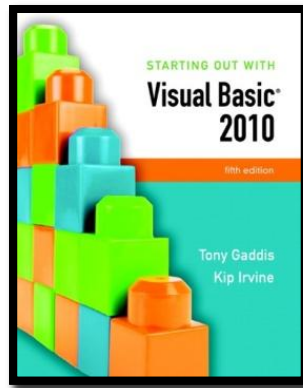
GroupBox Tab Order

- A GroupBox has its own place in form tab order
- Once the tab order reaches the GroupBox
 - Must tab through all controls in the GroupBox before tabbing to controls outside GroupBox
 - Tab order of controls inside the GroupBox can be assigned in any order
- The GroupBox to the right is 2nd in the form tab order
- Tab order of controls in the GroupBox is 2.1, 2.3, & 2.5



Selecting Multiple Controls

- Multiple controls can be selected and then acted upon as a group
 - Click and drag over the desired controls
 - Any control partially or completely within the selection box will be selected
 - Or hold the Ctrl key while clicking the controls
- Once selected, a group of controls may
 - Be moved together as a group
 - Be deleted in a single step
 - Have their properties set in a single step



Section 3.9

THE LOAD EVENT

When an application's form loads into memory, an event known as the Load event takes place. You can write an event handler for the Load event, and that handler will execute just before the form is displayed.

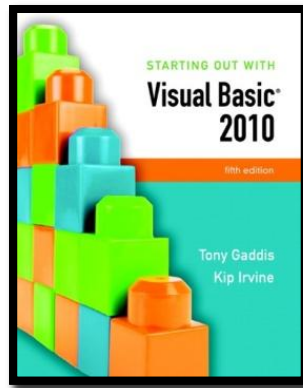
Addison Wesley
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

Load Event Handler

- Every form has a Load event
 - Executes when the form is first displayed
 - Double-click in any empty space on the form
 - The code window will appear
 - Place the code to be executed between the Private Sub and End Sub lines of the event handler
- ```
Private Sub Form1_Load(...) Handles MyBase.Load
 MessageBox.Show("Prepare to see the form!")
End Sub
```



## Section 3.10

# FOCUS ON PROGRAM DESIGN AND PROBLEM SOLVING: BUILDING THE *ROOM CHARGE CALCULATOR APPLICATION*

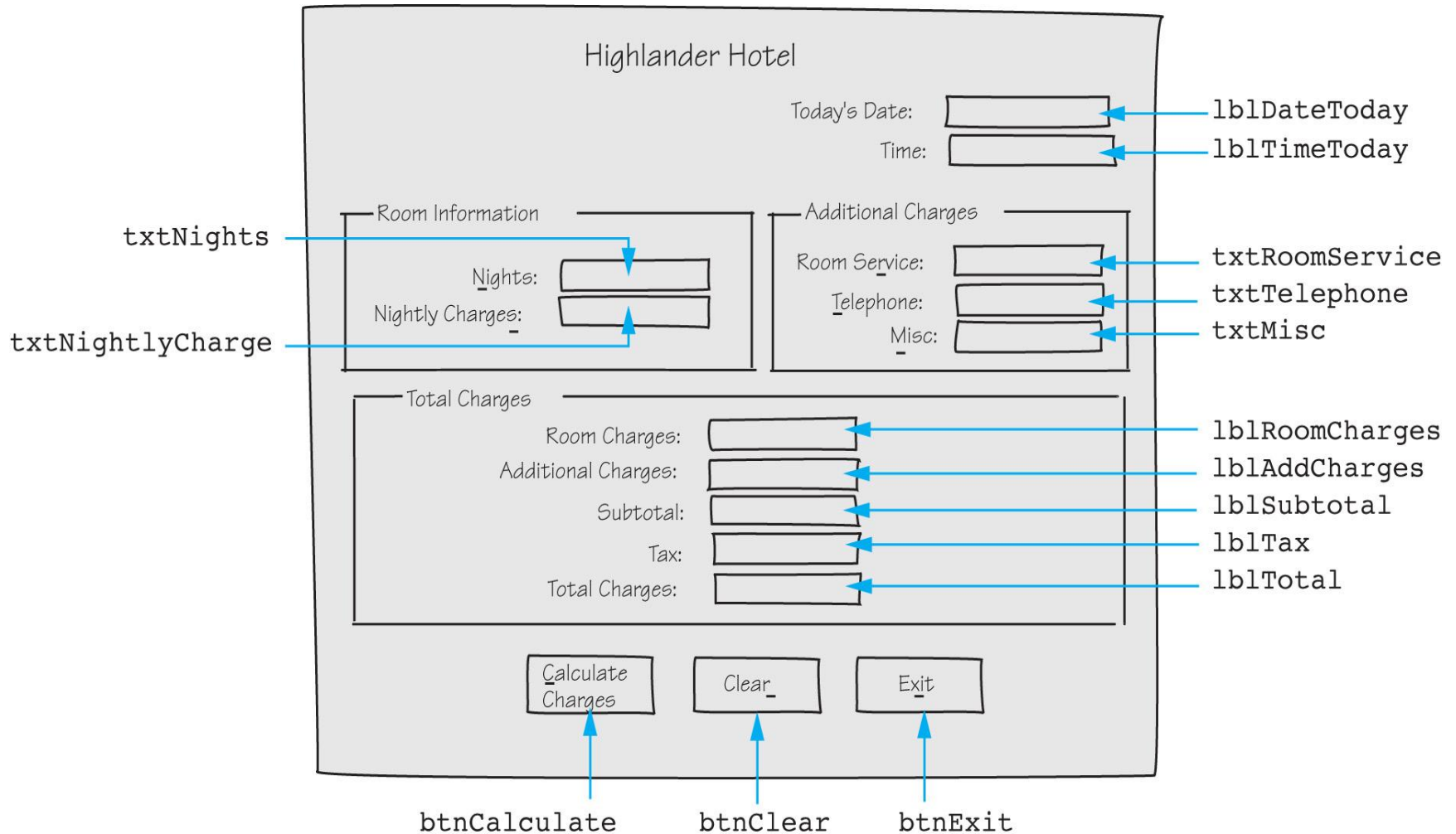
The Room Charge Calculator Application applies the various concepts discussed in this chapter.

