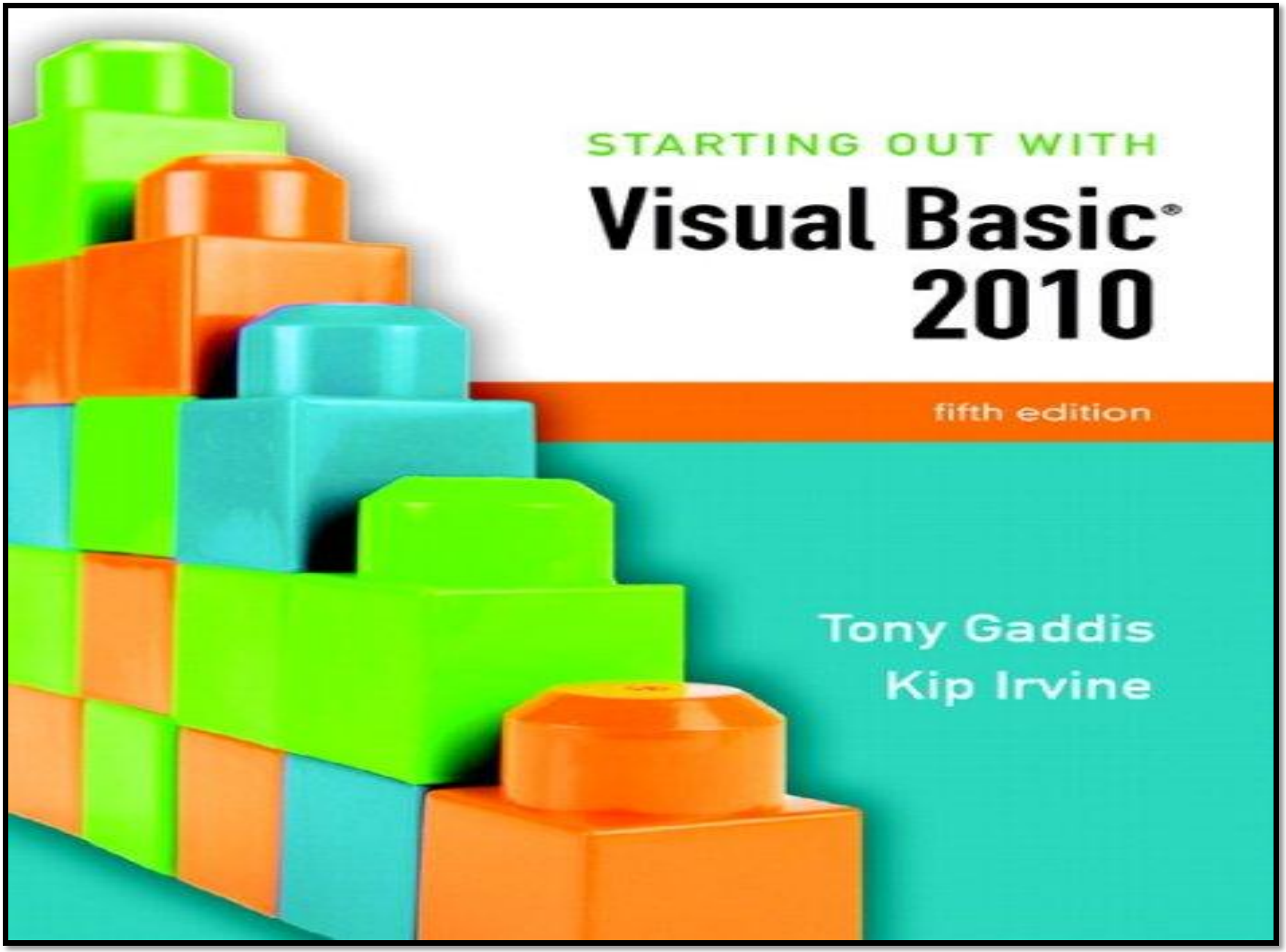
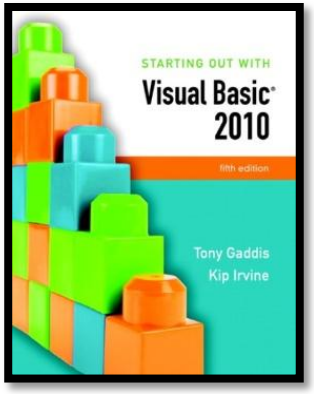

STARTING OUT WITH
Visual Basic®
2010
fifth edition
Tony Gaddis
Kip Irvine



Chapter 7

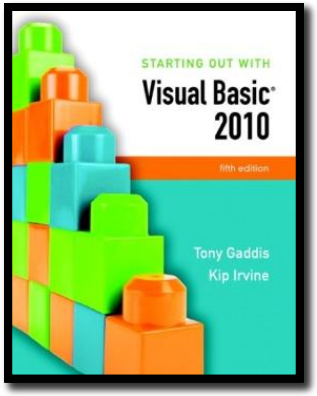
Multiple Forms, Modules, and Menus

Addison Wesley
is an imprint of

PEARSON

Introduction

- This chapter demonstrates how to:
  - Add multiple forms to a project
  - Create a module to hold procedures and functions
  - Create a menu system with commands and submenus
  - Create context menus that appear when the user right-clicks on an item



Section 7.1

MULTIPLE FORMS

Visual Basic projects can have multiple forms. The startup form is the form that is displayed when the project executes. Other forms in a project are displayed by programming statements.

Addison Wesley
is an imprint of

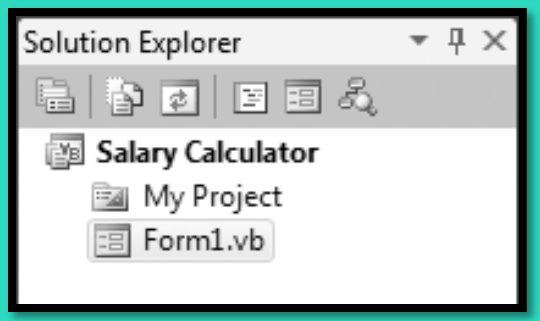
PEARSON

Windows Forms Applications

- Windows Forms applications are not limited to only a single form
- You may create multiple forms
  - To use as dialog boxes
  - Display error messages
  - And so on
- Windows Forms applications typically have one form called the **startup form**
  - Automatically displayed when the application starts
  - Assigned to the first form by default
  - Can be assigned to any form in the project

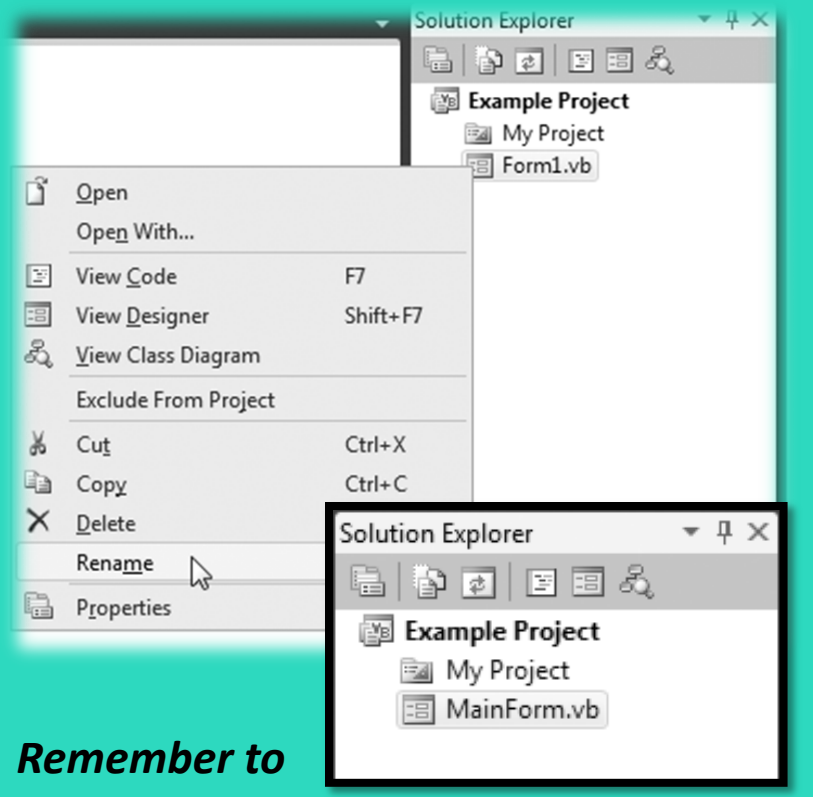
Form Files and Form Names

- Each form has a **Name** property
  - Set to **Form1** by default
- Each form also has a file name
  - Stores the code associated with the form
  - Viewed in the ***Code*** window
  - Has the same name as the form
  - Followed by the ***.vb*** extension
  - Shown in the ***Solution Explorer*** window



