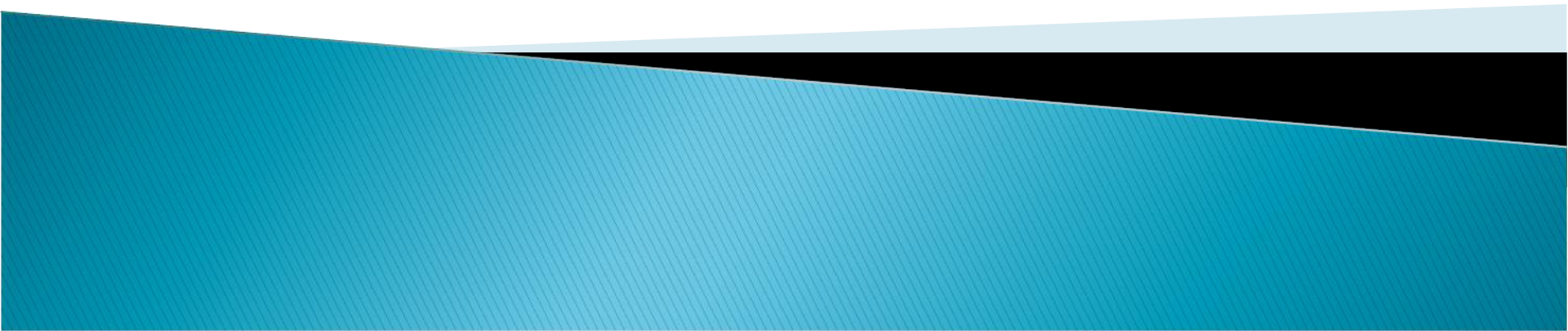
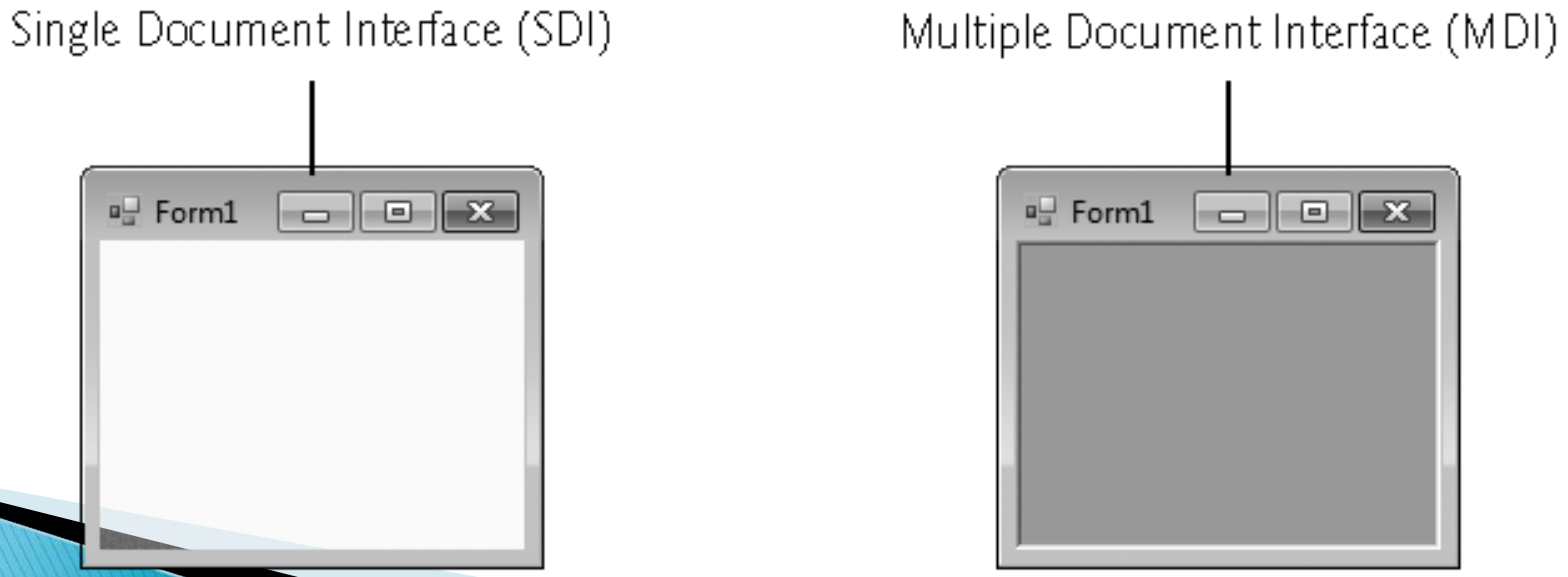
Windows Forms: MDI, TabControl, Visual Inheritance, and UserDefined Controls



