

# **Module 6: Using Windows Forms**

# Overview

- Why Use Windows Forms?
- Structure of Windows Forms
- Using Windows Forms
- Using Controls
- Windows Forms Inheritance

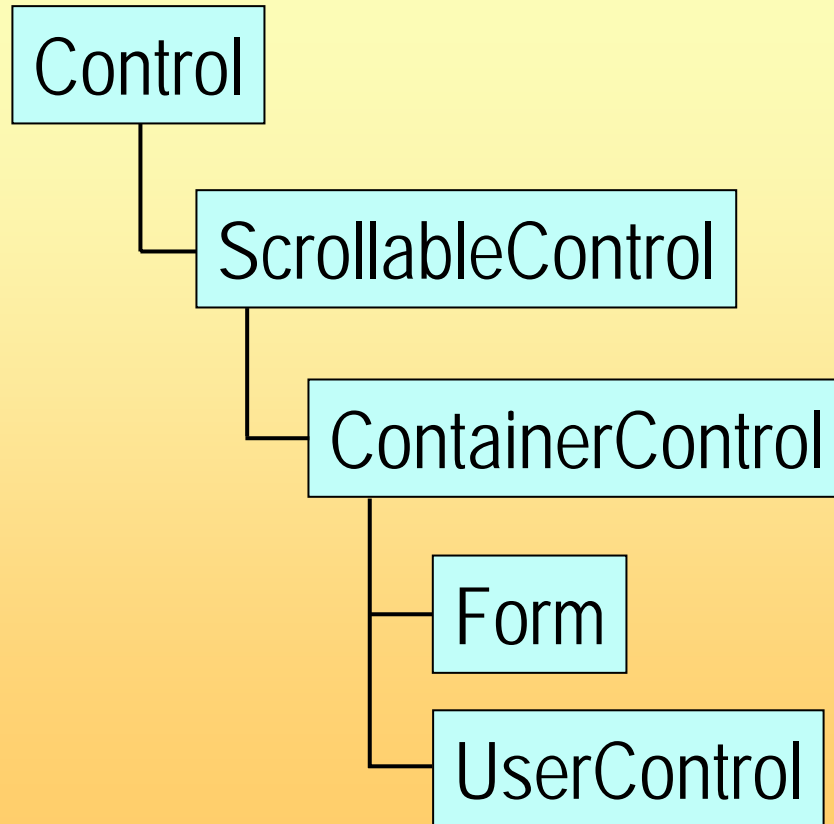
# Why Use Windows Forms?

- Rich set of controls
- Flat look style
- Advanced printing support
- Advanced graphics support – GDI+
- Accessibility support
- Visual inheritance
- Extensible object model
- Advanced forms design

# ◆ Structure of Windows Forms

- Windows Forms Class Hierarchy
- Using the `Windows.Forms.Application` Class
- Examining the Code Behind Windows Forms

# Windows Forms Class Hierarchy



# Using the Windows.Forms.Application Class

## ■ Starting and ending applications

```
Sub Main( )  
    Dim frmFirst as New Form1( )  
    frmFirst.Show( )           'Displays the first form  
    Application.Run( )  
    'Allows the application to continue after the form is closed  
End Sub
```

## ■ Using DoEvents

## ■ Setting and retrieving application information

```
Dim strAppPath As String  
strAppPath = Application.StartupPath  
'use this path to access other files installed there
```

# Examining the Code Behind Windows Forms

## ■ Imports

- To alias namespaces in external assemblies

```
Imports Winforms = System.Windows.Forms
```

## ■ Class

- Inherits from **System.Windows.Forms.Form**
- Constructor – **Sub New()**
- Initializer – **Sub InitializeComponent()**
- Destructor – **Sub Dispose()**

# ◆ Using Windows Forms

- Using Form Properties
- Using Form Methods
- Using Form Events
- Handling Events
- Creating MDI Forms
- Using Standard Dialog Boxes

# Using Form Properties

- DialogResult
- Font
- Opacity
- MaximumSize and MinimumSize
- TopMost
- AcceptButton and CancelButton

# Using Form Methods

## ■ Close

```
If blnEndApp = True Then  
    Me.Close( )  
End If
```

## ■ Show and ShowDialog

```
Dim frm2 As New Form2( )  
frm2.ShowDialog( )  
If frm2.DialogResult = DialogResult.OK Then  
    MessageBox.Show("Processing request")  
ElseIf frm2.DialogResult = DialogResult.Cancel Then  
    MessageBox.Show("Cancelling request")  
End If  
frm2.Dispose( )
```

