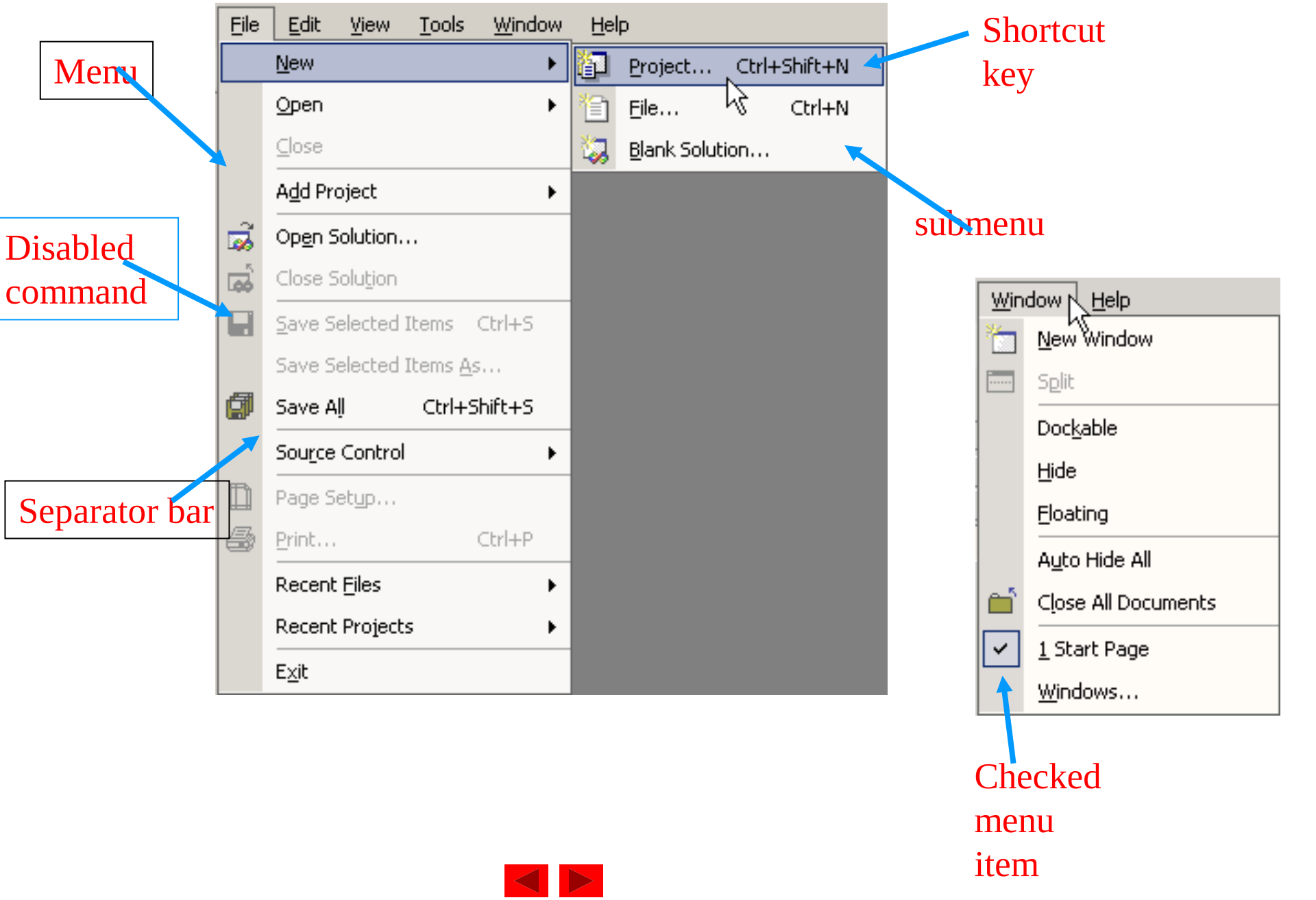
Chapter 13 – Graphical User Interfaces Part 1

