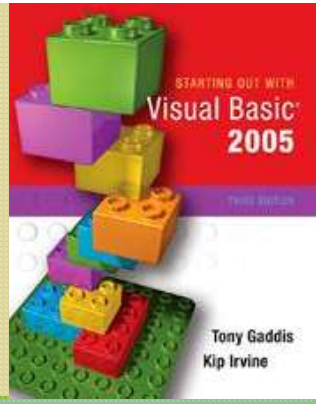


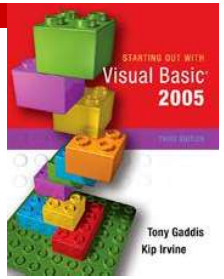
7.1 Multiple Forms

Visual Basic Projects May Have Multiple Forms

A Form Designated as the Startup Object Is Displayed When the Project Executes

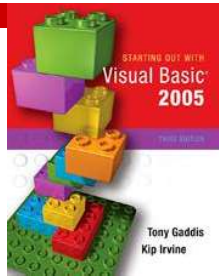
Other Forms in a Project Are Displayed by Programming Statements

Form Names



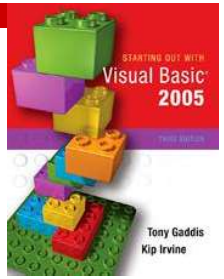
- Each form has its specific name
  - Programs refer to a form by this name
  - VB assigns a default name **`Form1`** to forms
  - A form's *Name* property allows us to set or change the form name
  - Standard prefix for form names is **`frm`**
- Each form also has a file name (.vb extension)
  - Forms are stored on disk using this name
  - To change the file name:
    - Right click in Solution Explorer, select Rename

Adding a New Form to a Project



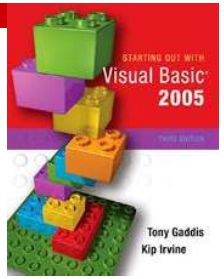
- Click *Add New Item* on the toolbar
  - Or *Project* on menu, then *Add Windows Form*
- *Add New Item* dialog box appears
- Click on *Windows Form* under *Templates*
- Change the default name if you wish
- Click the *Open* button
- New form now appears in:
  - Design window
  - Solution Explorer

Switching from Forms to Form Code



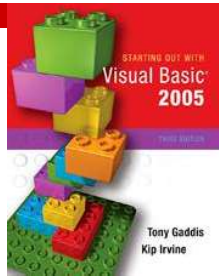
- Design window has two tabs for each form
  - One for form design
  - One for the code associated with a form
- For two forms named frmMain and frmError, can select from the following tabs:
  - frmMain.vb[Design] Main form design
  - frmMain.vb Main form code
  - frmError.vb[Design] Error form design
  - frmError.vb Error form code

Changing the Startup Form



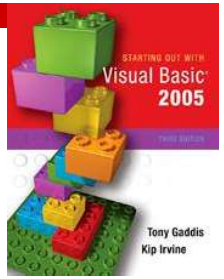
- First form created in a project becomes the *startup object*
  - The form displayed when application runs
- To make another form the startup object
  - Right-click project name in Solution Explorer
  - Click *Properties*
  - Click down arrow in *Startup Form* box
  - Select new startup form from drop-down list
  - Click Ok

Classes and Instances



- 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 one or more instances of the form
  - Like building a house from the blueprint
- In order to use a form in a program, we must first create an instance of it from the design

Creating an Instance of a Form



- Dim statement used to create instance of form

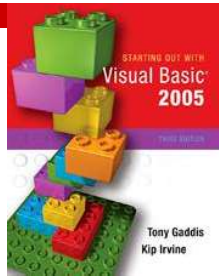
  `Dim` *`ObjectVariable`* `As New` *`ClassName()`*

- To create an instance of frmError:

  ```
  Dim errorForm As New frmError()
  ```

  - `frmError` is the form design name (the class)
  - `New frmError` creates an instance of the form
  - Variable `errorForm` refers to the form instance and is used to perform operations on the form
- The form is not yet visible, but it now exists
- *Show* or *ShowDialog* makes the form visible

Modal Forms & ShowDialog Method

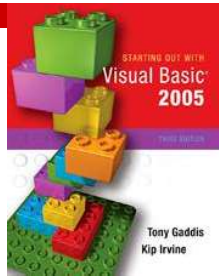


- A *modal form* prevents the user from changing focus to another form in the application as long as it remains open
- For example:

```
errorForm.ShowDialog()
```

  - Variable **`errorForm`** represents an instance of **`frmError`** as shown in the previous slide
  - The *ShowDialog* method displays the form instance named **`errorForm`** as a modal form
- Must close **`errorForm`** in order to change focus to another form in the application

Modeless Forms & Show Method

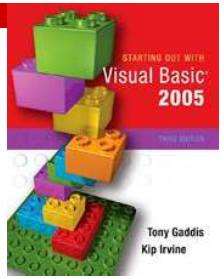


- A *modeless form* allows the user to change focus at will to another form in the application while that form remains open
- For example:

```
errorForm.Show()
```