# Using Form Events

- Activated and Deactivate
- Closing
- Closed
- MenuStart and MenuComplete

# Handling Events

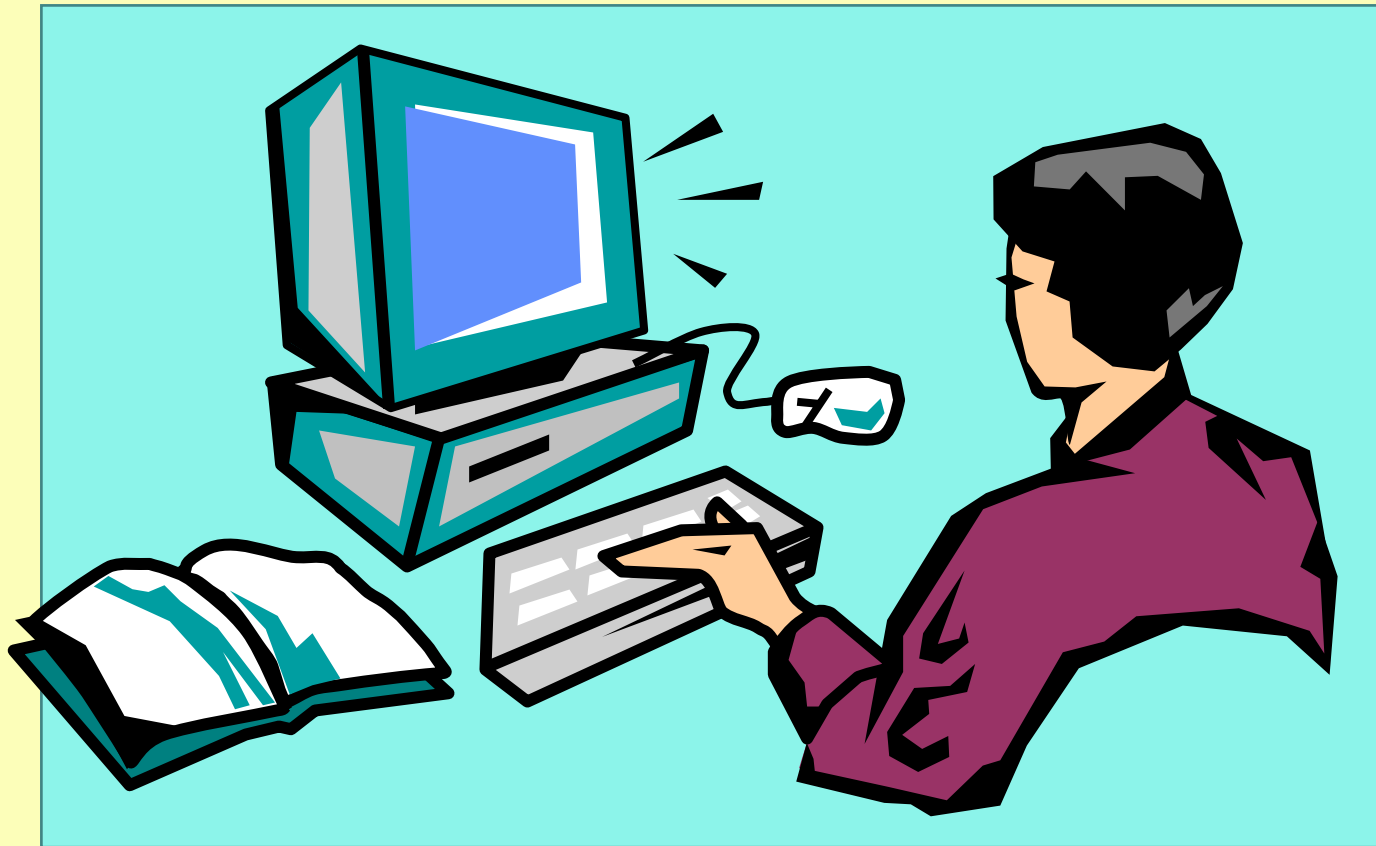
- Handling multiple events with one procedure

```
Private Sub AddOrEditButtonClick(ByVal sender As Object,  
    ByVal e As System.EventArgs)  
    Handles btnAdd.Click, btnEdit.Click
```