**Addison Wesley**  
is an imprint of

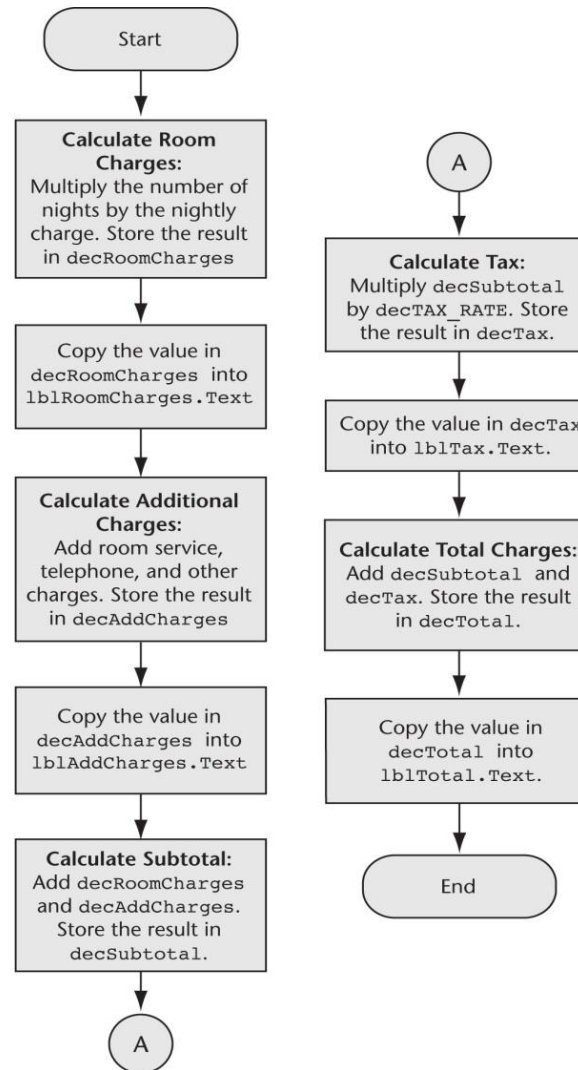


© 2011 Pearson Addison-Wesley. All rights reserved.