Renaming an Existing Form File

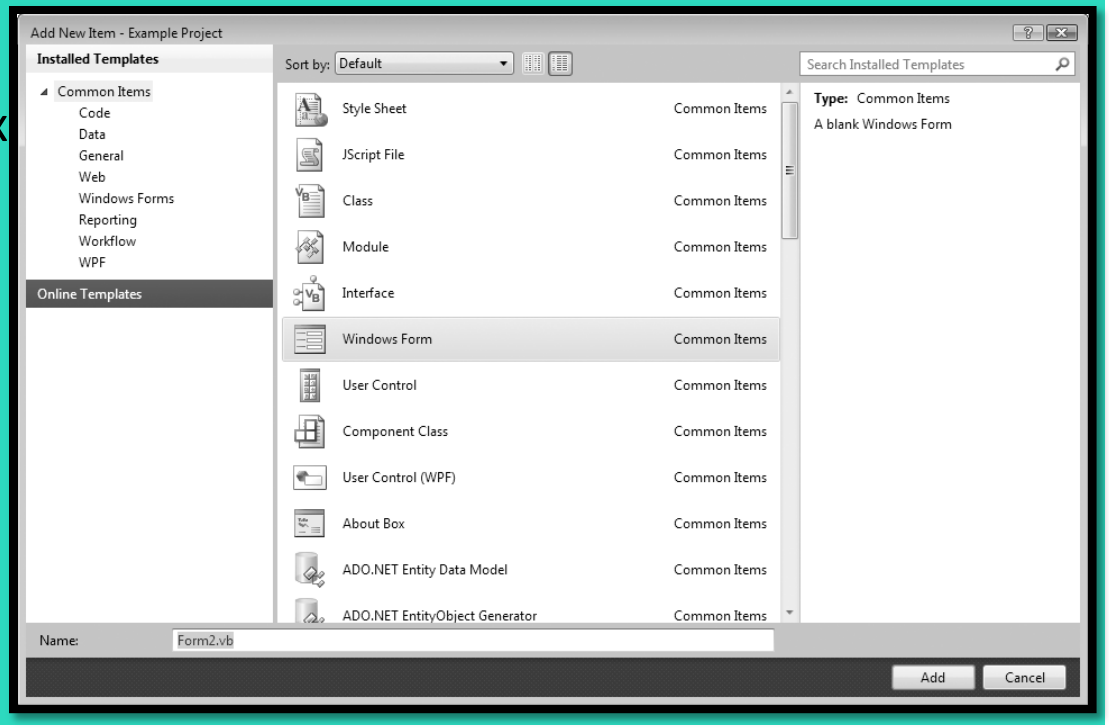
- Always use the ***Solution Explorer*** window to change the file name and the form's **Name** property will change automatically
- To rename a form file:
  - **Right-click** file name in ***Solution Explorer***
  - Select ***Rename*** from the menu
  - Type the new name for the form
  - Be sure to keep the ***.vb*** extension

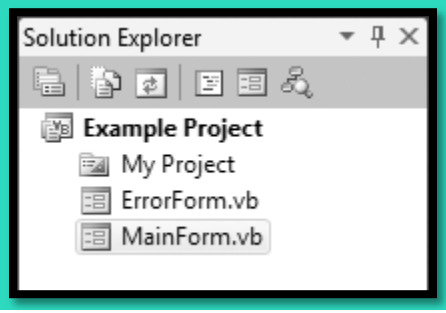


Remember to give each form a meaningful name

Adding a New Form to a Project

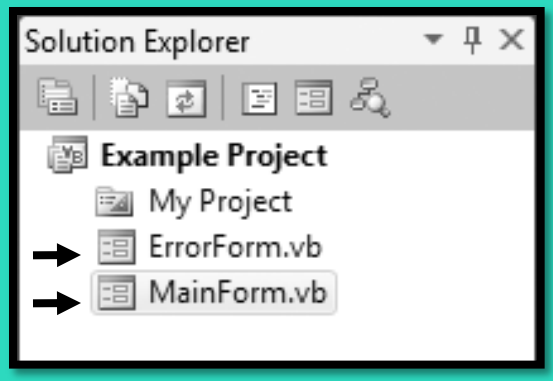
- To add a new form to a project:
  - Click ***Project*** on the Visual Studio menu bar, and then select ***Add Windows Form . . .*** The ***Add New Item*** window appears
  - Enter the new Name in the ***Name*** text box
  - Click the ***Add*** button
- A new blank form is added to your project





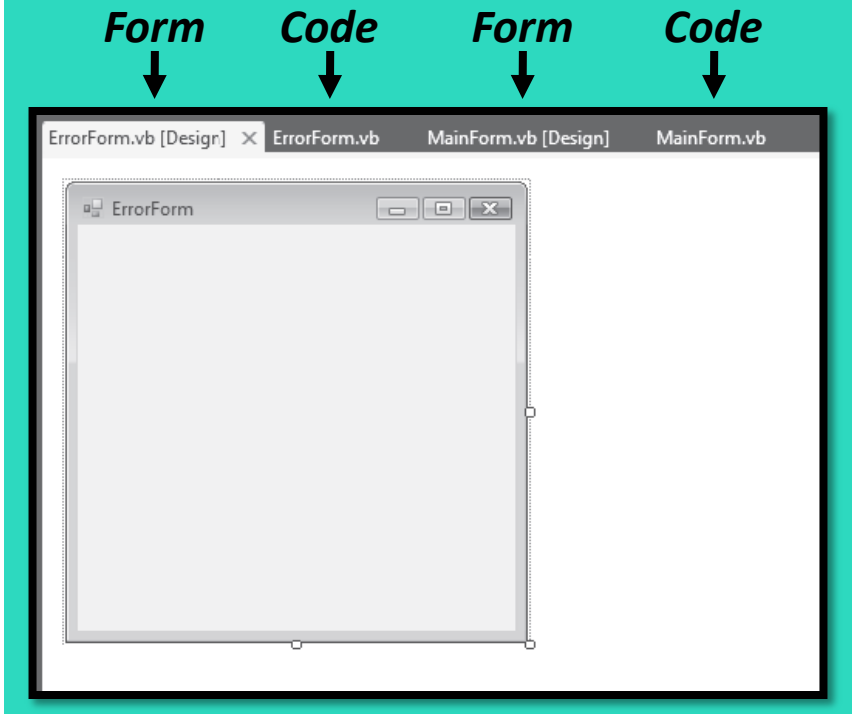
Switching between Forms and Form Code

- To switch to another form:
  - **Double-click** the form's entry in the ***Solution Explorer*** *window*