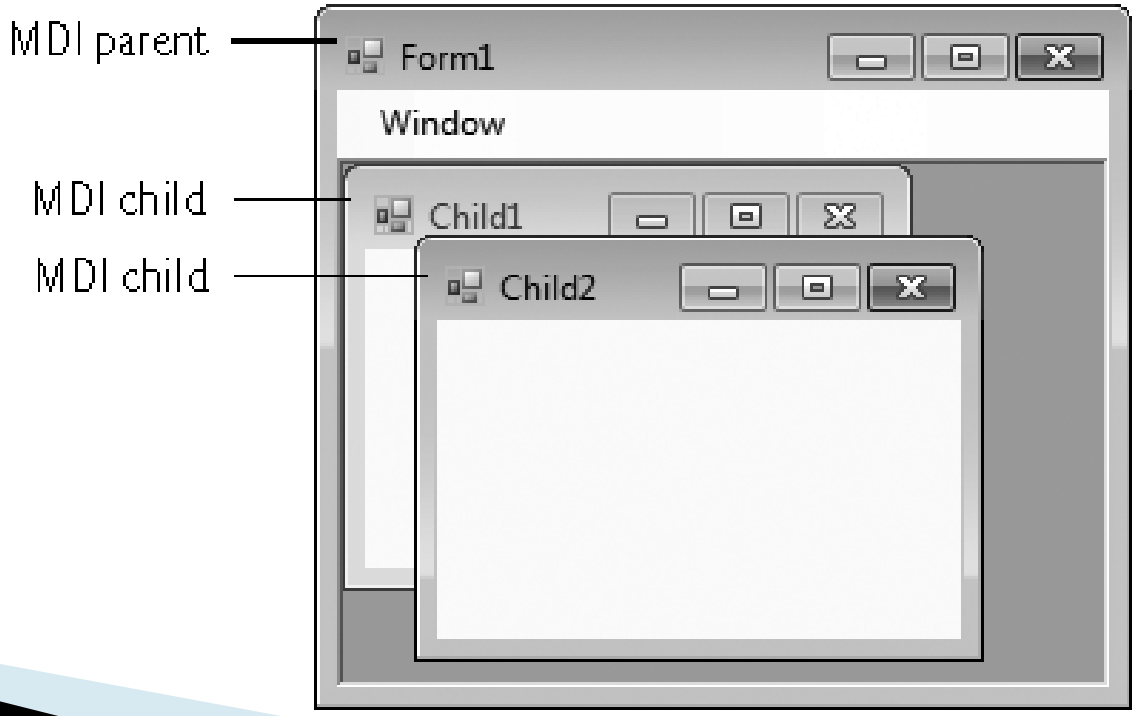
15.12 Multiple Document Interface (MDI) Windows

- **SDI: Single Document Interface**
  - Such programs can support only one open window or document at a time (i.e. MS Notepad and Paint).
- **MDI: Multiple Document Interface**
  - Programs which allow users to edit multiple documents at once (e.g. MS Office products).