- Using AddHandler

```
AddHandler btnNext.Click, AddressOf NavigateBtnClick
```

# Practice: Using Form Events



# Creating MDI Forms

- Creating the parent form

```
Me.IsMdiContainer = True  
Me.WindowState = FormWindowState.Maximized
```

- Creating child forms

```
Dim doc As Form2 = New Form2( )  
doc.MdiParent = Me  
doc.Show( )
```

- Accessing child forms

- Arranging child forms

# Using Standard Dialog Boxes

## ■ MsgBox

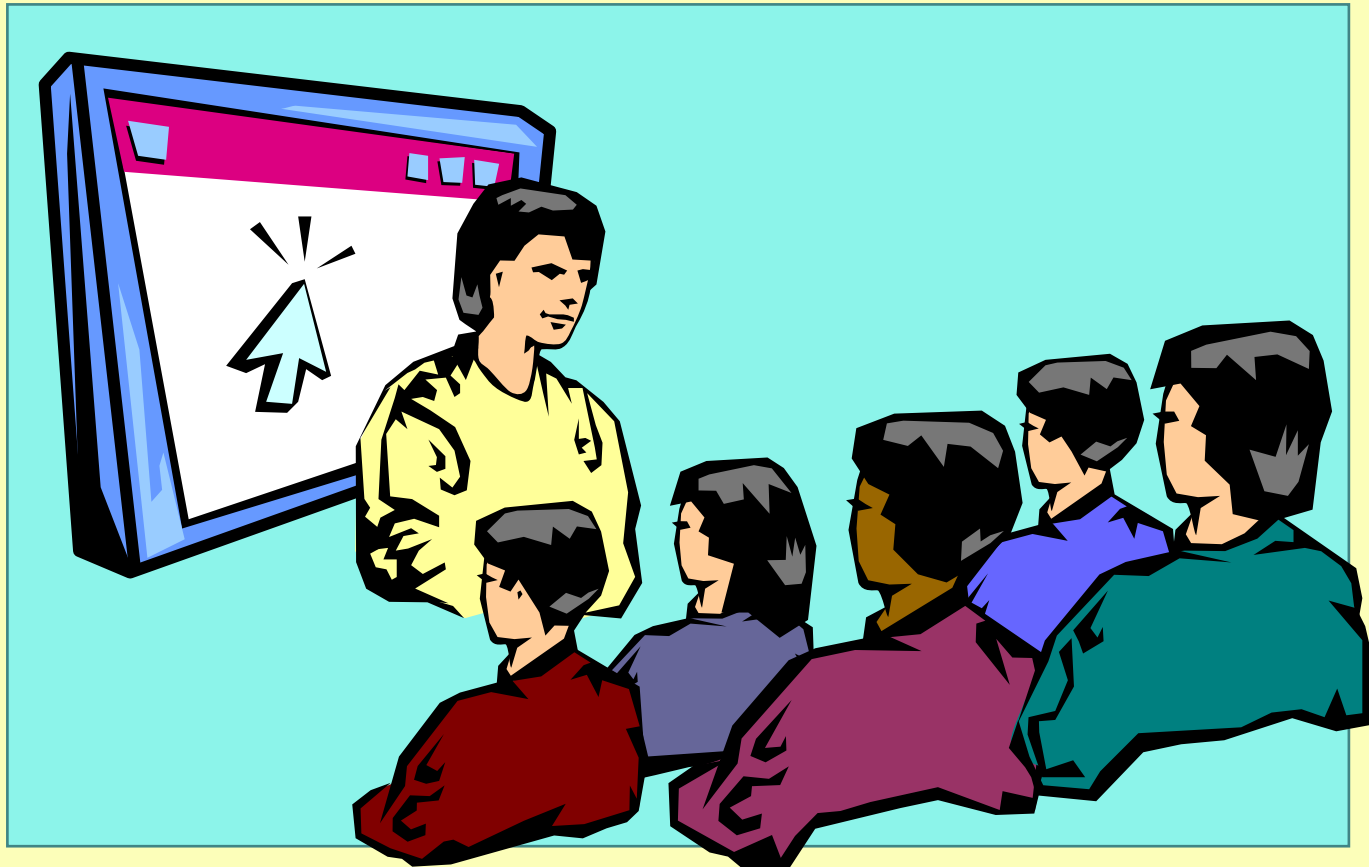
```
If MsgBox("Continue?", MsgBoxStyle.YesNo +  
MsgBoxStyle.Question, "Question") = MsgBoxResult.Yes Then  
    ...  
End If
```