Switching only works at design time

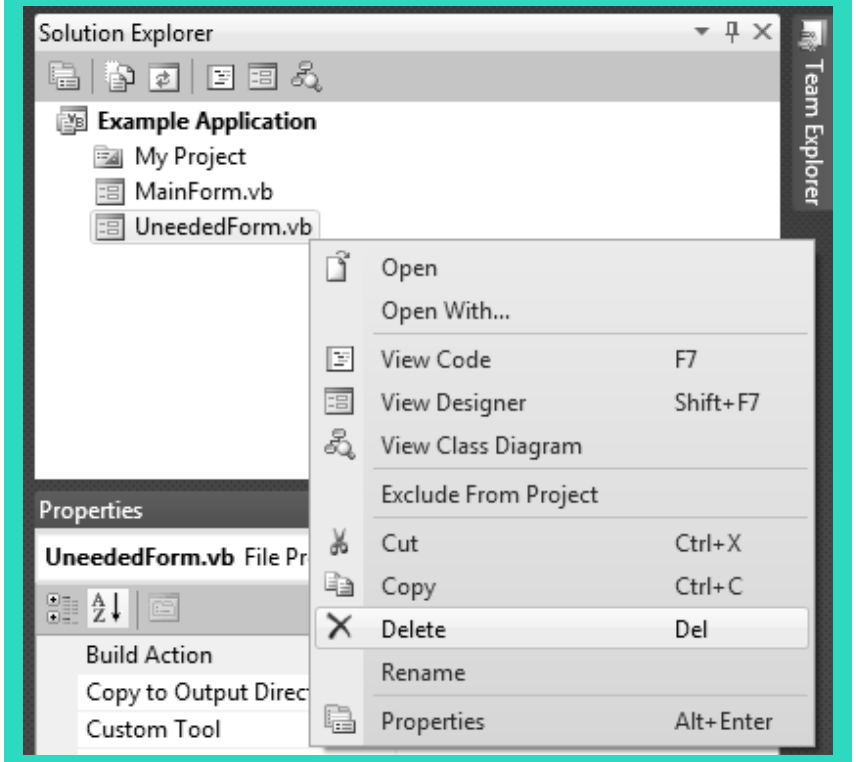
- To switch between forms or code:
  - Use the tabs along the top of the ***Designer*** window



Removing a Form

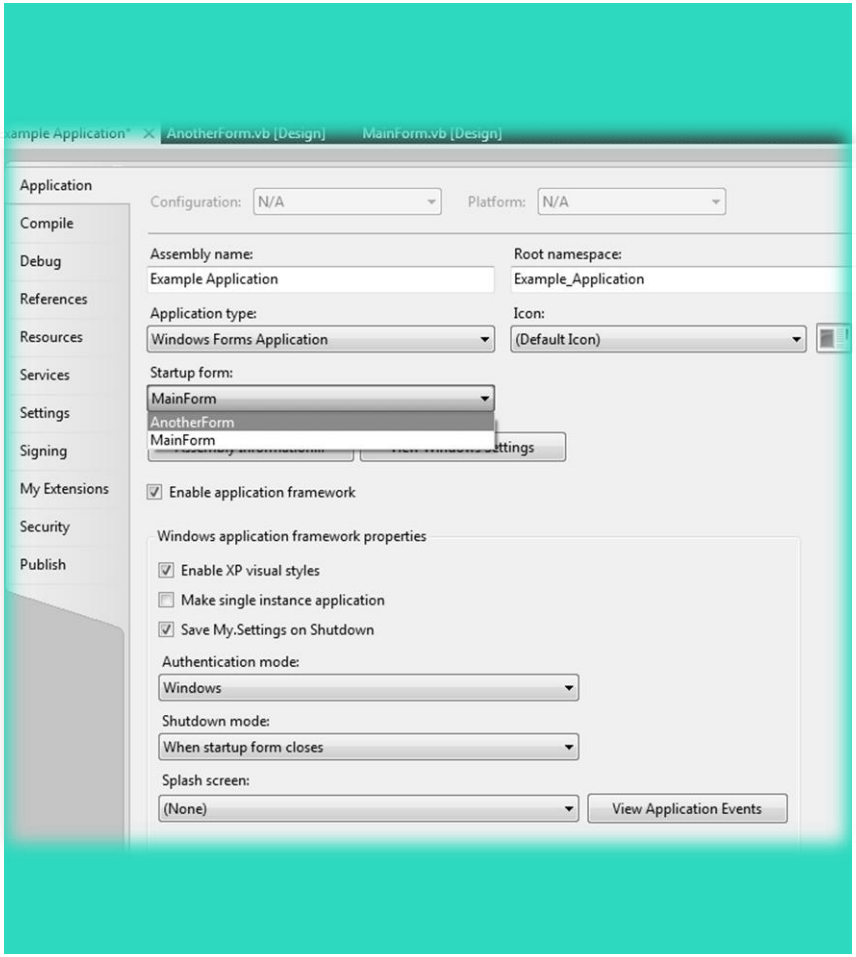
- To remove a form from a project and delete its file from the disk:
  - **Right-click** the form's entry in the ***Solution Explorer*** *window*
  - On the pop-up menu, click ***Delete***
- To remove a form from a project but leave its file on disk*:
  - **Right-click** the form's entry in the ***Solution Explorer*** *window*
  - On the pop-up menu, click ***Exclude From Project***

Not available in Visual Basic Express



Designating the Startup Form

- To make another form the startup form:
  - **Right-click** the project name in the ***Solution Explorer*** *window*
  - On the pop-up menu, click ***Properties***, the **properties page** appears
  - Select the ***Application*** *tab*
  - Click the **down arrow** in the ***Startup Form drop-down list***
  - Select a form from the list of available forms

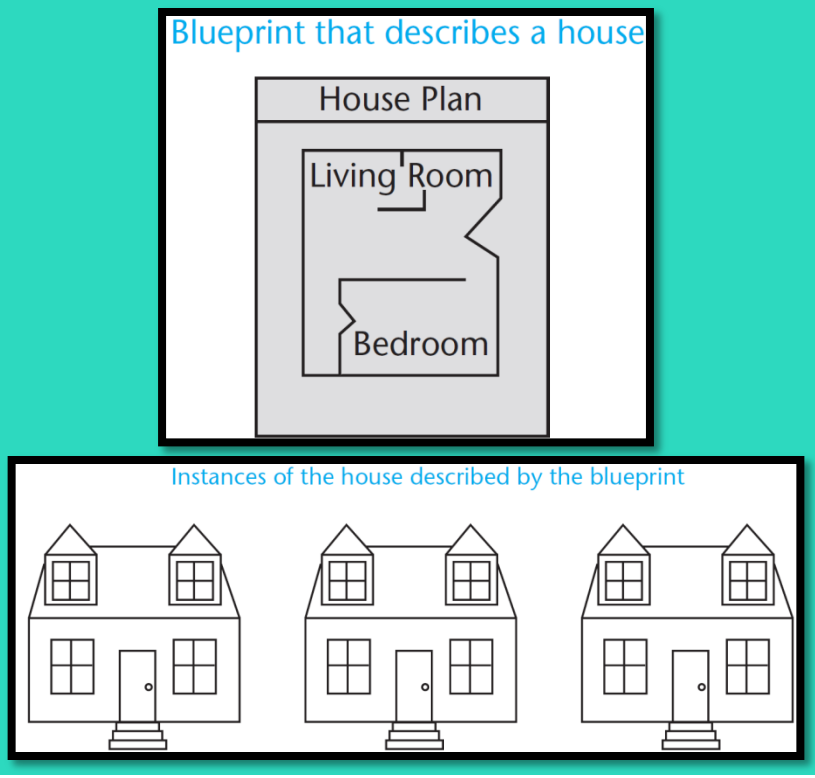


Creating an Instance of a Form

- The form design is a **class**
  - It's only a design or description of a form
  - Think of it like a blueprint
    - A blueprint is a detailed description of a house
    - A blueprint is ***not*** a house
- The form design can be used to create instances of the form
  - Like building a house from the blueprint
- To display a form, we must first create an instance of the form

Public Class *FormName*

End Class



Displaying a Form

- The first step is to create an instance of the form with the **Dim** statement
  - Here is the general format:

Dim *ObjectVariable As New ClassName*

- ***ObjectVariable*** is the name of an object variable that references an instance of the form
- An **object variable**
  - Holds the memory address of an object
  - Allows you to work with the object
- ***ClassName*** is the form's class name