15.12 Multiple Document Interface (MDI) Windows

- **Parent window**
  - An MDI program's main window
- **Child window**
  - Each window inside an MDI app



15.12 Multiple Document Interface (MDI) Windows

- An MDI app can have many child windows.
- Each child window has only one parent window.
- A maximum of one child window can be active at once.

- Child windows cannot be parents themselves and cannot be moved outside their parent.

- A child window behaves like any other window (with regard to closing, minimizing, resizing, and so on).

- A child window's functionality can differ from that of other child windows of the parent.

15.12 Multiple Document Interface (MDI) Windows

- To create an MDI Form
  - Create a parent form:

    Create a new `Form` and set its `IsMdiContainer` property to `true` (it is listed under Windows Style category).
  - Create a child form:

    Create a new `Form` and name the file.
  - Open a child `Form` from its parent `Form`:

    In the parent's code behind file and usually inside an event handler:

```
ChildFormClass childForm = New ChildFormClass();
childForm.MdiParent = this;
childForm.Show();
```

    - `this` refers to `parentsForm`, since in most cases, the parent `Form` creates the child.

MDI `Form` properties, a method and an event	**Description**
Common MDI Child Properties	
`IsMdiChild`	Indicates whether the `Form` is an MDI child. If `true`, `Form` is an MDI child (read-only property).
`MdiParent`	Specifies the MDI parent `Form` of the child.
Common MDI Parent Properties	
`ActiveMdiChild`	Returns the `Form` that's the currently active MDI child (returns `null` if no children are active).
`IsMdiContainer`	Indicates whether a `Form` can be an MDI parent. If `true`, the `Form` can be an MDI parent. The default value is `false`.
`MdiChildren`	Returns the MDI children as an array of `Forms`.

MDI `Form` properties, a method and an event	**Description**
Common Method	
`LayoutMdi`	Determines the display of child forms on an MDI parent. The method takes as a parameter an `MdiLayout` enumeration with possible values `ArrangeIcons`, `Cascade`, `TileHorizontal` and `TileVertical`. Figure 15.42 depicts the effects of these values.
Common Event	
`MdiChildActivate`	Generated when an MDI child is closed or activated.

15.12 Multiple Document Interface (MDI) Windows

- Property `MdiWindowListItem` of `MenuStrip`

  specifies which menu item, if any, displays a list of open child windows.

  – When a new child window is opened, an entry is added to the end of the list.