Menus'terminology



13.1 Introduction

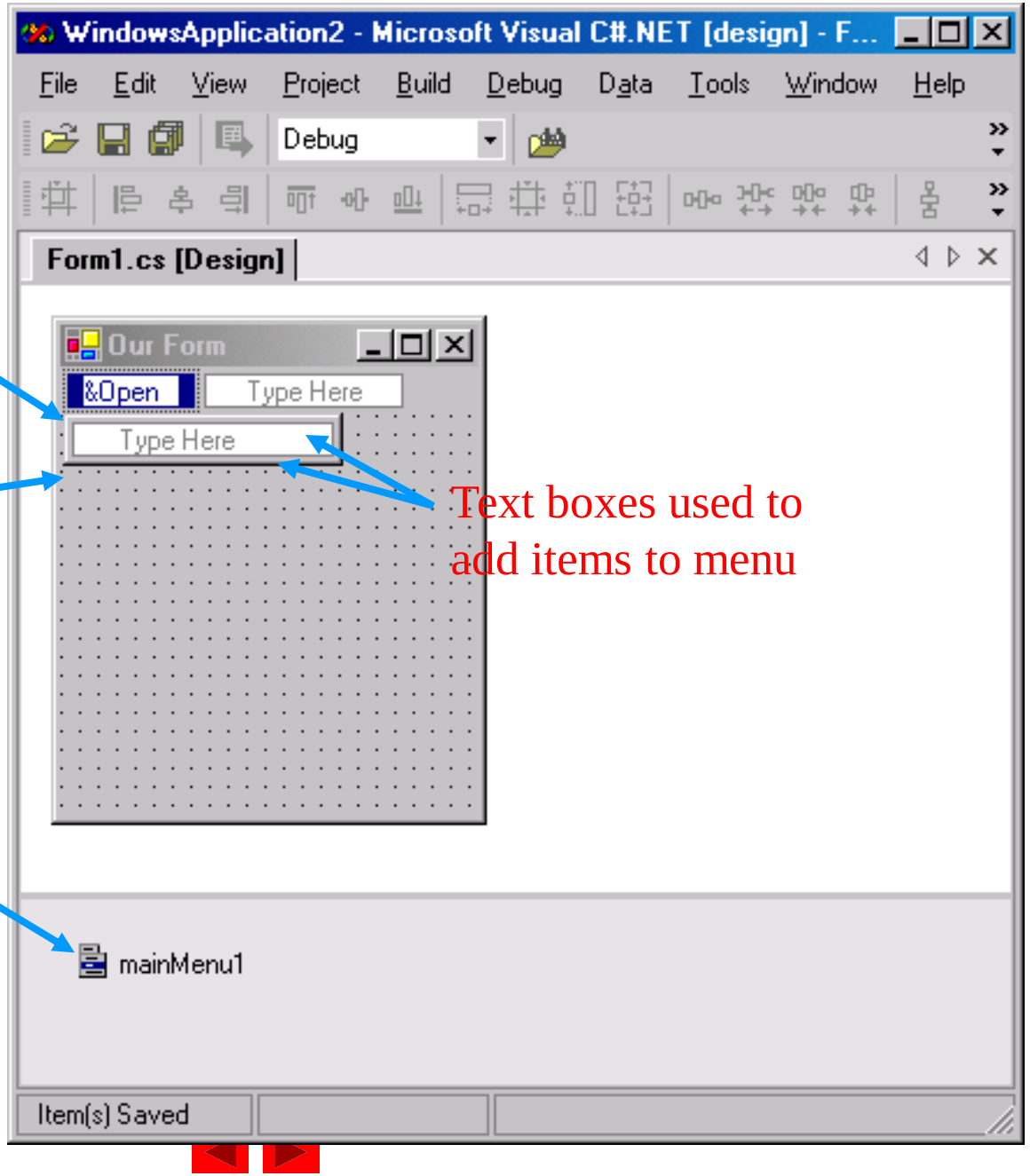
- Continues study of Graphical User Interface
- Explores:
  - Menus
  - **`LinkLabels`**
  - **`ListBox`**
  - **`CheckedListBox`**
  - **`ComboBoxes`**
  - **`TreeView`** control
  - Tab controls
  - Multiple-document interface windows

Menus'Visual programming (MenuScript)