- The following statement creates an instance of the **ErrorForm** form in memory:

Dim frmError As New ErrorForm

- **frmError** variable references the **ErrorForm** object
- Statement does not cause the form to be displayed on the screen
- To display the form on the screen:
  - Use the object variable to invoke one of the form's methods

The prefix frm is used to indicate that the variable references a form

The **ShowDialog** and **Show** Methods

- If a **modal** form is displayed:
  - No other form in the application can receive the focus until the form is closed
- The **ShowDialog** method causes a form to be displayed as a **modal** form
  - Here is the general format:

ObjectVariable.ShowDialog()

- For example:

```
Dim frmError As New ErrorForm
frmError.ShowDialog()
```

- If a **modeless** form is displayed:
  - The user is allowed to switch focus to another form while it is displayed
- The **Show** method causes a form to be displayed as a **modeless** form
  - Here is the general format:

ObjectVariable.Show()

- For example:

```
Dim frmError As New ErrorForm
frmError.Show()
```

Closing a Form with the **Close** Method

- The **Close method** closes a form and removes its visual part from memory
- A form closes itself using the keyword **Me**
- For example:

Me.Close()

- Causes the current instance of the form to call its own **Close** method, thus closing the form

The word Me in Visual Basic is a special variable that references the currently executing object

The **Hide** Method

- The **Hide method**
  - Makes a form or control invisible
  - Does not remove it from memory
  - Similar to setting the **Visible** property to **False**
- A form uses the **Me** keyword to call its own **Hide** method
- For example:

Me.Hide()

- To redisplay a hidden form:
  - Use the **ShowDialog** or **Show** methods
- Tutorial 7-1 creates a simple application that has two forms

More on Modal and Modeless Forms

- When a procedure calls the **ShowDialog** method
  - Display of a **modal** form causes execution of calling statements to halt until form is closed

statement
statement
frmMessage.ShowDialog()
statement ← Halt!
statement ← Halt!
statement ← Halt!

- When a procedure calls the **Show** method
  - Display of a **modeless** form allows execution to continue uninterrupted

statement
statement
frmMessage.Show()
statement ← Go!
statement ← Go!
statement ← Go!

- Tutorial 7-2 demonstrates this difference between modal and modeless forms

The **Load** Event

- The **Load** event is triggered just before the form is initially displayed
- Any code needed to prepare the form prior to display should be in the Load event
- If some controls should not be visible initially, set their Visible property in the Load event
- Double click on a blank area of the form to set up a Load event as shown below

```
Private Sub MainForm_Load(…) Handles MyBase.Load

End Sub
```

The **Activated** Event

- The **Activated** event occurs when the user switches to the form from another form or application
- To create an **Activated event handler**, follow these steps:
  1. Click the class drop-down list, which appears at the top left of the *Code* window
  2. On the drop-down list, select (*FormName Events*), where *FormName* is the name of the form
  3. Click the method drop-down list, which appears at the top right of the *Code* window, and select ***Activated***
- After completing these steps, a code template for the Activated event handler is created in the *Code* window

The **FormClosing** Event

- The **FormClosing** event is triggered as the form is being closed, but before it has closed
- The FormClosing event can be used to ask the user if they really want the form closed
- To create an **FormClosing event handler**, follow these steps:
  1. Click the class drop-down list, which appears at the top left of the *Code* window
  2. On the drop-down list, select (*FormName Events*), where *FormName* is the name of the form
  3. Click the method drop-down list, which appears at the top right of the *Code* window, and select ***FormClosing***
- After completing these steps, a code template for the FormClosing event handler is created in the *Code* window

The **FormClosed** Event

- The **FormClosed** event occurs after a form has closed.
- Create a **FormClosed event handler** by following these steps:
  1. Click the class drop-down list, which appears at the top left of the *Code* window
  2. On the drop-down list, select (*FormName Events*), where *FormName* is the name of the form
  3. Click the method drop-down list, which appears at the top right of the *Code* window, and select ***FormClosed***
- After completing these steps, a code template for the FormClosed event handler is created in the *Code* window

You cannot prevent a form from closing with the FormClosed event handler. You must use the FormClosing event handler to prevent a form from closing.

Accessing Controls on a Different Form

- Once you have created an instance of a form, you can access controls on that form in code
  - The following code shows how you can
    - Create an instance of a form
    - Assign a value to the form's label control's Text property
    - Display the form in modal style

```
Dim frmGreetings As New GreetingsForm
frmGreetings.lblMessage.Text = "Good day!"
frmGreetings.ShowDialog()
```

- Tutorial 7-3 demonstrates accessing controls on a different form

Class-Level Variables in a Form

- Class-level variables are declared **Private** by the **Dim** statement
- Private variables are not accessible by code in other forms

```
Dim dblTotal As Double          ' Class-level variable
```

- Use the **Public** keyword to make a class-level variable available to methods outside the class

```
Public dblTotal As Double       ' Class-level variable
```

- Explicitly declare class-level variables with the **Private** keyword to make your source code more self-documenting

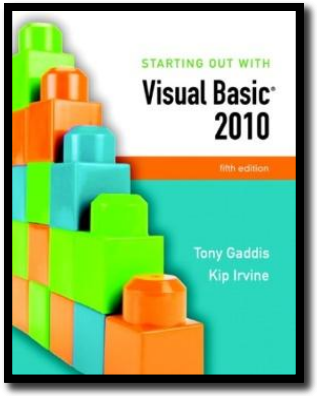
```
Private dblTotal As Double      ' Class-level variable
```

Using **Private** and **Public** Procedures in a Form

- Procedures, by default, are **Public**
- They can be accessed by code outside their form
- To make a procedure invisible outside its own form, declare it to be **Private**
- You should always make the procedures in a form private
  - Unless you specifically want statements outside the form to execute the procedure

Using a Form in More Than One Project

- After a form has been created and saved to a file, it may be used in other projects
- Follow these steps to add an existing form to a project:
  1. With the receiving project open in Visual Studio, click ***Project*** on the menu bar, and then click ***Add Existing Item***
  2. The ***Add Existing Item*** dialog box appears
  3. Locate the form file that you want to add to the project, select it and click the ***Open*** button
- A copy of the form is now added to the project



Section 7.2

MODULES

A module contains code—declarations and procedures—that are used by other files in a project.

Addison Wesley
is an imprint of

PEARSON

What is a Module?

- A **module** is a Visual Basic file that contains only code
  - General purpose procedures, functions, and declarations of variables and constants
  - Can be accessed by all forms in the same project
  - No event handlers
  - Stored in files that end with the .vb extension
  - Appears in the *Solution Explorer* along with entries for the project's form files

Module Names and Module Files

- A module
  - begins with a **Module** statement
  - ends with an **End Module** statement
- Here is the general format:

```
Module ModuleName
    [Module Contents]
End Module
```

- **ModuleName** is the name of the module
  - Can be any valid identifier
  - That describes its purpose
- Code is stored in a file that is named with the .vb extension
- Normally, the name of the file is the same as the name of the module

Example Module

- The following code shows the contents of a module named RetailMath

```
Module RetailMath
    ' Global constant for the tax rate
    Public Const decTAX_RATE As Decimal = 0.07D

    ' The SalesTax function returns the sales tax on a purchase.
    Public Function SalesTax(ByVal decPurchase As Decimal) As Decimal
        Return decPurchase * decTAX_RATE
    End Function
End Module
```

Adding a Module

- Follow these steps to add a module to a project:
  1. Click ***Project*** on the menu bar and then click ***Add Module***. The ***Add New Item*** windows appears
  2. Change the default name that appears in the ***Name*** text box to the name you wish to give the new module file
  3. Click the ***Add*** button
- A new empty module will be added to your project
- The module is displayed in the *Code* window
- An entry for the module appears in the *Solution Explorer* window