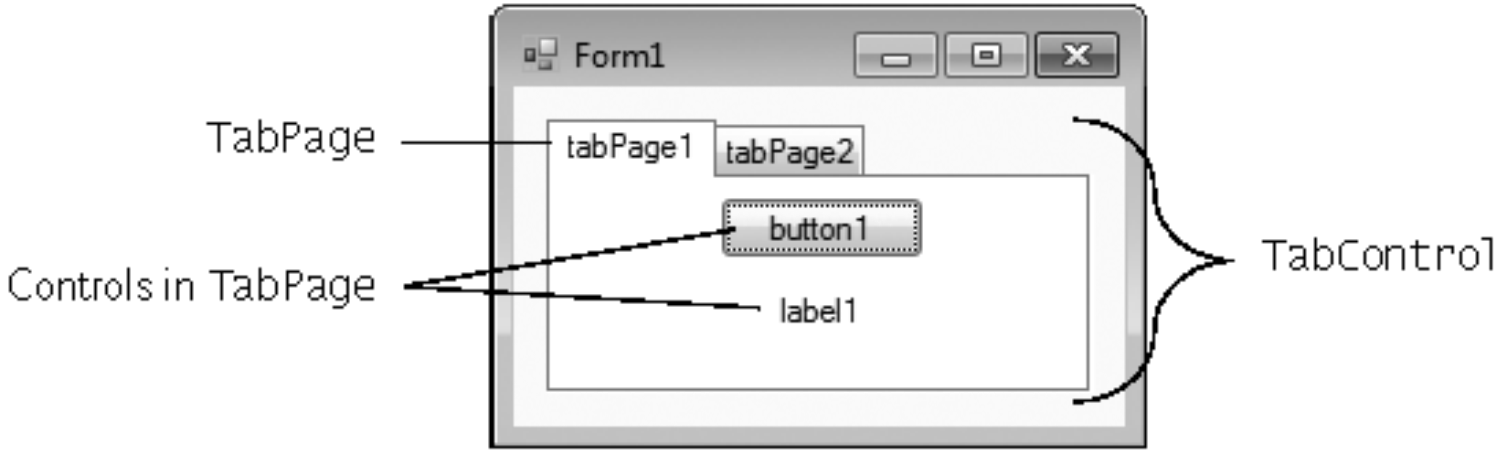
- *Practice:*

  – Download 18-1-UsingMDI-Practice

  – Follow the instructions in the project

15.11 TabControl Control

- The `TabControl` creates tabbed windows.
- `TabControl`s contain `TabPage` objects, which are similar to `Panel`s.



15.11 TabControl Control

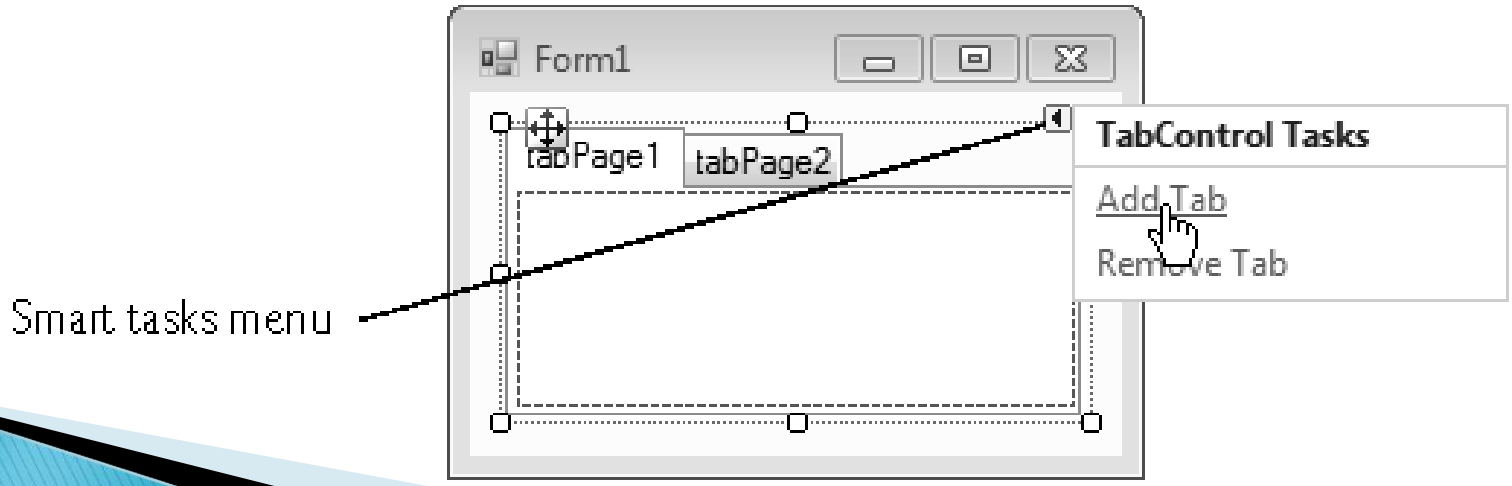
- First add controls to the TabPage objects, then add the TabPages to the TabControl.

```
myTabPage.Controls.Add( myControl );
myTabControl.TabPages.Add( myTabPage );
```

- We can use method AddRange to add an array of TabPages or controls to a TabControl or TabPage.