Place & character before the letter to be underlined

Menu Designer

`MainMenu` icon



Text boxes used to add items to menu

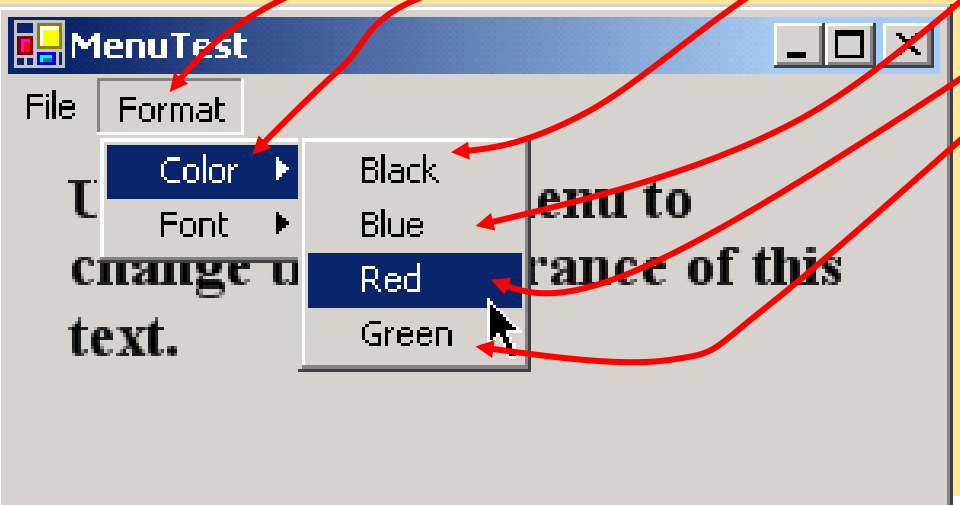
13.2 Menus

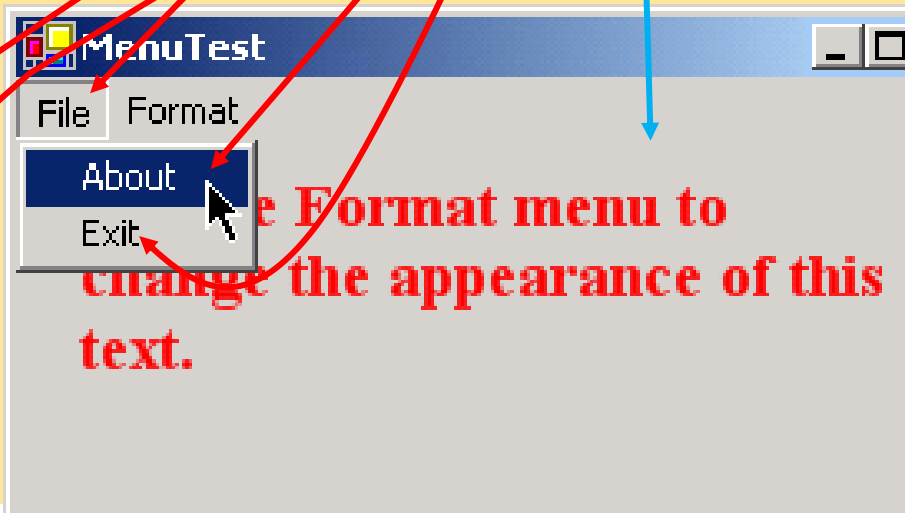
MainMenu and MenuItem events and properties	Description / Delegate and Event Arguments
***MainMenu** Properties*	
MenuItems	Lists the **MenuItems** that are contained in the **MainMenu**.
RightToLeft	Causes text to display from right to left. Useful for languages that are read from right to left.
***MenuItem** Properties*	
Checked	Indicates whether a menu item is checked (according to property **RadioCheck**). Default **False**, meaning that the menu item is not checked.
Index	Specifies an item's position in its parent menu.
MenuItems	Lists the submenu items for a particular menu item.
MergeOrder	Sets the position of a menu item when its parent menu is merged with another menu.
MergeType	Takes a value of the **MenuMerge** enumeration. Specifies how a parent menu merges with another menu. Possible values are **Add**, **MergeItems**, **Remove** and **Replace**.
RadioCheck	Indicates whether a selected menu item appears as a radio button (black circle) or displays a checkmark. **True** creates radio button, **False** displays checkmark; default **False**.
Shortcut	Specifies the shortcut key for the menu item (e.g., *Ctrl + F9* can be equivalent to clicking a specific item).

13.2 Menus

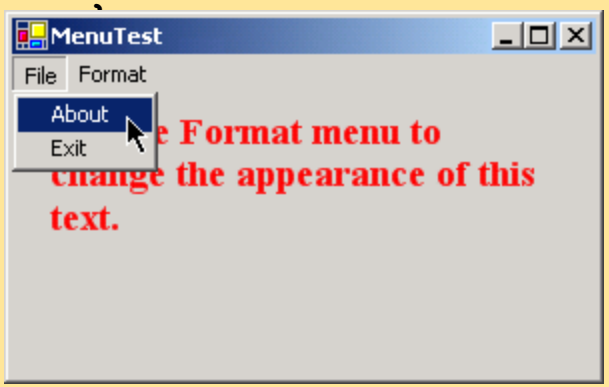
`MainMenu` and `MenuItem` events and properties	Description / Delegate and Event Arguments
`ShowShortcut`	Indicates whether a shortcut key is shown beside menu item text. Default is `True`, which displays the shortcut key.
`Text`	Specifies the text to appear in the menu item. To create an *Alt* access shortcut, precede a character with `&` (e.g., `&File` for **File**).
Common Event	*(Delegate* ***`EventHandler`****, event arguments* ***`EventArgs`****)*
`Click`	Generated when item is clicked or shortcut key is used. Default when double-clicked in designer.

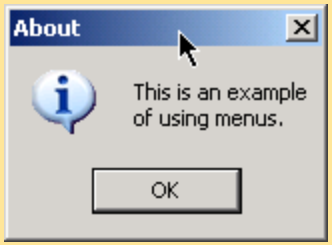
```
3     Using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     public class MenuTest : System.Windows.Forms.Form {
12       private System.Windows.Forms.Label displayLabel;
14       private System.Windows.Forms.MainMenu mainMenu;
16       private System.Windows.Forms.MenuItem fileMenuItem;
17       private System.Windows.Forms.MenuItem aboutMenuItem;
18       private System.Windows.Forms.MenuItem exitMenuItem;
20       private System.Windows.Forms.MenuItem formatMenuItem;
22       private System.Windows.Forms.MenuItem colorMenuItem;
23       private System.Windows.Forms.MenuItem blackMenuItem;
24       private System.Windows.Forms.MenuItem blueMenuItem;
25       private System.Windows.Forms.MenuItem redMenuItem;
26       private System.Windows.Forms.MenuItem greenMenuItem;
```