Module-Level Variables

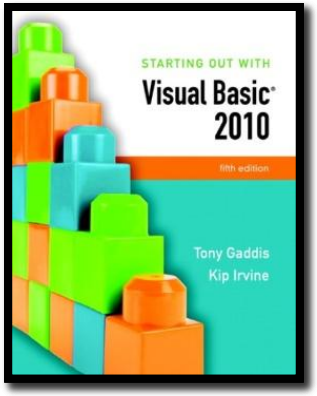
- A **module-level variable** is a variable that is declared inside a module, but not inside a procedure or function
- The same rules about the scope of class-level variables in a form apply to module-level variables in a module
- Variables with **module scope** are declared with **Dim** or **Private**
  - Accessible to any function or procedure in the module
  - Not accessible to statements outside of the module
- A **global variable** is declared with the **Public** keyword
  - Accessible to any statement in the application
  - Some programmers prefix global variables with **g_**

```
Public g_decPurchaseAmount As Decimal          ' Global variable
```

- Tutorial 7-4 examines an application that uses a module

Using a Module in More Than One Project

- It is possible to use more than one module in a project
- Suppose you want to add an existing module to a new project
- Follow these steps to add an existing standard module to a project:
  1. Click ***Project*** on the menu bar, and then click ***Add Existing Item***. The ***Add Existing Item*** dialog box appears
  2. Use the dialog box to locate the module file you want to add to the project. When you locate the file, select it and click the ***Open*** button
- The module is now added to the project



Section 7.3

MENUS

Visual Basic allows you to create a system of drop-down menus for any form in your application. You use the menu designer to create a menu system.

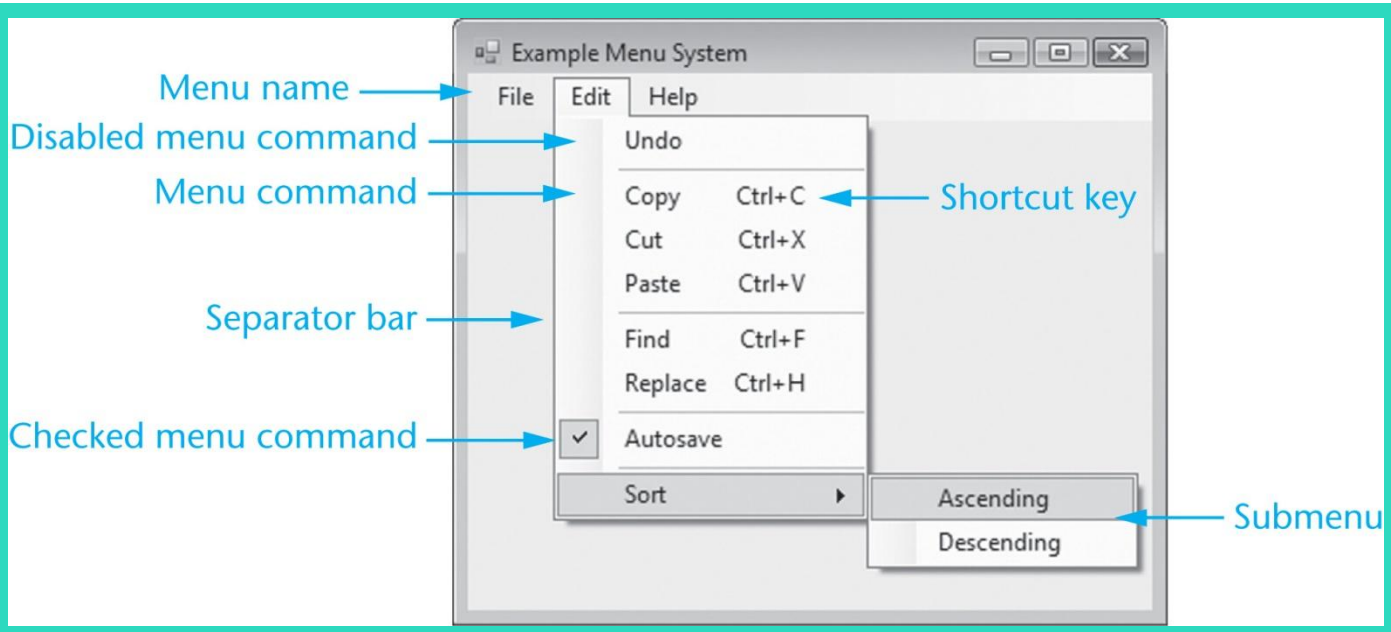
Addison Wesley
is an imprint of

PEARSON

Menu Systems

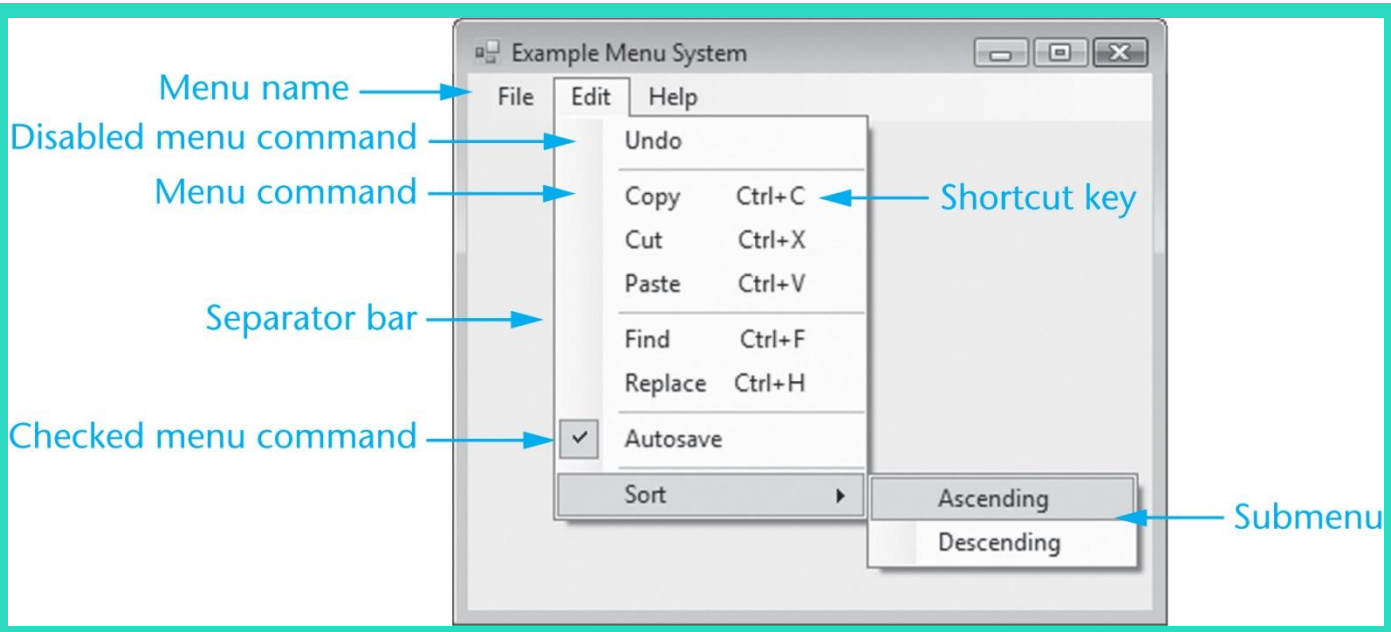
- A **menu system** is a collection of commands organized in one or more drop-down menus
  - commonly used when an application has several options for the user to choose from
- The **menu designer** allows you to visually create a custom menu system
  - for any form in an application

Components of a Menu System



- Each drop-down menu has a **menu name**
- Each drop-down menu has a list of actions or **menu commands** that can be performed
- Some commands may lead to a **submenu**