15.11 TabControl Control

- In Design mode
  - Add TabControls visually by dragging and dropping them onto a Form.
  - To add TabPages, click the top of the TabControl, open its *smart tasks menu* and select Add Tab.
  - To select a TabPage, click the control area underneath the tabs.

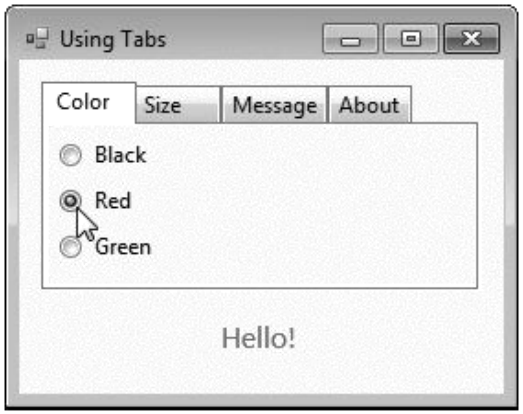


`TabControl` properties and an event	Description
Common Properties	
`ImageList`	Specifies images to be displayed on tabs.
`ItemSize`	Specifies the tab size.
`Multiline`	Indicates whether multiple rows of tabs can be displayed.
`SelectedIndex`	Index of the selected `TabPage`.
`SelectedTab`	The selected `TabPage`.
`TabCount`	Returns the number of tab pages.
`TabPages`	Returns the collection of `TabPages` within the `TabControl` as a `TabControl.TabPageCollection`.
Common Event	
`SelectedIndexChanged`	Generated when `SelectedIndex` changes (i.e., another `TabPage` is selected).

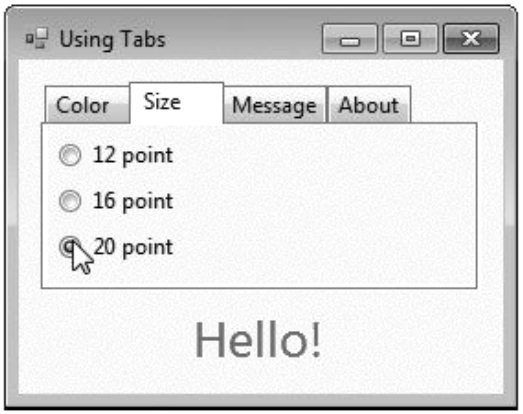
15.11 TabControl Control

- *Example*: 18-2-UsingTabs

a) Selecting the **Red** `RadioButton` from the **Color** tab



b) Selecting the **20 Point** `RadioButton` from the **Size** tab



c) Selecting the **Goodbye!** `RadioButton` from the **Message** tab

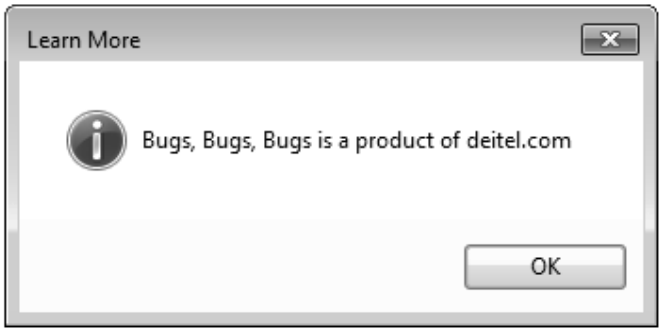
d) Selecting the **About** tab

15.13 Visual Inheritance

- **Visual inheritance** enables you to achieve visual consistency.
- For example, you could define a base `Form` that contains a product's logo and a specific background color.
- You then could use the base `Form` throughout an app for uniformity and branding.