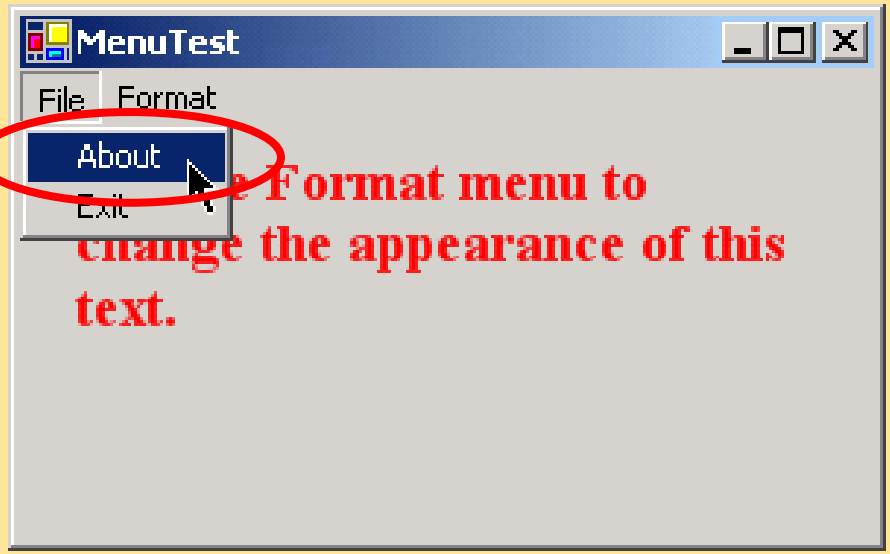


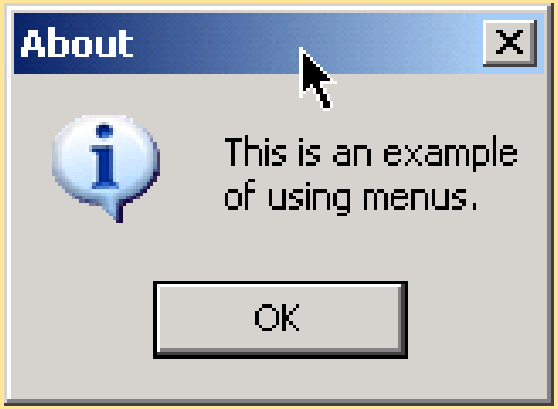
```
28      private System.Windows.Forms.MenuItem timesMenuItem;
29      private System.Windows.Forms.MenuItem courierMenuItem;
30      private System.Windows.Forms.MenuItem comicMenuItem;
31      private System.Windows.Forms.MenuItem boldMenuItem;
32      private System.Windows.Forms.MenuItem italicMenuItem;
33      private System.Windows.Forms.MenuItem fontMenuItem;
34      private System.Windows.Forms.MenuItem separatorMenuItem;
35      [STAThread]
36      static void Main() {
38         Application.Run( new MenuTest() );
39      }
41      private void aboutMenuItem_Click(
42         object sender, System.EventArgs e ) {
44         MessageBox.Show("This is an example\nof using menus.",
46            "About", MessageBoxButtons.OK, MessageBoxIcon.Information );
48      }
50      private void exitMenuItem_Click(
51         object sender, System.EventArgs e ) {
53         Application.Exit();
54      }
```





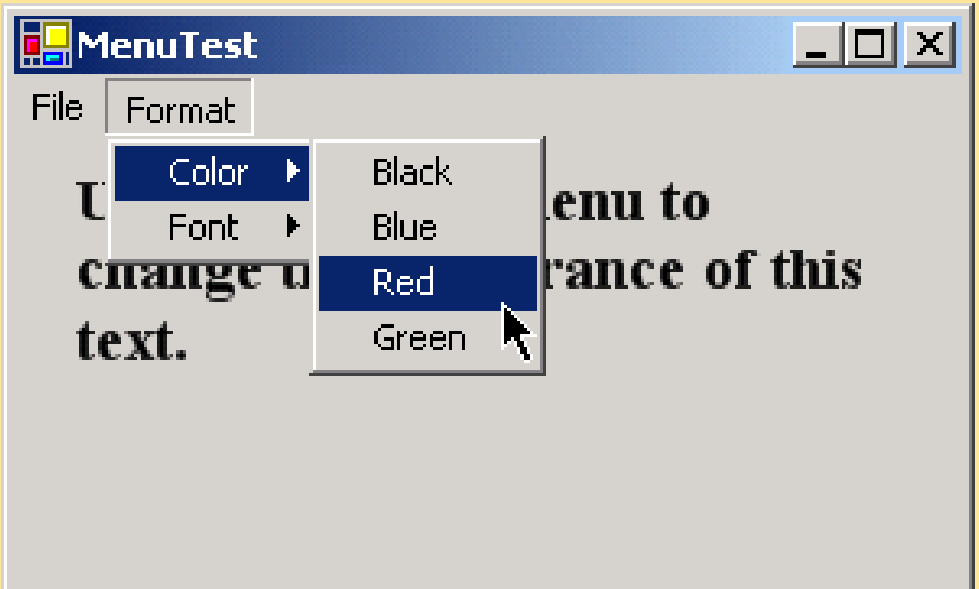
```
28      private System.Windows.Forms.MenuItem timesMenuItem;
29      private System.Windows.Forms.MenuItem courierMenuItem;
30      private System.Windows.Forms.MenuItem comicMenuItem;
31      private System.Windows.Forms.MenuItem boldMenuItem;
32      private System.Windows.Forms.MenuItem italicMenuItem;
33      private System.Windows.Forms.MenuItem fontMenuItem;
34      private System.Windows.Forms.MenuItem separatorMenuItem;
35      [STAThread]
36      static void Main() {
38         Application.Run( new MenuTest() );
39      }
41      private void aboutMenuItem_Click(
42         object sender, System.EventArgs e ) {
44         MessageBox.Show("This is an example\nof using menus.",
46            "About", MessageBoxButtons.OK, MessageBoxIcon.Information );
48      }
```





```
50      private void exitMenuItem_Click(
51          object sender, System.EventArgs e ) {
53          Application.Exit();
54      }
56      private void ClearColor() {
59          blackMenuItem.Checked = false;
60          blueMenuItem.Checked = false;
61          redMenuItem.Checked = false;
62          greenMenuItem.Checked = false;
63      }
65      private void blackMenuItem_Click(
66          object sender, System.EventArgs e ){
69          ClearColor();
71          displayLabel.ForeColor = Color.Black;
72          blackMenuItem.Checked = true;
73      }
```