Components of a Menu System

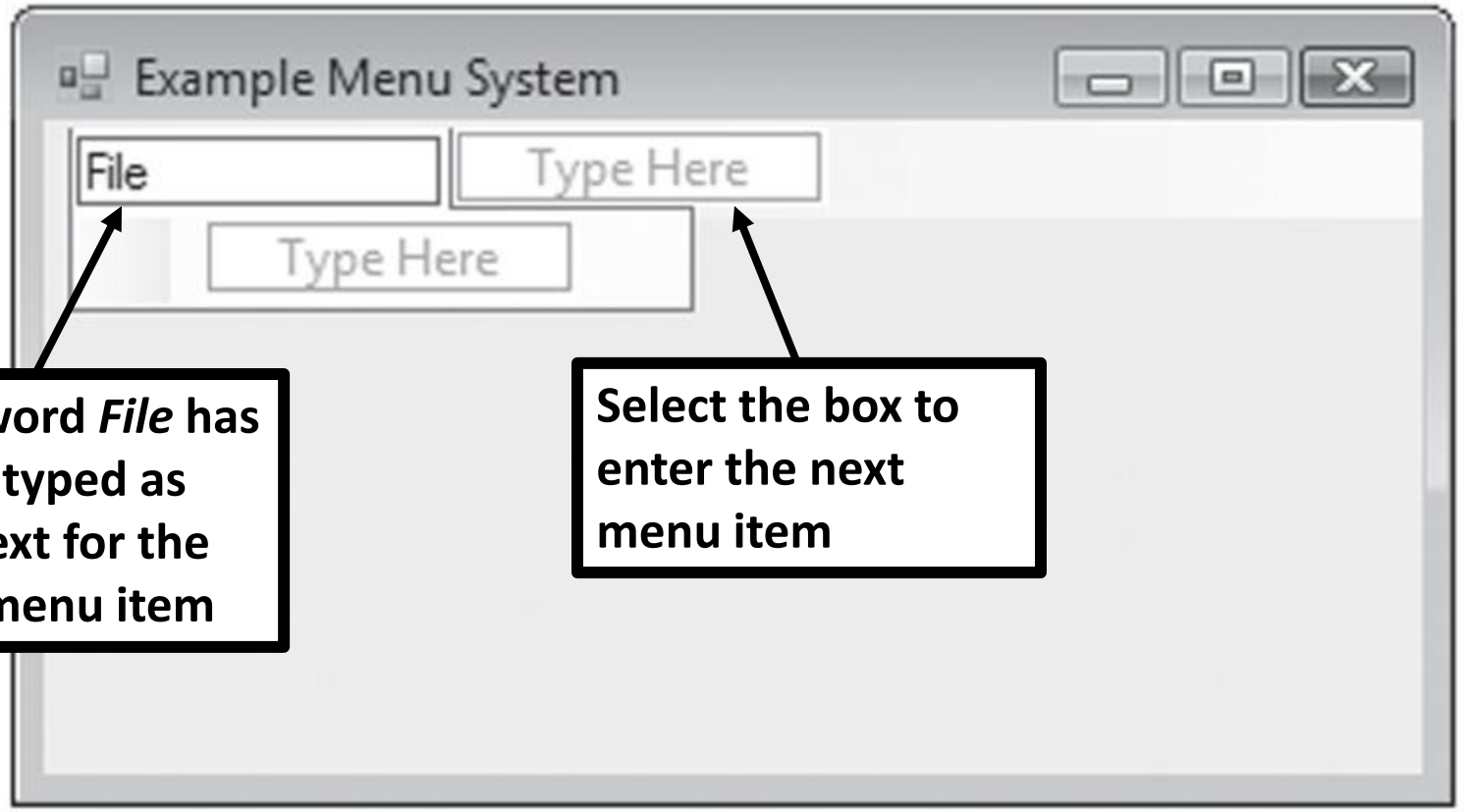


- Actions may be performed using a key or key combination called a **shortcut key**
- A **checked menu command** toggles between the checked (if on) and unchecked (if off) states
- A **separator bar** helps group similar commands

MenuStrip Control

- A **MenuStrip control** adds a menu to a form
  - Double-click on the ***MenuStrip*** icon in the ***Menus & Toolbars*** section of the ***Toolbox***
- The MenuStrip control is displayed in the component tray (bottom of *Design* window)
- A MenuStrip can have many **ToolStripMenuItem objects**:
  - Each represents a single menu command
  - **Name** property - used by VB to identify it
  - **Text** property – text displayed to the user

How to Use the Menu Designer

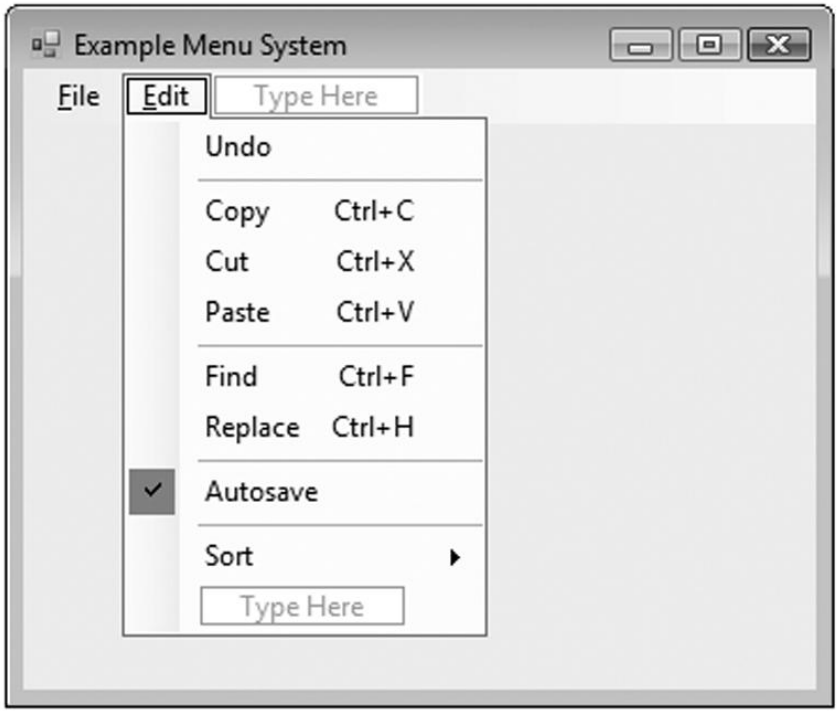


ToolStripMenuItem Object Names

- It is recommended that you change the default value of the Name property so that it
  - Begins with the **mnu** prefix
  - Reflects the Text property and position in the menu hierarchy
    - mnuFile
    - mnuFileSave
    - mnuFilePrint
    - mnuFileExit

Shortcut Keys

- Combination of keys that cause a menu command to execute
  - **Ctrl + C** to copy an item to the clipboard
  - Set with the **ShortcutKeys property**
  - Displayed only if the **ShowShortcut property** is set to **True**



Checked Menu Items

- Turns a feature on or off
  - For example, an alarm for a clock
- To create a checked menu item:
  - Set **CheckOnClick property** to **True**
- Set Checked property to True if feature should be on when the form is initially displayed

```
If mnuSettingsAlarm.Checked = True Then
   MessageBox.Show("WAKE UP!")
End If
```