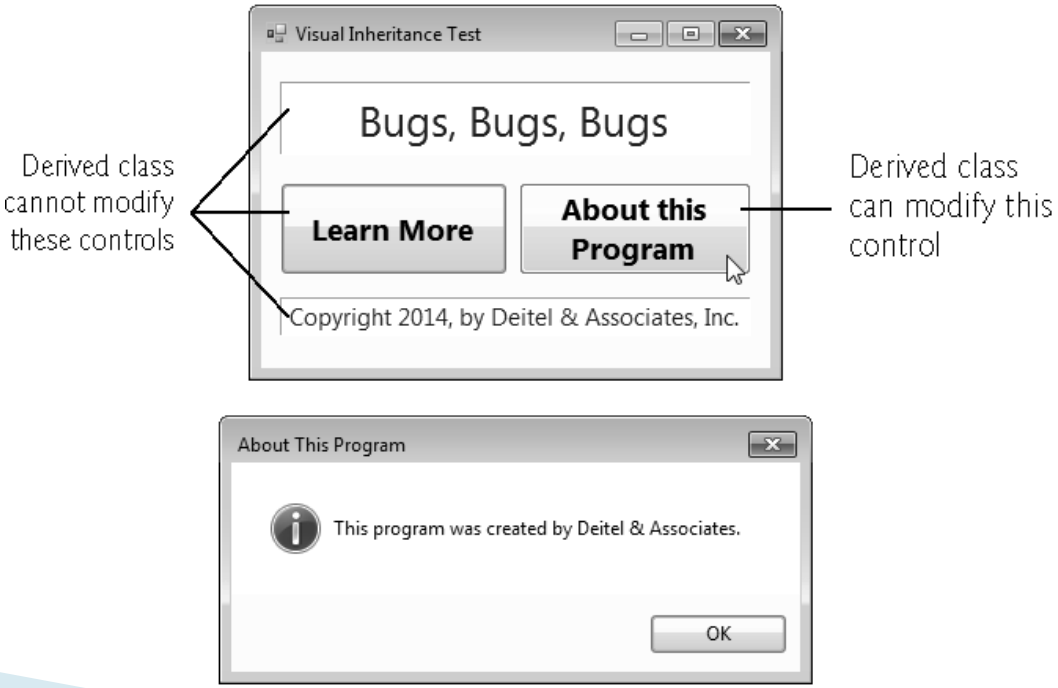
15.13 Visual Inheritance

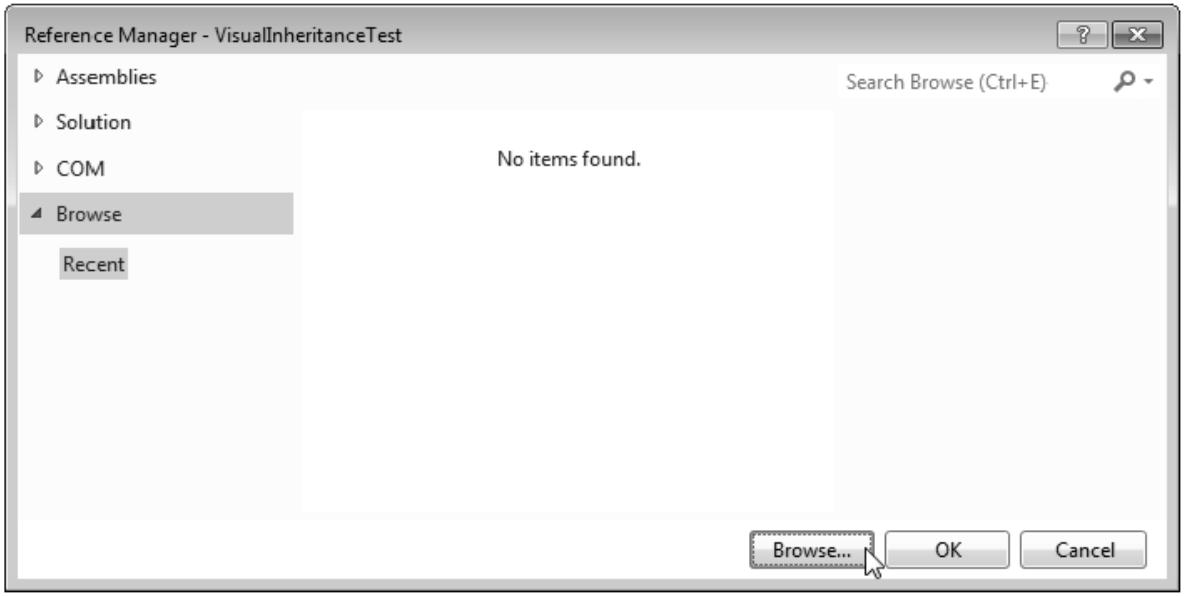
- *Example:* 18-3-VisualInheritanceBase
  - Class `VisualInheritanceBaseForm` derives from `Form`.
  - We use the public class `VisualInheritanceBaseForm`.
  - Use the namespace declaration that was created for us by the IDE.
  - Right click the project name in the **Solution Explorer** and select **Properties**, then choose the **Application** tab.
  - In the **Output** type drop-down list, change **Windows Application** to **Class Library**.
  - Building the project produces the **.dll**.

