CHAPTER 5:

MULTIPLE FORMS

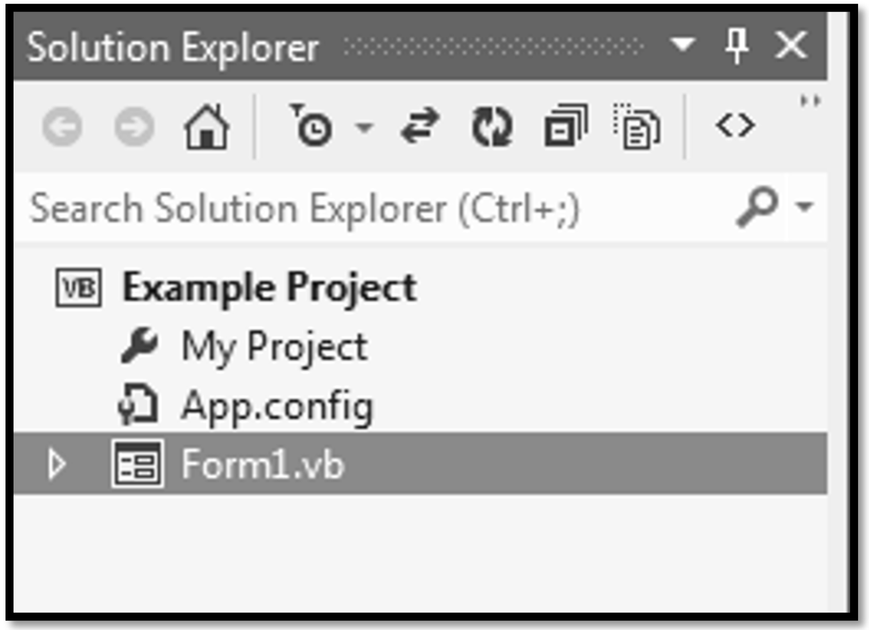
- Create a project with multiple forms, use the Show and Hide methods to display and hide forms
- Differentiate between variables that are global to project and those visible only to a form

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 executes
  - Assigned to the first form by default
  - Can be assigned to any form in the project

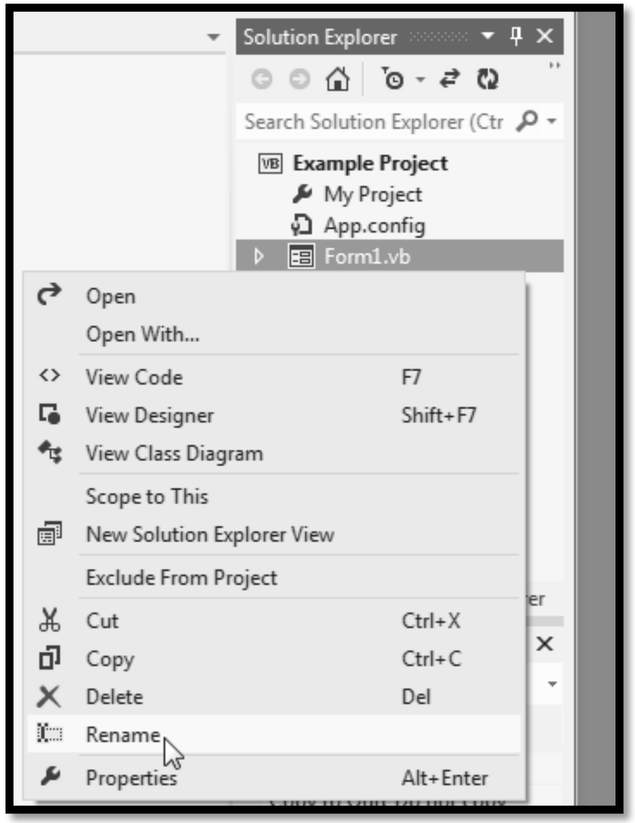
FORM FILES AND FORM NAMES

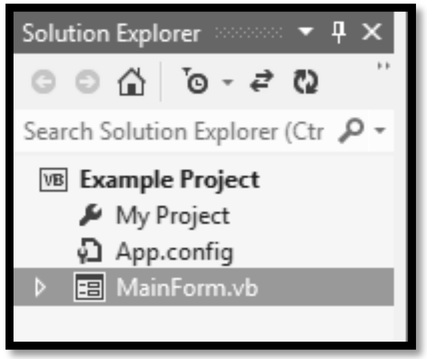
- Each form has a Name property
  - ❖ Set to `Form1` by default
- Each form also has a file name
  - ❖ Stores the code associated with the form
  - ❖ Viewed in the *Code* window
  - ❖ Followed by the *.vb* extension
  - ❖ Shown in the *Solution Explorer* window