Disabled Menu Items

- A menu item is grayed out (disabled) with the Enabled property, for example:
  - Paste option is initially disabled and only enabled after something is cut or copied
  - Code initially disables the Paste option

mnuEditPaste.Enabled = False

  - Following a cut or copy, Paste is enabled

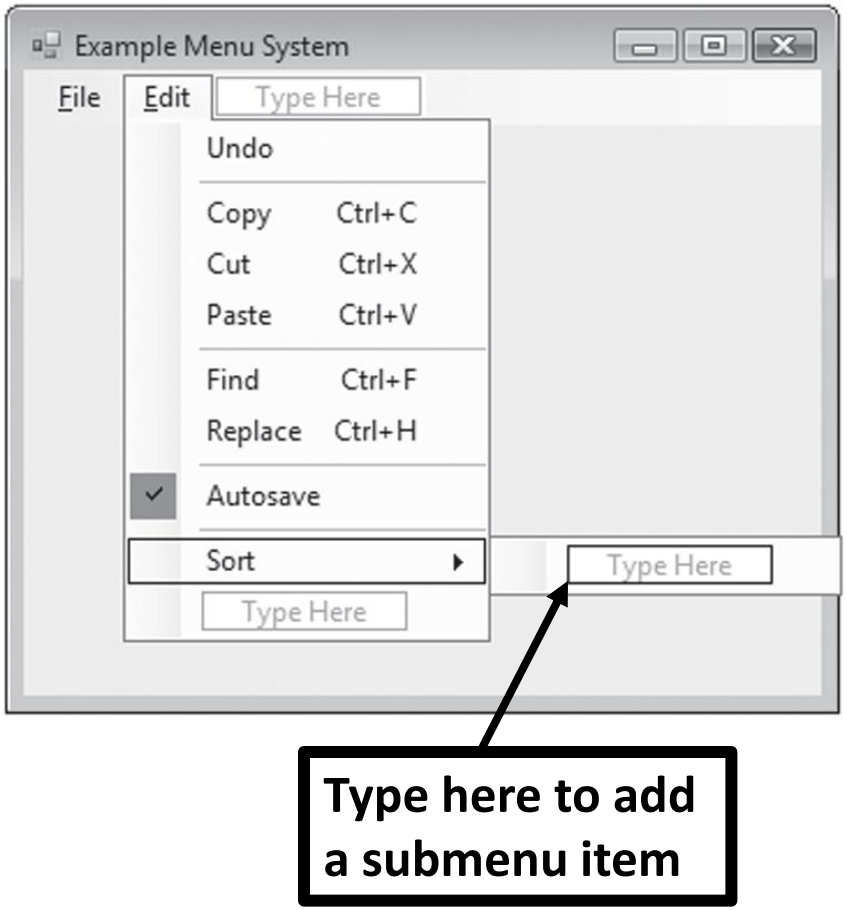
mnuEditPaste.Enabled = True

Adding Separator Bars

- Right-click menu item, select ***Insert Separator***
  - A separator bar will be inserted above the menu item
- Or type a hyphen (**-**) as a menu item's Text property

Submenus

- When selecting a menu item in the designer, a ***Type Here*** box appears to the right
  - Begin a submenu by setting up this menu item
- If a menu item has a submenu, a solid right-pointing arrow(▶)will be shown



Inserting, Deleting, And Rearranging Menu Items

- To insert a new menu item
  - Right-click an existing menu item
  - Select ***Insert*** then ***MenuItem*** from pop-up menu
  - A new menu item will be inserted above the existing menu item
- To delete a menu item
  - Right-click on the item
  - Choose ***Delete*** from the pop-up menu
  - Or select the menu item and press the **Delete** key
- To rearrange a menu item
  - Simply select the menu item in the menu designer and drag it to the desired location

ToolStripMenuItem Click Event

- Menus and submenus require no code
- Commands must have a click event procedure
  - Double click on the menu item
  - Event procedure created in the code window
  - Programmer supplies the code to execute
- Suppose a menu system has a ***File*** menu with an ***Exit*** command named **mnuFileExit**

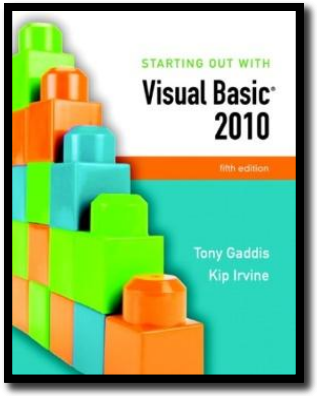
```
Private Sub mnuFileExit_Click(...) Handles mnuFileExit.Click
    ' Close the form.
    Me.Close()
End Sub
```

Standard Menu Items

- Most applications to have the following menu items
  - ***File*** as the leftmost item on the menu strip
    - Access key **Alt + F**
  - An ***Exit*** command on the *File* menu
    - Access key **Alt + X**
    - Shortcut key **Alt + Q** (optional)
  - ***Help*** as the rightmost item on the menu strip
    - Access key **Alt + H**
  - An ***About*** command on the Help menu
    - Access key **Alt + A**
    - Displays an *About* box
- Tutorial 7-5 demonstrates how to build a simple menu system

Context Menus

- A **context menu**, or pop-up menu, is displayed when the user right-clicks a form or control
- To create a context menu
  - Double-click the *ContextMenuStrip* icon in the *Toolbox* window
  - A *ContextMenuStrip* control appears in the component tray
  - Change the *ContextMenuStrip* control's default Name property
  - Add menu items with the menu designer
  - Create click event procedures for the menu items
  - Associate the context menu with a control
  - Set the control's *ContextMenuStrip* property to the name of the *ContextMenuStrip* control



Section 7.4

FOCUS ON PROBLEM SOLVING: BUILDING THE *HIGH ADVENTURE TRAVEL AGENCY PRICE QUOTE APPLICATION*

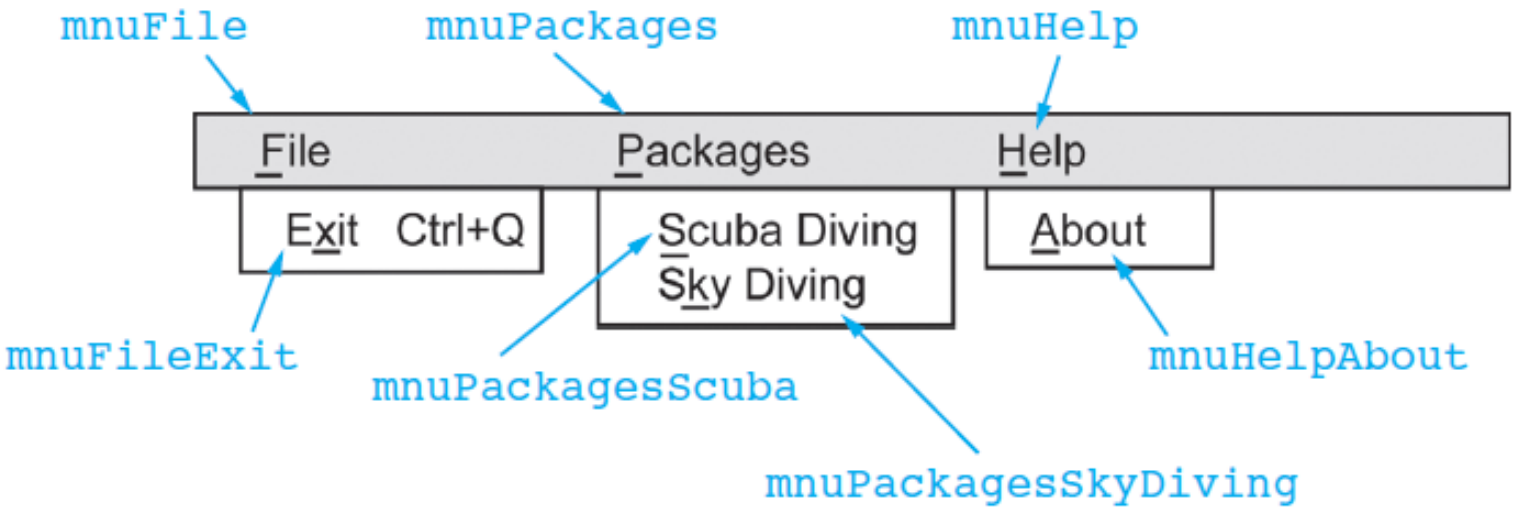
In this section you build an application for the High Adventure Travel Agency. The application uses multiple forms, a module, and a menu system.