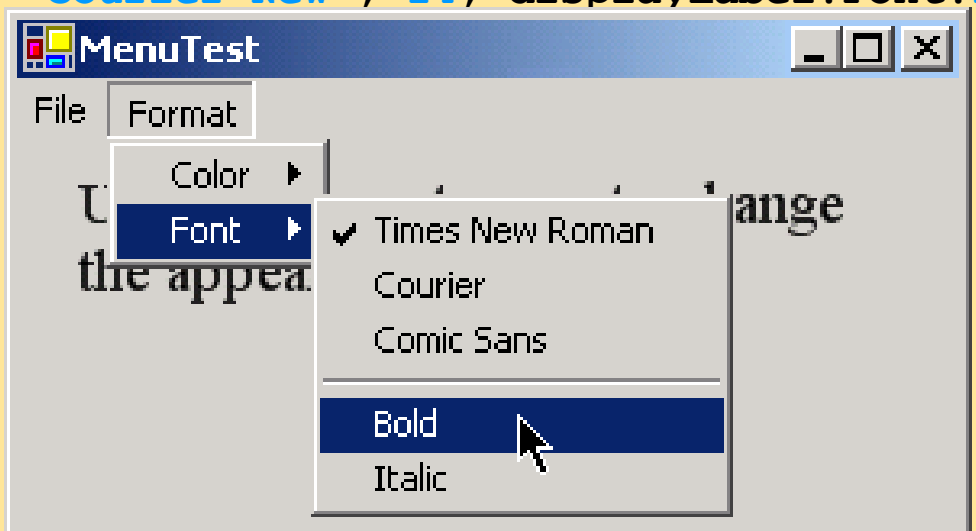
User's Function called by User program



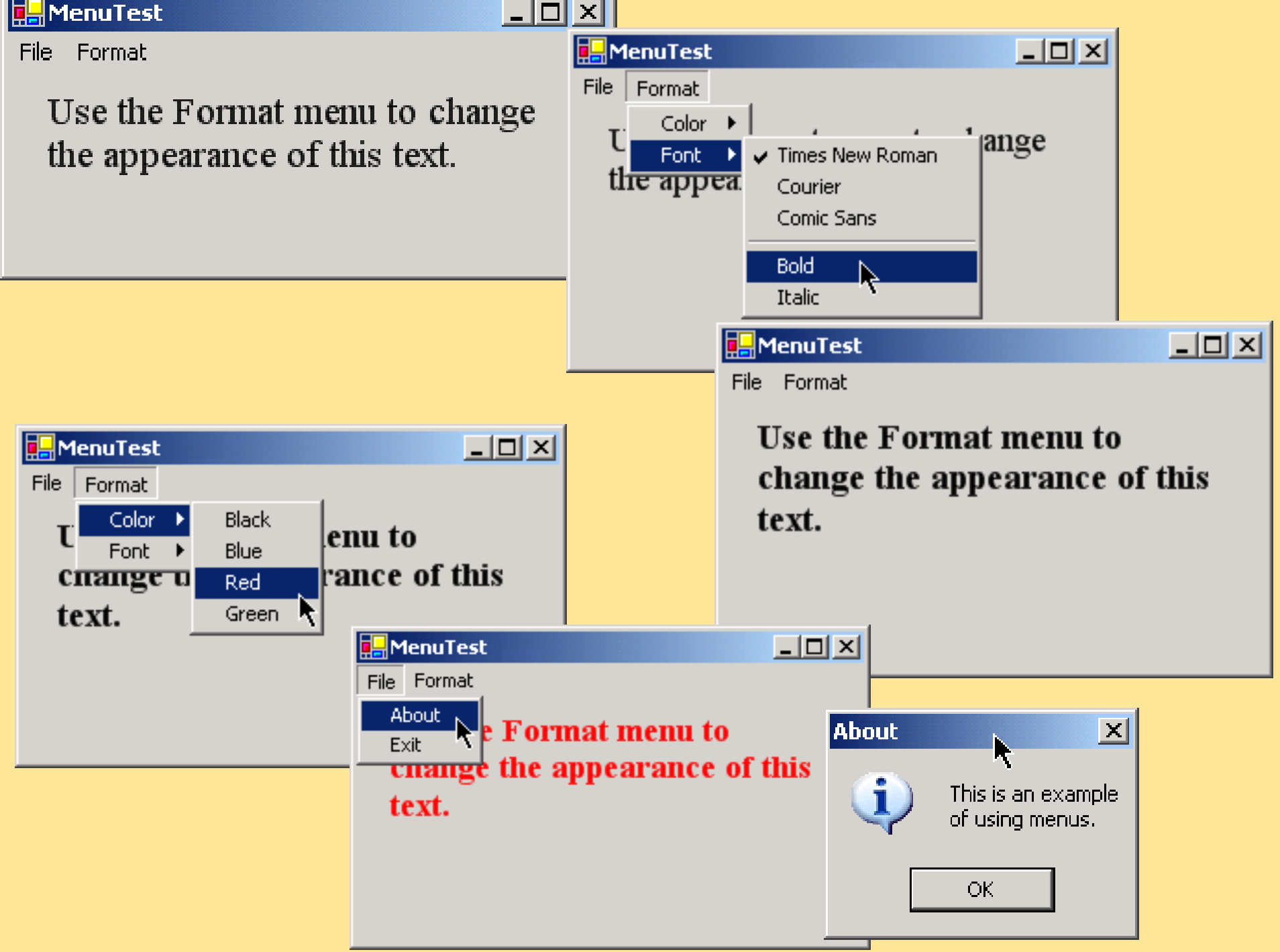
```
75      private void blueMenuItem_Click(
76         object sender, System.EventArgs e ) {
79         ClearColor();
81         displayLabel.ForeColor = Color.Blue;
82         blueMenuItem.Checked = true;
83      }
85      private void redMenuItem_Click(
86         object sender, System.EventArgs e ) {
89         ClearColor();
91        displayLabel.ForeColor = Color.Red;
92         redMenuItem.Checked = true;
93      }
95      private void greenMenuItem_Click(
96         object sender, System.EventArgs e ){
99         ClearColor();
101        displayLabel.ForeColor = Color.Green;
102        greenMenuItem.Checked = true;
103     }
```

```
105     private void ClearFont() {
108        timesMenuItem.Checked = false;
109        courierMenuItem.Checked = false;
110        comicMenuItem.Checked = false;
111     }
113     private void timesMenuItem_Click(
114        object sender, System.EventArgs e ) {
117        ClearFont();
119        timesMenuItem.Checked = true;
120        displayLabel.Font = new Font(
121           "Times New Roman", 14, displayLabel.Font.Style );
122     }
124     private void courierMenuItem_Click(
125        object sender, System.EventArgs e ) {
128        ClearFont();
130        courierMenuItem.Checked = true;
131        displayLabel.Font = new Font(
132           "Courier New", 14, displayLabel.Font.Style );
```