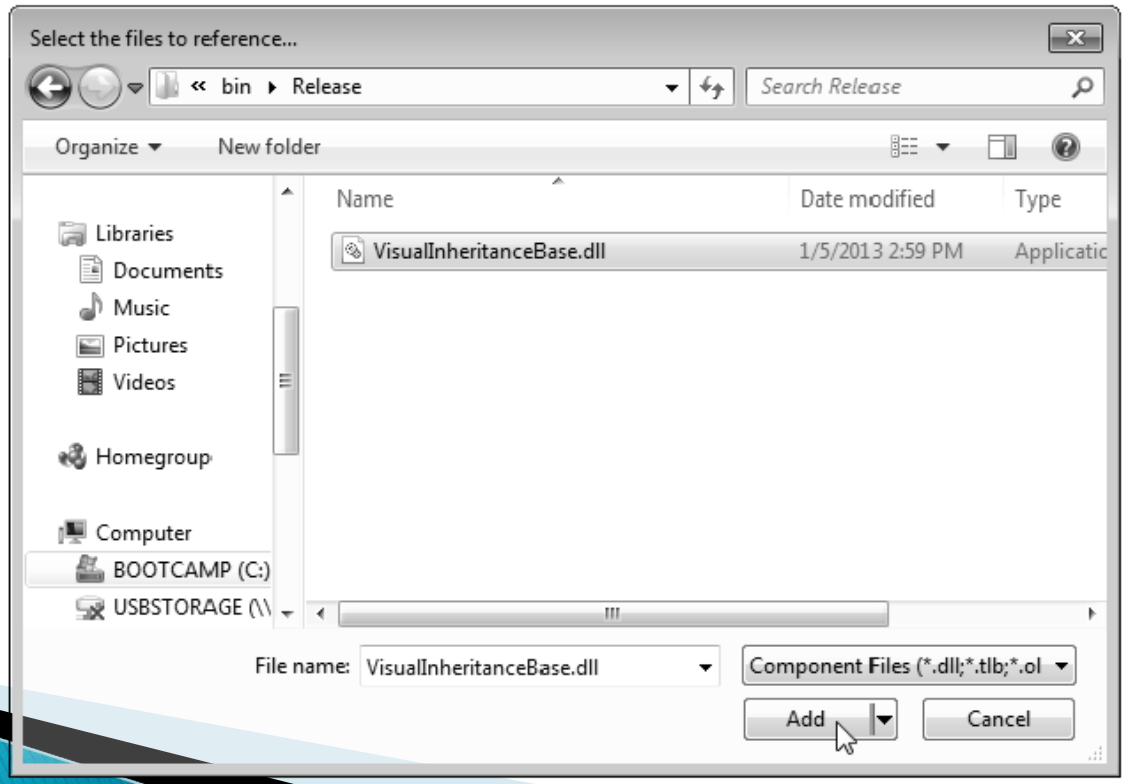


15.13 Visual Inheritance

- Example: 18-3-VisualInheritanceTest
  - To visually inherit from `VisualInheritanceBaseForm`, create a new Windows Forms app.
  - In this app, add a reference to the `.dll` you just created.