## ■ MessageBox Class

```
If MessageBox.Show("Continue?", "Question",  
MessageBoxButtons.YesNo, MessageBoxIcon.Question)  
= DialogResult.Yes Then  
    ...  
End If
```

## ■ InputBox

# Demonstration: Manipulating Windows Forms

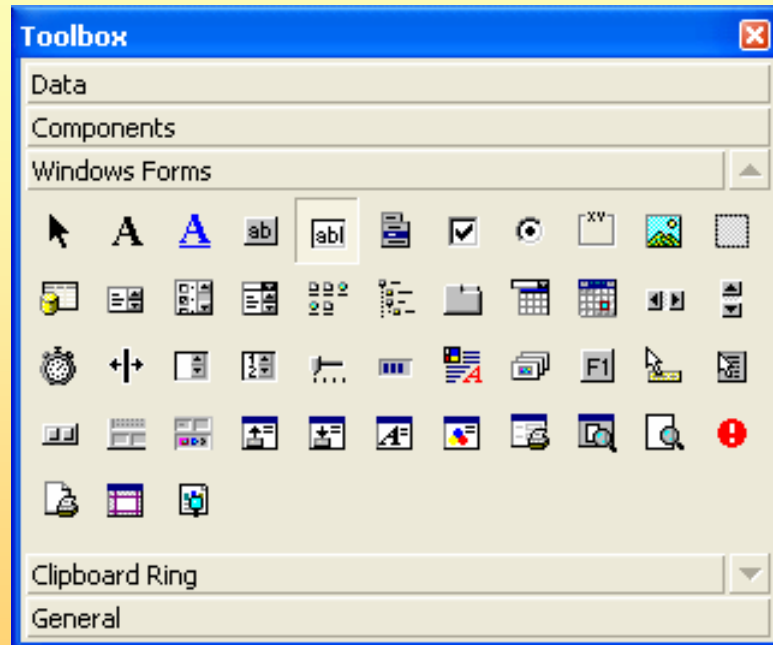


# ◆ Using Controls

- New Controls
- Using Control Properties
- Using Control Methods
- Creating Menus
- Providing User Help
- Implementing Drag-and-Drop Functionality

# New Controls

- CheckedListBox
- LinkLabel
- Splitter
- ToolTip
- NotifyIcon



# Using Control Properties

- Positioning
  - Anchor
  - Location
- Text property

```
Button1.Text = "Click Me"
```

# Using Control Methods

- BringToFront and SendToBack

```
Button1.BringToFront( )  
Button2.SendToBack( )
```

- Focus

```
TextBox1.Focus( )  
TextBox1.SelectAll( )
```

# Creating Menus

- Menu classes
- Creating menus at design time
  - Use the Menu Designer
- Creating menus at run time

```
Dim mnuMain As New MainMenu( )  
Dim mnuItem1 As New MenuItem, mnuItem2 As New MenuItem( )  
mnuItem1.Text = "File"  
mnuMain.MenuItems.Add(mnuItem1)  
mnuItem2.Text = "Exit"  
mnuMain.MenuItems(0).MenuItems.Add(mnuItem2)  
AddHandler mnuItem2.Click, AddressOf NewExitHandler  
Menu = mnuMain
```