  - Variable **`errorForm`** represents an instance of **`frmError`** as shown previously
  - The *Show* method displays the form instance named **`errorForm`** as a modeless form
- Can change focus to other forms in the application while **`errorForm`** remains open

Closing a Form



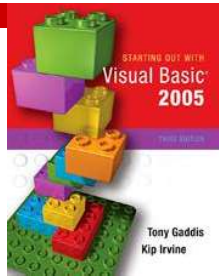
- A form may close itself using the *Close* method and referring to itself as "Me":

```
Me.Close()
```

- As in

```
Private Sub btnClose_Click(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) _
                           Handles btnClose.Click
        Me.Close()
End Sub
```

Hiding a Form

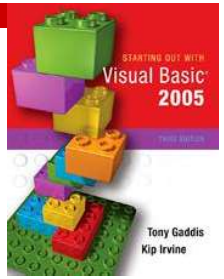


- Closing a Form eliminates it from memory
- To retain the form in memory but remove it from the display, use the *Hide* Method:

```
Me.Hide()
```

- To redisplay the form use the *ShowDialog* or *Show* method

More on Modal and Modeless Forms



- Display of a modal form causes execution of calling statements to halt until form is closed

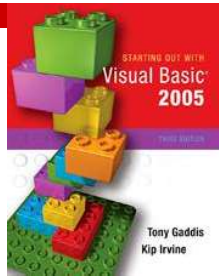
```
statement;
messageForm.ShowDialog()
        ' Statements below will
        ' not execute until the
        ' Form is closed
statement;
```

- Display of a modeless form allows execution to continue

```
statement;
messageForm.Show()
        ' Statements below will
        ' execute right after the
        ' Form is displayed
statement;
```

- Tutorial 7-1 demonstrates these differences

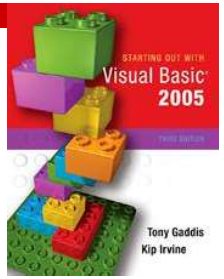
The Form Load Event



- 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

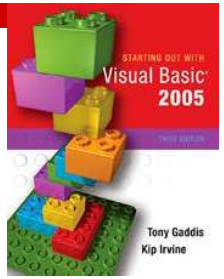
```
Private Sub frmMain_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

End Sub
```

The Form Activated Event



- The *Activated* event is triggered when focus switches to the form from another form or application
- The Load event is triggered once when the form is initially displayed
- The Activated event is also triggered when the form is initially displayed
  - Occurs immediately after the Load event
- The Activated event may be triggered many more times while a form is being displayed

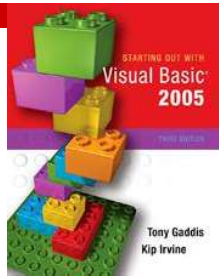
The Form Closing Event



- The *Closing* event is triggered as the form is being closed, but before it has closed
- The Closing event can be used to ask the user if they really want the form closed

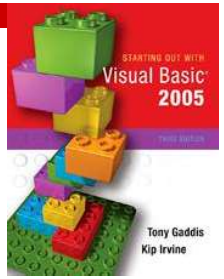
```
Private Sub frmMain_Closing(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    If MessageBox.Show("Are you Sure?", "Confirm", _
       MessageBoxButtons.YesNo) = DialogResult.Yes Then
       e.Cancel = False              'continue, close form
    Else
       e.Cancel = True               'cancel form close
    End If
End Sub
```

The Form Closed Event



- *Closed* event triggered after a form is closed
- Note that it is now too late to prevent the form from being closed (it is already)

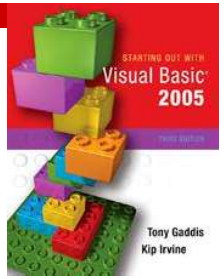
Using Objects on a Different Form



- When code in a form refers to an object, it is assumed that object is in that same form
- You can refer to an object in another form
  - Simply preface the object name with the variable name associated with that form
  - frmGreeting has a control named lblMessage
  - Set Text property to *Hello* before displaying

```
Dim greetingForm As New frmGreeting()
greetingForm.lblMessage.Text = "Hello!"
greetingForm.ShowDialog()
```

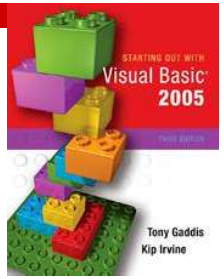
Class-level Variables in a Form



- Class-level variables are Private by default
- This means they are not accessible by code in other forms
- If you want to access from other forms, they must be declared with the Public qualifier:

```
Public sngTotal As Single
       ' Instead of the declaration
       ' Dim sngTotal As Single
```

Public/Private Procedures in a Form



- Procedures, by default, are Public
- They can be accessed by code outside of their Form
- To make a procedure invisible outside its own form, declare it to be Private

- Tutorial 7-2 provides an opportunity to work with a multiple form application