RENAMING AN EXISTING FORM FILE

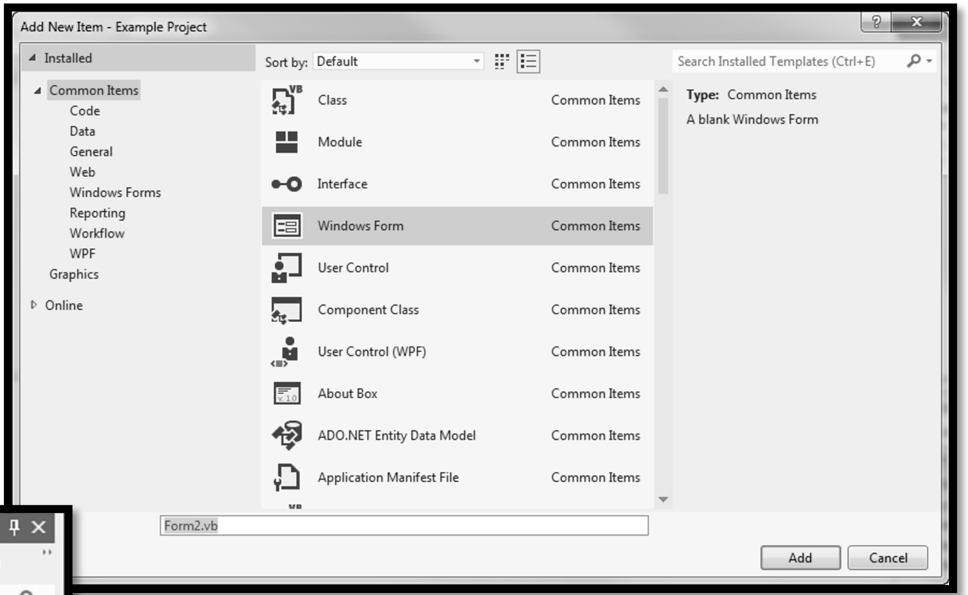
- Always use the *Solution Explorer* window to change the file name and the form's Name property will change automatically
- To rename a form file:
  - ❖ Right-click file name in *Solution Explorer*
  - ❖ Select *Rename* from the menu
  - ❖ Type the new name for the form
  - ❖ Be sure to keep the *.vb*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





ADDING A NEW FORM TO A PROJECT

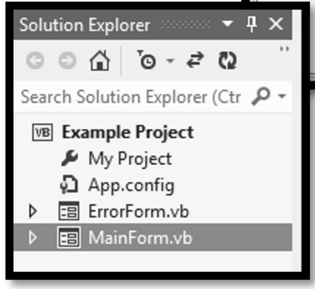
- To add a new form to a project:
  - Click *PROJECT* on the Visual Studio menu bar, and then select *Add Windows Form* . The *Add New Item* window appears
  - Enter the new Name in the *Name* text box
  - Click the *Add* button
- A new blank form is added to your project.


Add New Item - Example Project

Class	Common Items	
Module	Common Items	
Interface	Common Items	
Windows Form	Common Items	
User Control	Common Items	
Component Class	Common Items	
User Control (WPF)	Common Items	
About Box	Common Items	
ADO.NET Entity Data Model	Common Items	
Application Manifest File	Common Items	