User's Function called by User program

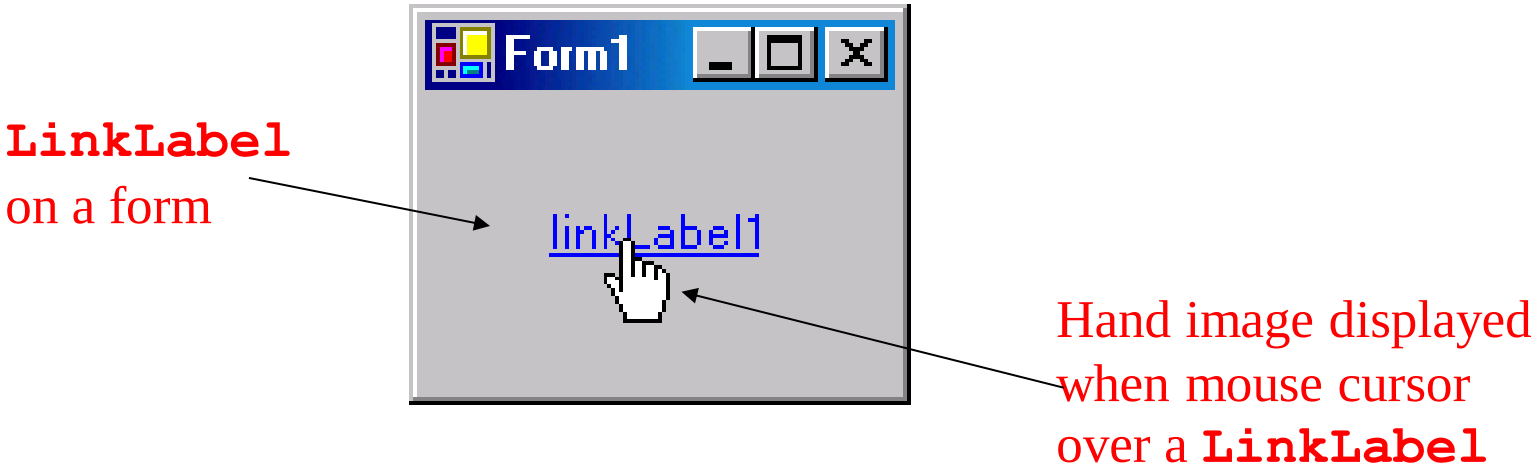


```
133     }
135     private void comicMenuItem_Click(
136        object sender, System.EventArgs e ) {
139        ClearFont();
141        comicMenuItem.Checked = true;
142        displayLabel.Font = new Font(
143           "Comic Sans MS", 14, displayLabel.Font.Style );
144     }
146     private void boldMenuItem_Click(
147        object sender, System.EventArgs e ) {
150        boldMenuItem.Checked = !boldMenuItem.Checked;
152        displayLabel.Font = new Font( displayLabel.Font.FontFamily, 14,
154           displayLabel.Font.Style ^ FontStyle.Bold );
155     }
157     private void italicMenuItem_Click(
158        object sender, System.EventArgs e) {
161        italicMenuItem.Checked = !italicMenuItem.Checked;
163        displayLabel.Font = new Font( displayLabel.Font.FontFamily, 14,
165           displayLabel.Font.Style ^ FontStyle.Italic );
166     }
167  }
```


MenuTest
File Format
Use the Format menu to change the appearance of this text.
MenuTest
File Format
Color
Font
Times New Roman
Courier
Comic Sans
Bold
Italic
MenuTest
File Format
Use the Format menu to change the appearance of this text.
MenuTest
File Format
Color
Font
Black
Blue
Red
Green
MenuTest
File Format
About
Exit
About
This is an example of using menus.
OK