Reference Manager - VisualInheritanceTest
Assemblies
Solution
COM
Browse
Recent
Search Browse (Ctrl+E)
No items found.
Browse...
OK
Cancel


Select the files to reference...
« bin ▸ Release
Search Release
Organize
New folder
Libraries
Documents
Music
Pictures
Videos
Homegroup
Computer
BOOTCAMP (C:)
USBSTORAGE (\\
Name
Date modified
Type
VisualInheritanceBase.dll
1/5/2013 2:59 PM
Applicatic
File name:
VisualInheritanceBase.dll
Component Files (*.dll;*.tlb;*.ol
Add
Cancel

15.13 Visual Inheritance

- Modify the line that defines the class:

```
public partial class VisualInheritanceTestForm :
VisualInheritanceBase.VisualInheritanceBaseForm
```

- In Design view, the new app's `Form` should now display the controls inherited from the base `Form`.

15.14 User-Defined Controls

- The .NET Framework allows you to create **custom controls**.
- Custom controls appear in the user's `Toolbox`.
- There are multiple ways to create a custom control, depending on the level of customization that you want.

Custom-control techniques and `PaintEventArgs` properties	**Description**
Custom-Control Techniques	
Inherit from Windows `Forms` control	You can do this to add functionality to a preexisting control. If you override method `OnPaint`, call the base class's `OnPaint` method. You only can add to the original control's appearance, not redesign it.
Create a `UserControl`	You can create a `UserControl` composed of multiple preexisting controls (e.g., to combine their functionality). You place drawing code in a `Paint` event handler or overridden `OnPaint` method.
Inherit from class `Control`	Define a brand new control. Override method `OnPaint`, then call base-class method `OnPaint` and include methods to draw the control. With this method you can customize control appearance and functionality.

Custom-control techniques and `PaintEventArgs` properties	Description
`PaintEventArgs` Properties	
`Graphics`	The control's graphics object, which is used to draw on the control.
`ClipRectangle`	Specifies the rectangle indicating the boundary of the control.

15.14 Timer

- Timers are non-visual components that generate Tick events at a set interval.

- The Timer's Interval property defines the number of milliseconds between events.

15.14 User-Defined Controls

- *Example*: 18-4-ClockControl
  - To create a `UserControl` that can be exported to other solutions, do the following:
    - Create a new `Class Library` project.
    - Delete `Class1.cs`, initially provided with the app.
    - Right click the project in the `Solution Explorer` and select `Add > User Control`…
      - We name the file (and the class) `ClockUserControl`.
    - Add controls and functionality to the `UserControl`.
      - Add a `Label` and a `Timer` to the `UserControl`.
      - Set the `Timer`'s `Interval` to `1000` milliseconds.
      - `clockTimer` must be enabled by setting `Enabled` to `true`.
    - Build the project.
      - Visual Studio creates a `.dll` file for the `UserControl` in the output directory (`bin/Release or bin/Debug`).

15.14 User-Defined Controls

- *Example:* 18-4-ClockControlTest
  - Create a new Windows app.
  - Add the `Clock` control to the `ToolBox`
    - Right click the `ToolBox` and select `Choose Items`....
    - In the `Choose Toolbox Items` dialog, click `Browse`...
    - Select the `ClockControl.dll` file that you created.
    - The item will then appear in the `Choose Toolbox Items` dialog.
    - Check this item and click `OK` to add the item to the `Toolbox`.
  - Add the `Clock` control to the app.