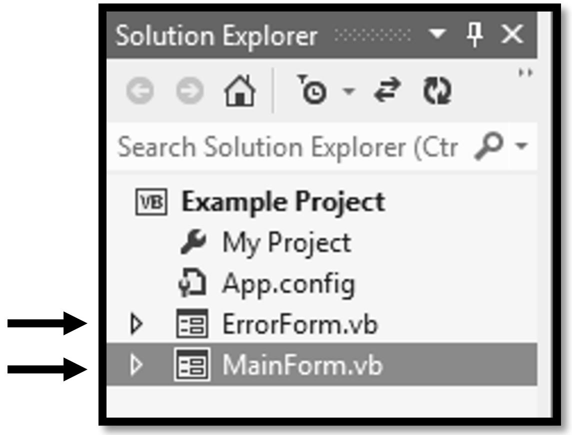
Type: Common Items
A blank Windows Form

Name: Form2.vb

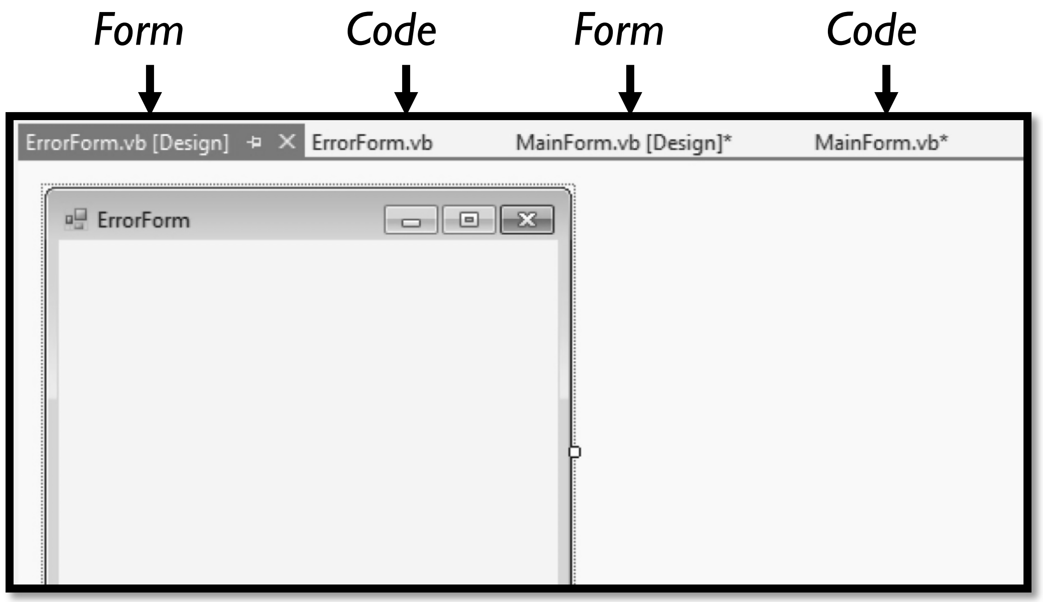


SWITCHING BETWEEN FORMS AND FORM CODE

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



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

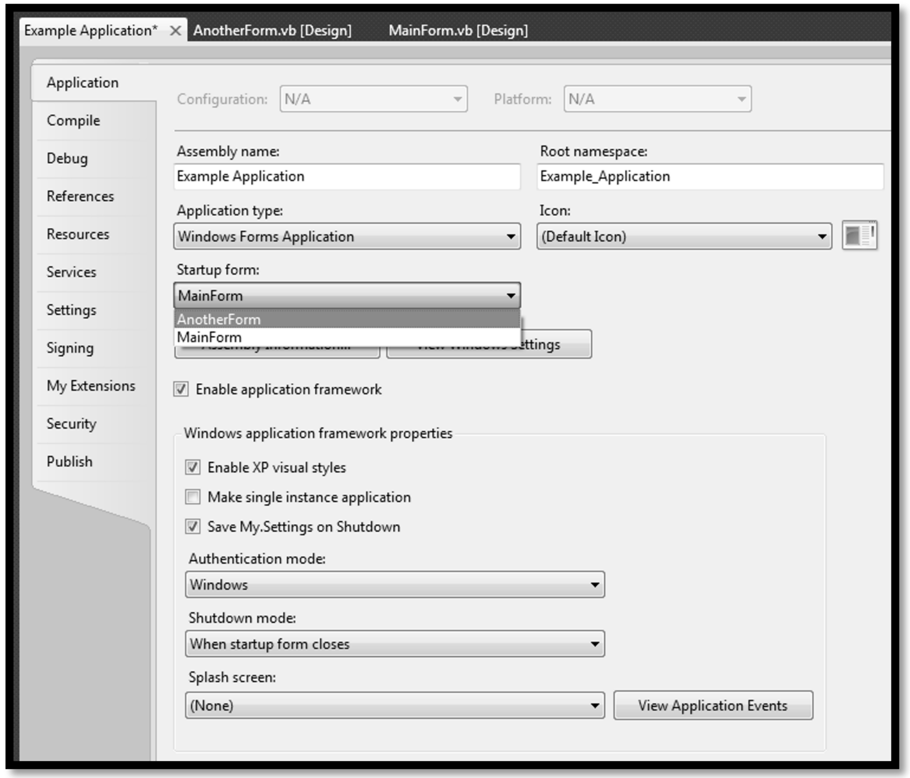


REMOVING A FORM

- 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, or*
  - Select the form's entry in the *Solution Explorer* window
  - Click *Project* on the menu, and click *Exclude From Project*

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



Configuration: N/A Platform: N/A

Assembly name: Example Application
Root namespace: Example_Application
Application type: Windows Forms Application
Icon: (Default Icon)
Startup form: MainForm
AnotherForm
MainForm

☑ Enable application framework

Windows application framework properties
☑ Enable XP visual styles
☐ Make single instance application
☑ Save My.Settings on Shutdown