# Providing User Help

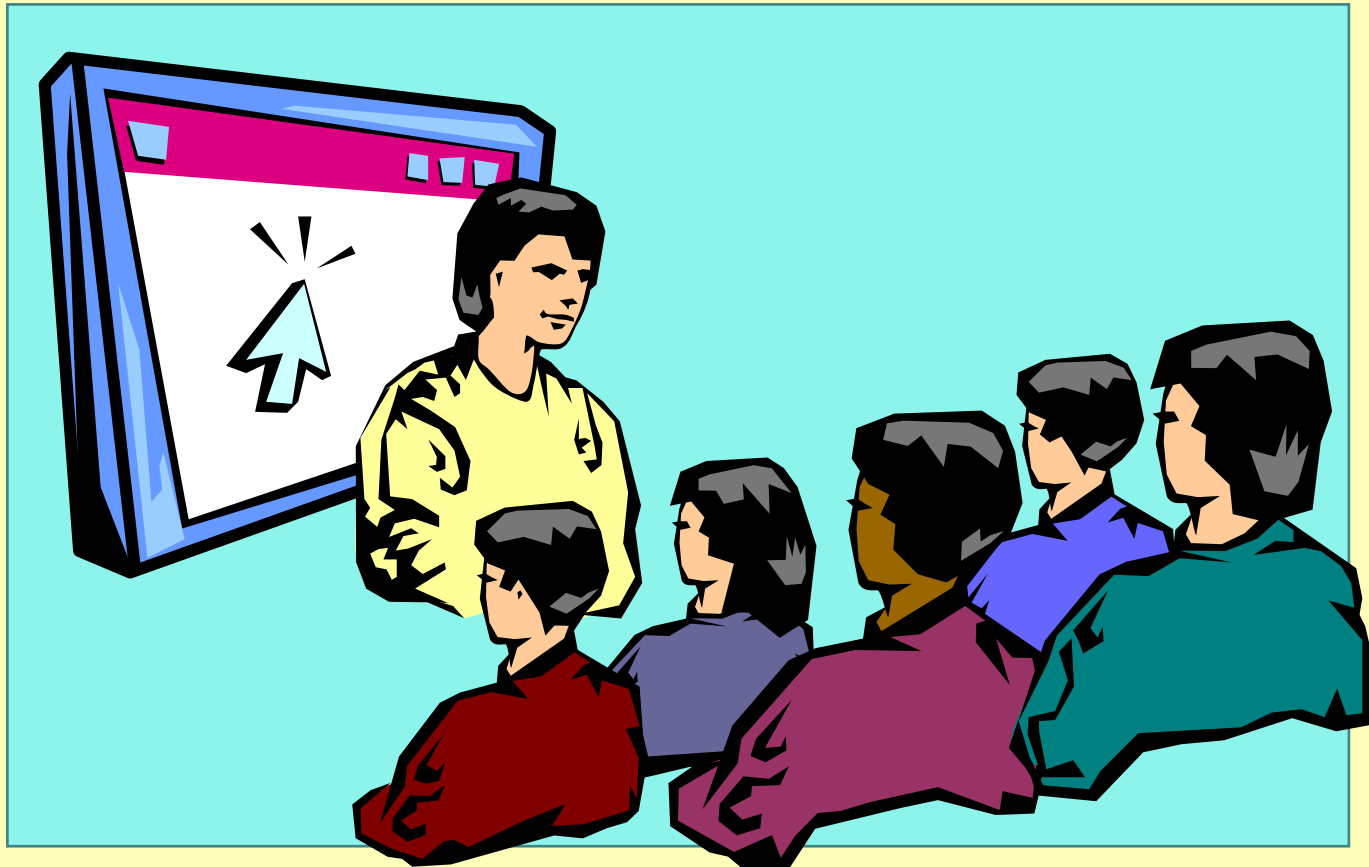
## ■ ErrorProvider control

- Error icon appears next to control, and message appears like a ToolTip when mouse pauses over icon
- Used mainly for data binding

## ■ HelpProvider control

- Points to .chm, .hlp, or .html Help file
- Controls provide Help information by means of **HelpString** or **HelpTopic** properties

# Demonstration: Using Controls



# Implementing Drag-and-Drop Functionality

## ■ Starting the process

- Use the **DoDragDrop** method in the **MouseDown** event of the originating control