13.3 `LinkLabels`

- Displays links to other objects
  - Uses **`event handlers`** to link to right file or program
  - **`Start`** method of **`Process`** class opens other programs
- Derived from class **`Label`**, inherits functionality

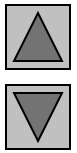


13.3 LinkLabels

LinkLabel properties and events	Description / Delegate and Event Arguments
Common Properties	
`ActiveLinkColor`	Specifies the color of the active link when clicked. Default is red.
`LinkArea`	Specifies which portion of text in the `LinkLabel` is treated as part of the link.
`LinkBehavior`	Specifies the link's behavior, such as how the link appears when the mouse is placed over it.
`LinkColor`	Specifies the original color of all links before they have been visited. Default is blue.
`Links`	Lists the `LinkLabel.Link` objects, which are the links contained in the `LinkLabel`.
`LinkVisited`	If `True`, link appears as if it were visited (its color is changed to that specified by property `VisitedLinkColor`). Default `False`.
`Text`	Specifies the text to appear on the control.
`UseMnemonic`	If `True`, `&` character in `Text` property acts as a shortcut (similar to the *Alt* shortcut in menus).
`VisitedLinkColor`	Specifies the color of visited links. Default is `Color.Purple`.

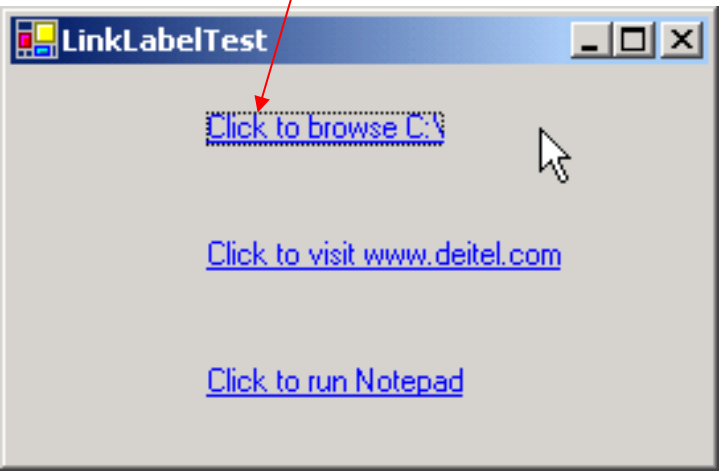
`LinkLabel` properties and events	Description / Delegate and Event Arguments
Common Event	*(Delegate* ***`LinkLabelLinkClickedEventHandler`****, event arguments* ***`LinkLabelLinkClickedEventArgs`****)*
`LinkClicked`	Generated when link is clicked. Default when control is double-clicked in designer.

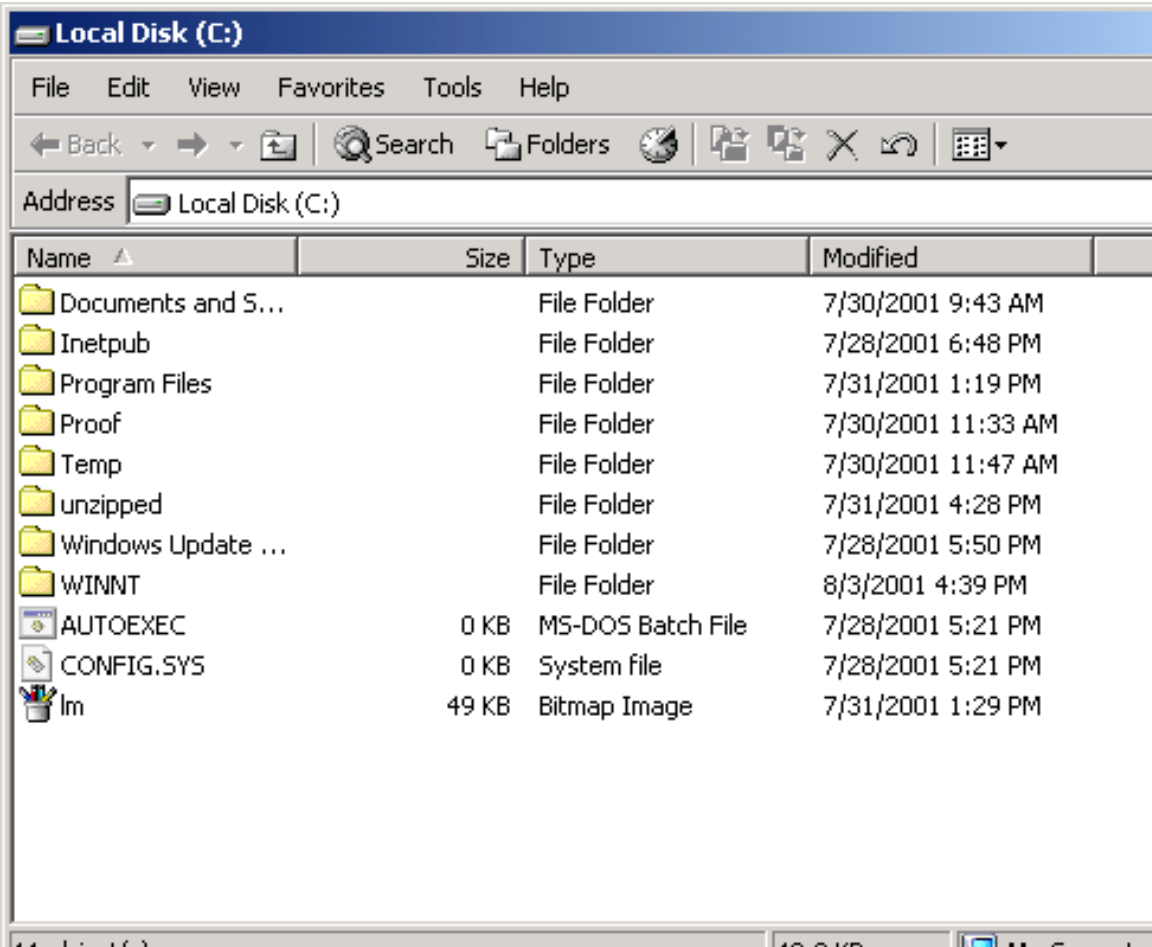
```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     public class LinkLabelTest : System.Windows.Forms.Form {
12       private System.Windows.Forms.LinkLabel driveLinkLabel;
13       private System.Windows.Forms.LinkLabel deitelLinkLabel;
14       private System.Windows.Forms.LinkLabel notepadLinkLabel;
15       [STAThread]
16       static void Main() {
18          Application.Run( new LinkLabelTest() );
19       }
21       private void driveLinkLabel_LinkClicked( object sender,
22          System.Windows.Forms.LinkLabelLinkClickedEventArgs e){
24          driveLinkLabel.LinkVisited = true;
25          System.Diagnostics.Process.Start( "C:\\" );
26       }
```