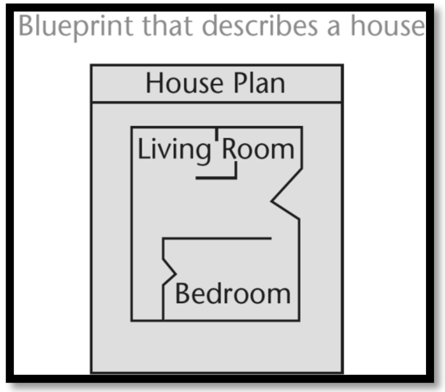
Authentication mode: Windows
Shutdown mode: When startup form closes
Splash screen: (None)

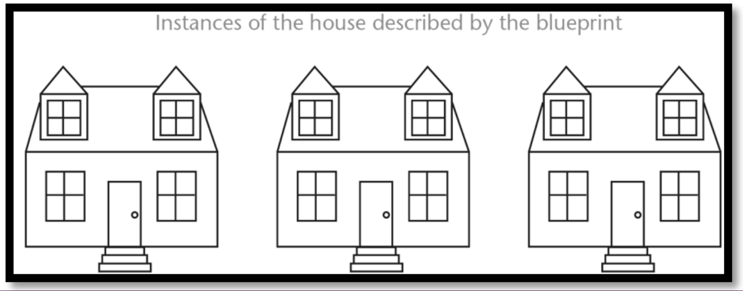
CREATING AN INSTANCE OF A FORM

- 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:

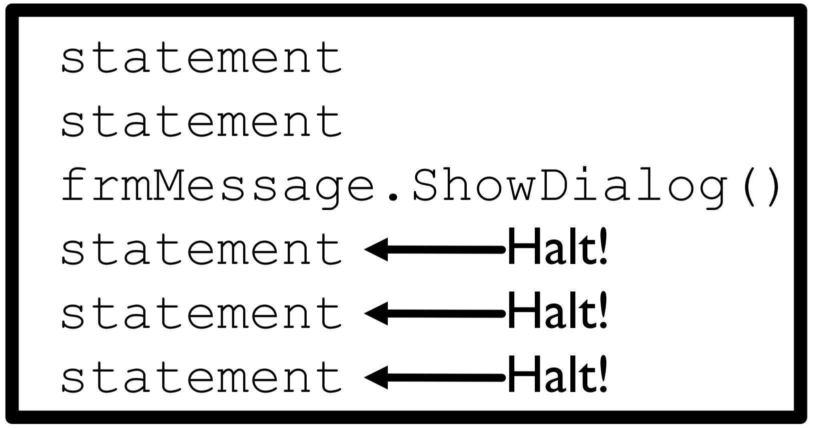
```
ObjectVariable.Show()
```

- For example:

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

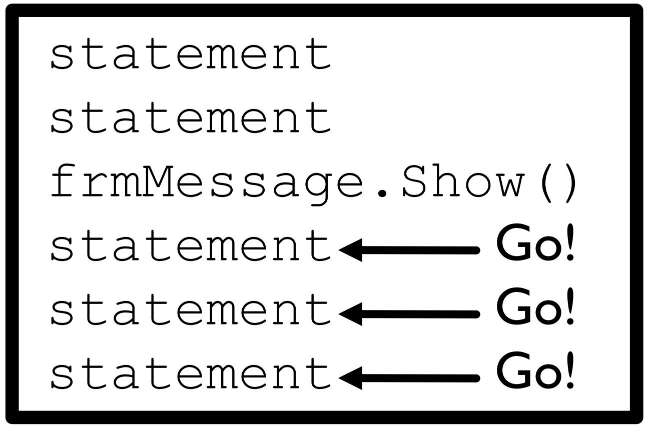
MORE ON MODAL AND MODELESS FORMS

- 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!
```

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 keyword `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

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
```

- Complete the template with the statements you wish the procedure to execute

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. Select the form in the *Designer* window
  2. Select the *Events* button in the *Properties* window toolbar
  3. Double-click the *Activated* event name in the Properties window
- After completing these steps, a code template for the 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 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. Select the form in the *Designer* window
  2. Select the *Events* button ⚡ in the *Properties* window toolbar
  3. Double-click the *FormClosing* event name in the Properties window
- After completing these steps, a code template for the event handler is created in the *Code* window

THE FORMCLOSED EVENT

- The FormClosing event occurs after a form has closed
- The event can be used to execute code immediately after a form has closed
- To create an FormClosed event handler, follow these steps:
  1. Select the form in the *Designer* window
  2. Select the *Events* button in the *Properties* window toolbar
  3. Double-click the *FormClosed* event name in the Properties window
- After completing these steps, a code template for the 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()
```

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 also Global
```

- 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

END OF CHAPTER 5 …
MULTIPLE FORMS