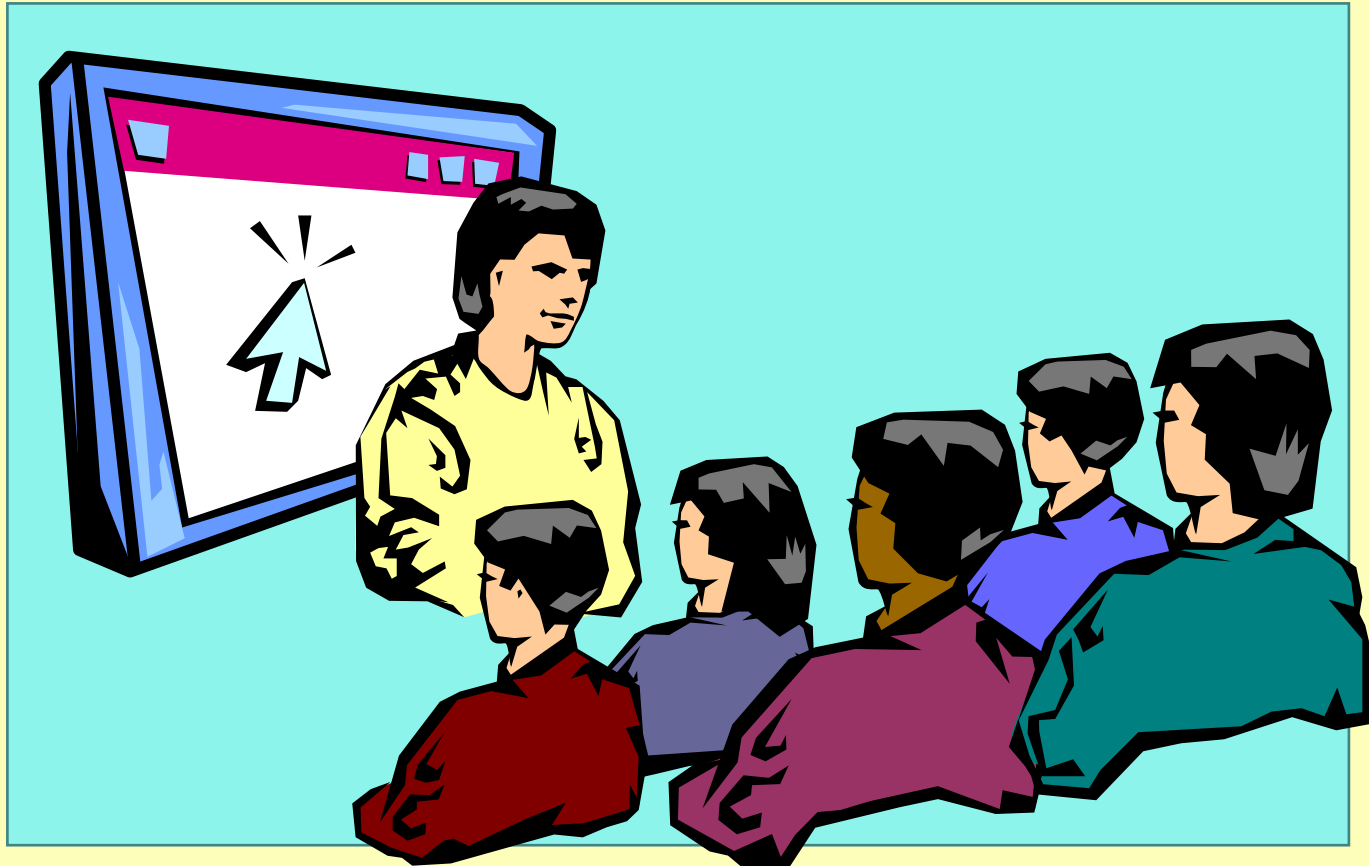
## ■ Changing the drag icon

- Set the **AllowDrop** property of the receiving control to **True**
- Set the **Effect** property of the **DragEventArgs** in the **DragOver** event of the receiving control

## ■ Dropping the data

- Use the **Data.GetData** method to access the data

# Demonstration: Implementing Drag-and-Drop Functionality



# ◆ Windows Forms Inheritance

- Why Inherit from a Form?
- Creating the Base Form
- Creating the Inherited Form
- Modifying the Base Form

# Why Inherit from a Form?

- A form is a class, so it can use inheritance
- Applications will have a standard appearance and behavior
- Changes to the base form will be applied to derived forms
- Common examples:
  - Wizard forms
  - Logon forms

# Creating the Base Form

1. Carefully plan the base form
2. Create the base form as for a normal form
3. Set the access modifiers property of controls
  - Private – Control can only be modified in the base form
  - Protected – Control can be modified by deriving form
  - Public – Control can be modified by any code module
  - Friend – Control can be modified within the base form project
4. Add the Overridable keyword to appropriate methods
5. Build the solution for the base form