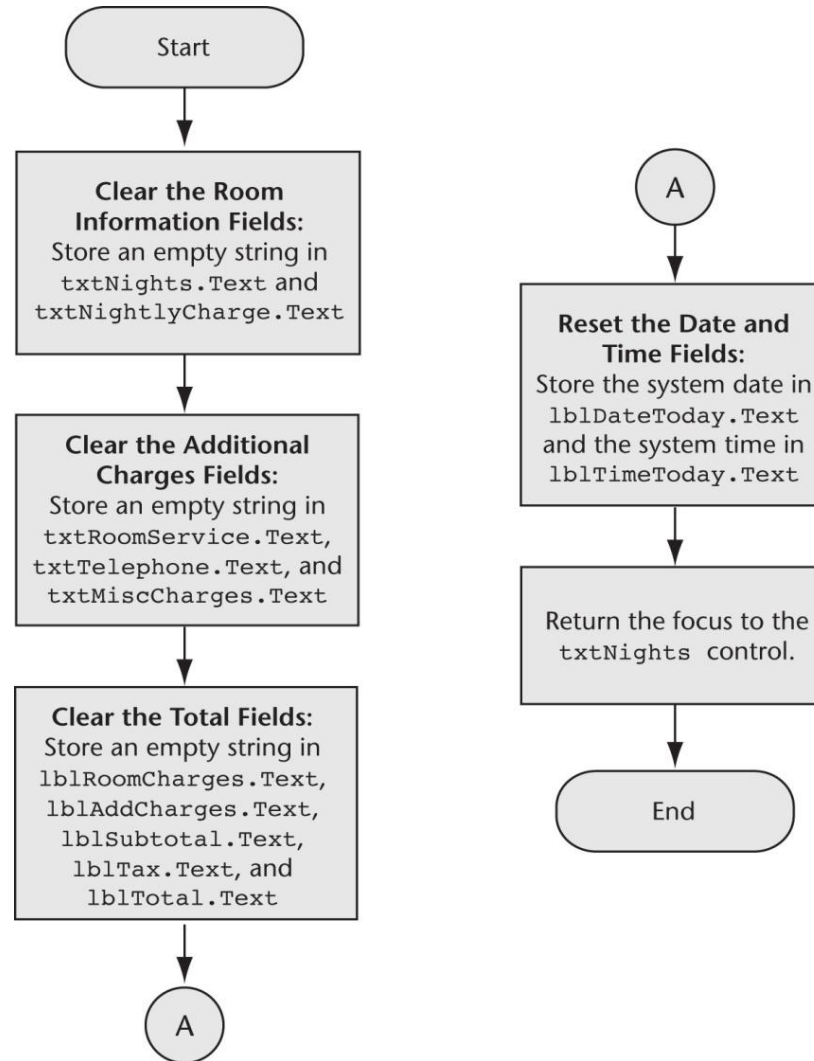
# The Room Charge Calculator



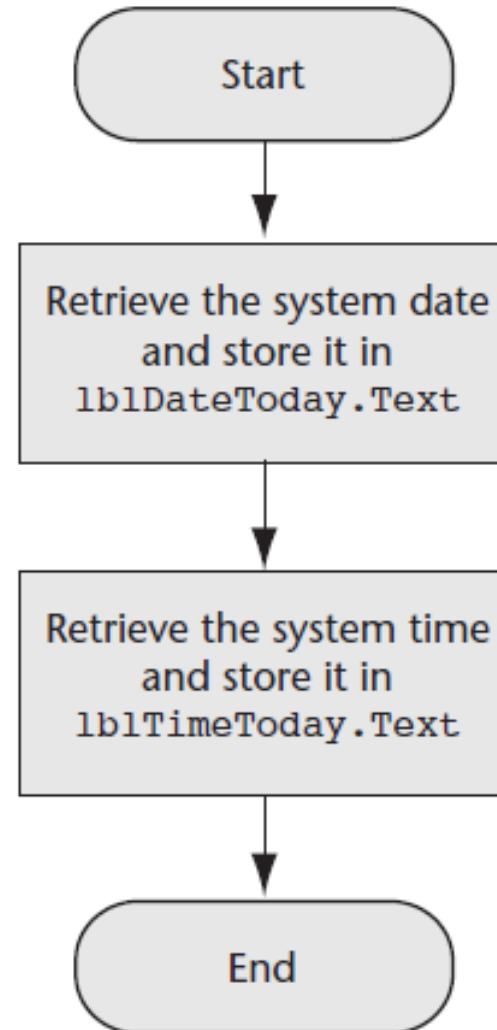
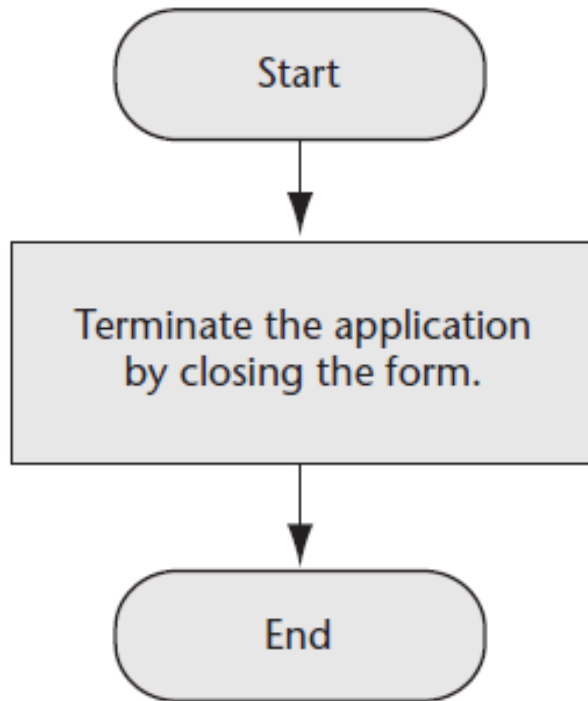
# The btnCalculate Click Event



# The btnClear Click Event



# The btnExit Click Event & The Form1 Load Event





# The Completed Form

**Room Charge Calculator**

## Highlander Hotel

Today's Date: **Wednesday, February 10, 2010**  
Time: **8:59:13 PM**

| Room Information |       | Additional Charges |       |
|------------------|-------|--------------------|-------|
| Nights:          | 5     | Room Service:      | 10.50 |
| Nightly Charge:  | 80.50 | Telephone:         | 5.25  |
|                  |       | Misc:              | 3.00  |

| Total Charges       |          |
|---------------------|----------|
| Room Charges:       | \$402.50 |
| Additional Charges: | \$18.75  |
| Subtotal:           | \$421.25 |
| Tax:                | \$33.70  |
| Total Charges:      | \$454.95 |

Buttons: Calculate Charges, Clear, Exit

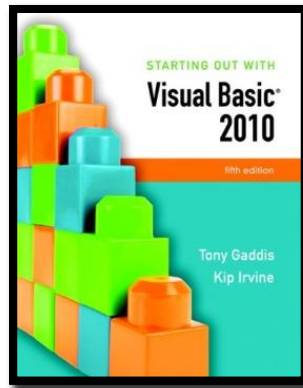
# Changing Colors with Code (Optional Topic)

- You can change color properties with code
  - The following code sets the label's background color to black and foreground color to yellow:

```
lblMessage.BackColor = Color.Black
lblMessage.ForeColor = Color.Yellow
```

- And the following code returns the background and foreground to the default colors:

```
lblMessage.BackColor = SystemColors.Control
lblMessage.ForeColor = SystemColors.ControlText
```



## Section 3.11

# MORE ABOUT DEBUGGING: LOCATING LOGIC ERRORS

Visual Studio allows you to pause a program, and then execute statements one at a time. After each statement executes, you may examine variable contents and property values.

**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

# Debugging Problem

- The program runs but does not work correctly (has one or more logic errors)
- Running the program with various inputs has not isolated where those logic errors lie
- What can be done?

# Visual Basic Debugging Aids

- You can set **breakpoints**
  - A line or lines you select in your source code
  - When execution reaches this line, it pauses
  - You may then examine the values in variables and certain control properties
  - You may also **single-step** through the program which executes one statement at a time
- This allows you to see and examine:
  - What is happening one statement at a time
  - Where it is happening
  - What the various data values are (**Watches**)

# Visual Basic Debugging Aids

- Tutorial 3-13 demonstrates how to:
  - Set breakpoints
  - Examine the values of variables and control properties
  - Use the Autos, Immediate, Locals, and Watch windows

# Debugging Commands in the Toolbar

- Visual Studio provides a toolbar for debugging commands