Addison Wesley
is an imprint of

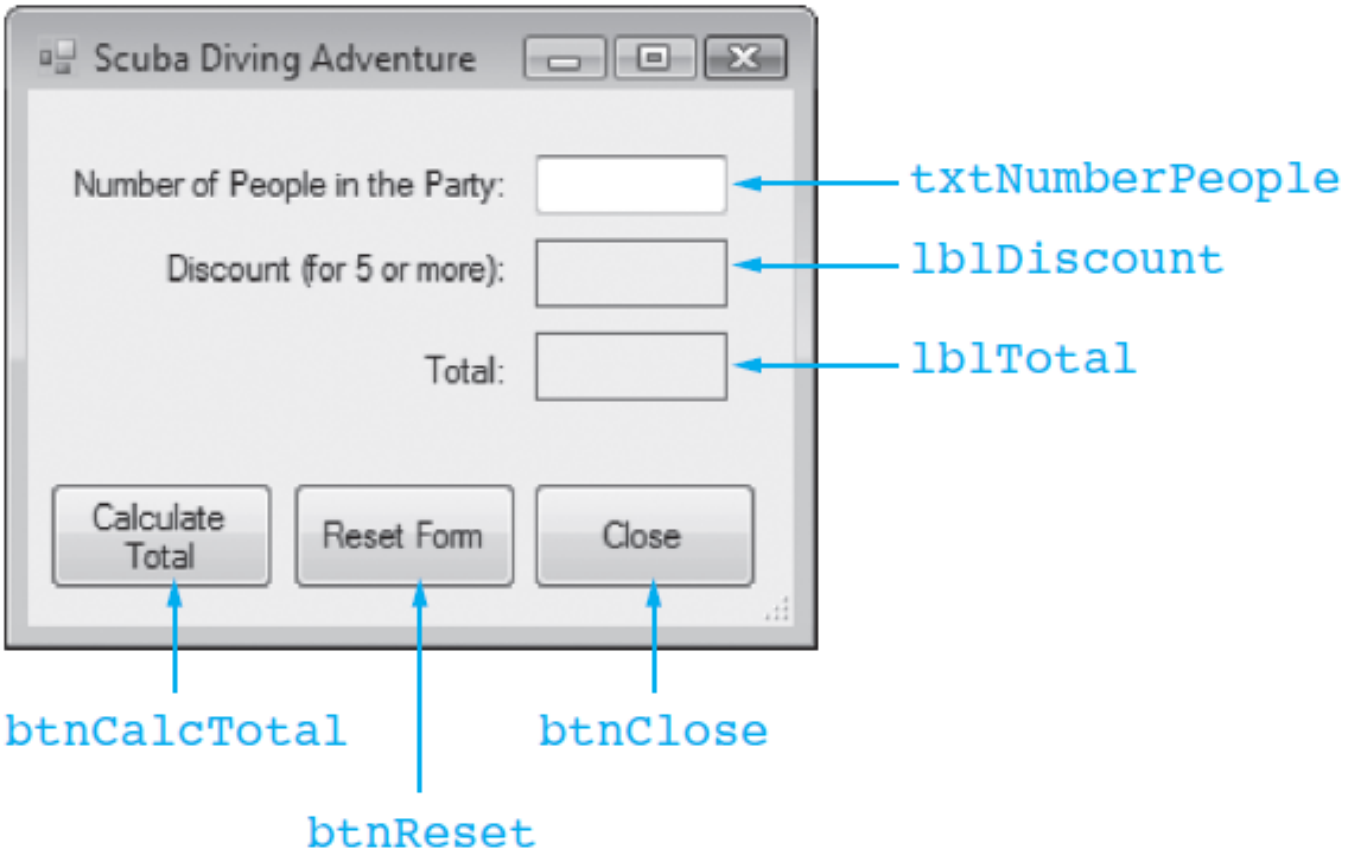
PEARSON

The MainForm Form

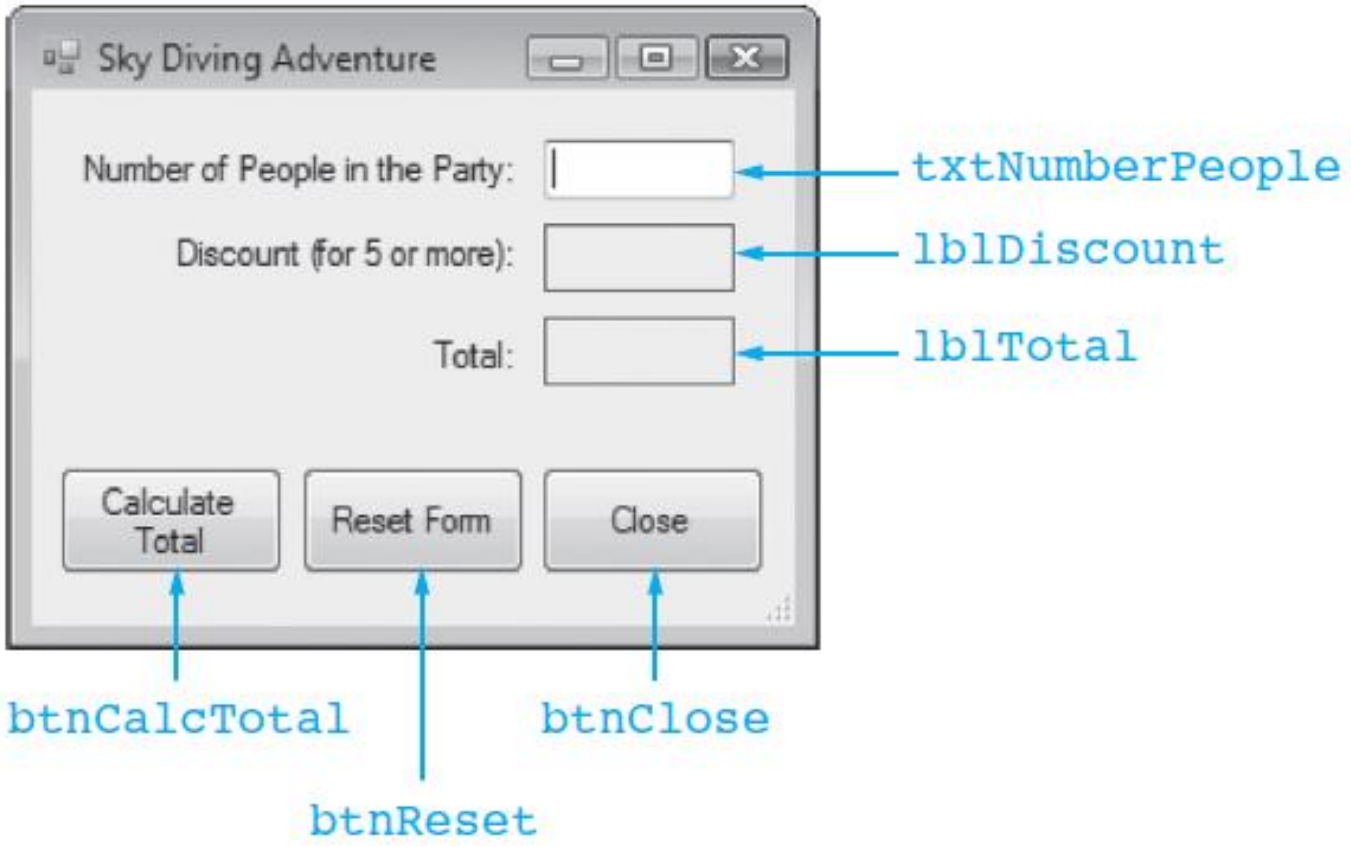
The MainForm Menu System



The ScubaForm Form



The SkyDiveForm Form



The PriceCalcModule Module

```
1    Module PriceCalcModule
2       ' Global constants
3       Public Const g_intMINIMUM_FOR_DISCOUNT As Integer = 5
4       Public Const g_decDISCOUNT_PERCENTAGE As Decimal = 0.1D
5
6       ' The DiscountAmount function accepts a package total
7       ' as an argument and returns the amount of discount
8       ' for that total.
9
10       Public Function DiscountAmount(ByVal decTotal As Decimal) As Decimal
11          Dim decDiscount As Decimal ' To hold the discount
12
13          ' Calculate the discount.
14          decDiscount = decTotal * g_decDISCOUNT_PERCENTAGE
15
16          ' Return the discount.
17          Return decDiscount
18       End Function
19    End Module
```