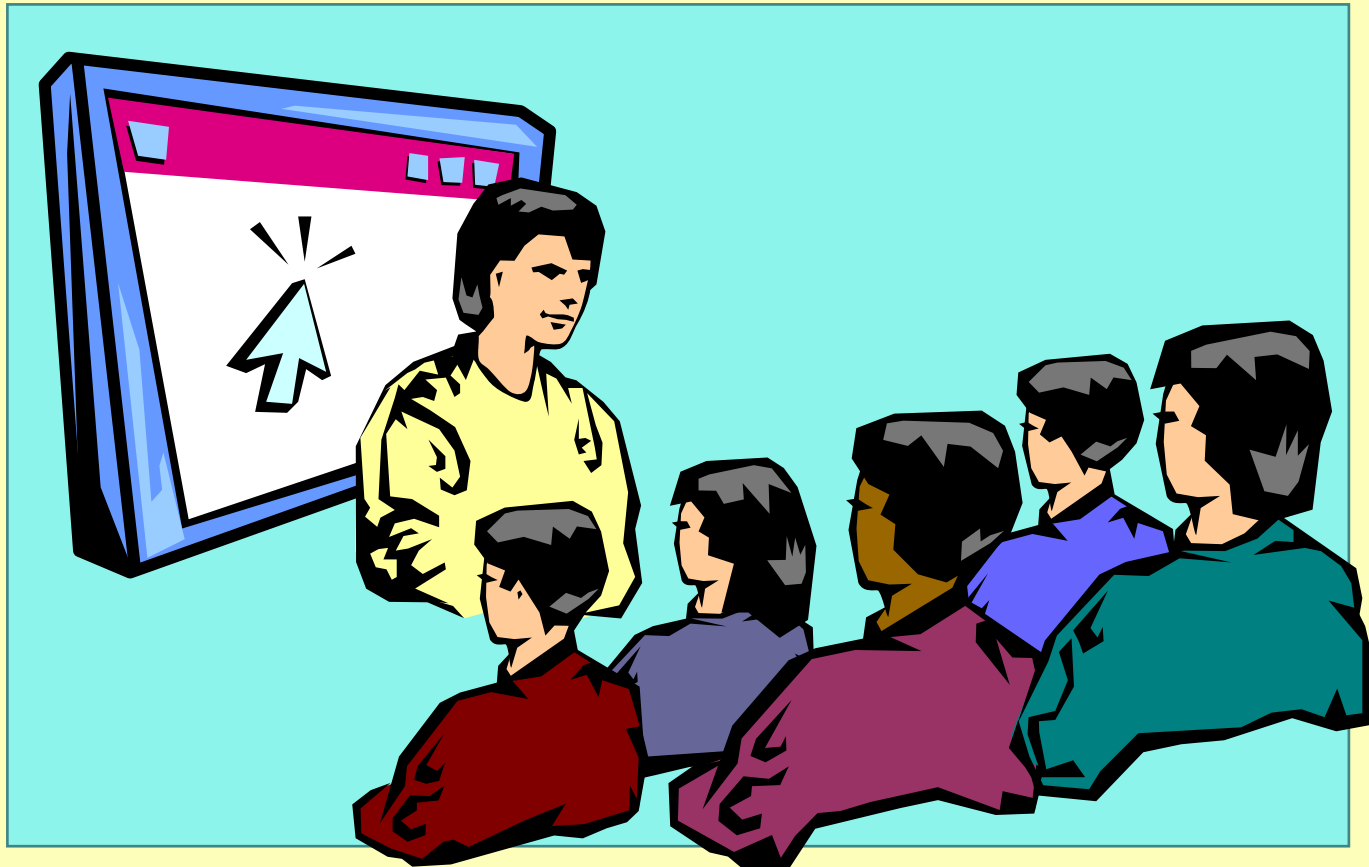
# Creating the Inherited Form

- Ensure that the base form is as complete as possible
- Reference the assembly
- Create a new Inherited Form item
- Change control properties where necessary
- Override methods or events as required

# Modifying the Base Form

- Changing the base form
  - Changes affect derived forms when rebuilt
- Checking derived forms
  - Verify changes before rebuilding application
  - Retest after rebuilding application

# Demonstration: Using Windows Forms Inheritance



# Lab 6.1: Creating the Customer Form



# Review

- Why Use Windows Forms?
- Structure of Windows Forms
- Using Windows Forms
- Using Controls
- Windows Forms Inheritance