`LinkLabelTest.cs`
Program Output

Click on first **`LinkLabel`** to look at contents of C drive

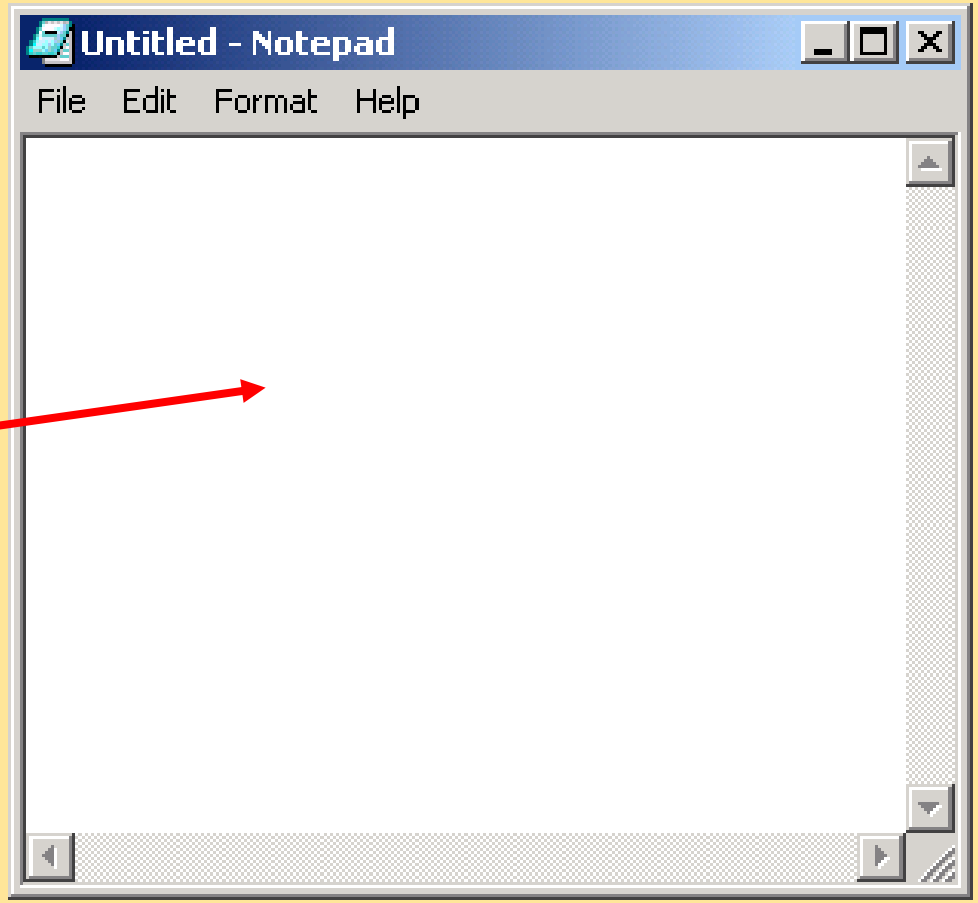




```
28      private void deitelLinkLabel_LinkClicked( object sender,
29         System.Windows.Forms.LinkLabelLinkClickedEventArgs e){
31         deitelLinkLabel.LinkVisited = true;
32         System.Diagnostics.Process.Start(
33            "IExplore", "http://www.deitel.com" );
34      }
```

Click on second **LinkLabel** to go to the Web Site

```
36      private void notepadLinkLabel_LinkClicked(
37         object sender,
38         System.Windows.Forms.LinkLabelLinkClickedEventArgs e){
40         notepadLinkLabel.LinkVisited = true;
43         System.Diagnostics.Process.Start( "notepad" );
44      }
45   }
```



Click the third **`LinkLabel`** to open notepad

13.4 ListBoxes and CheckListBoxes

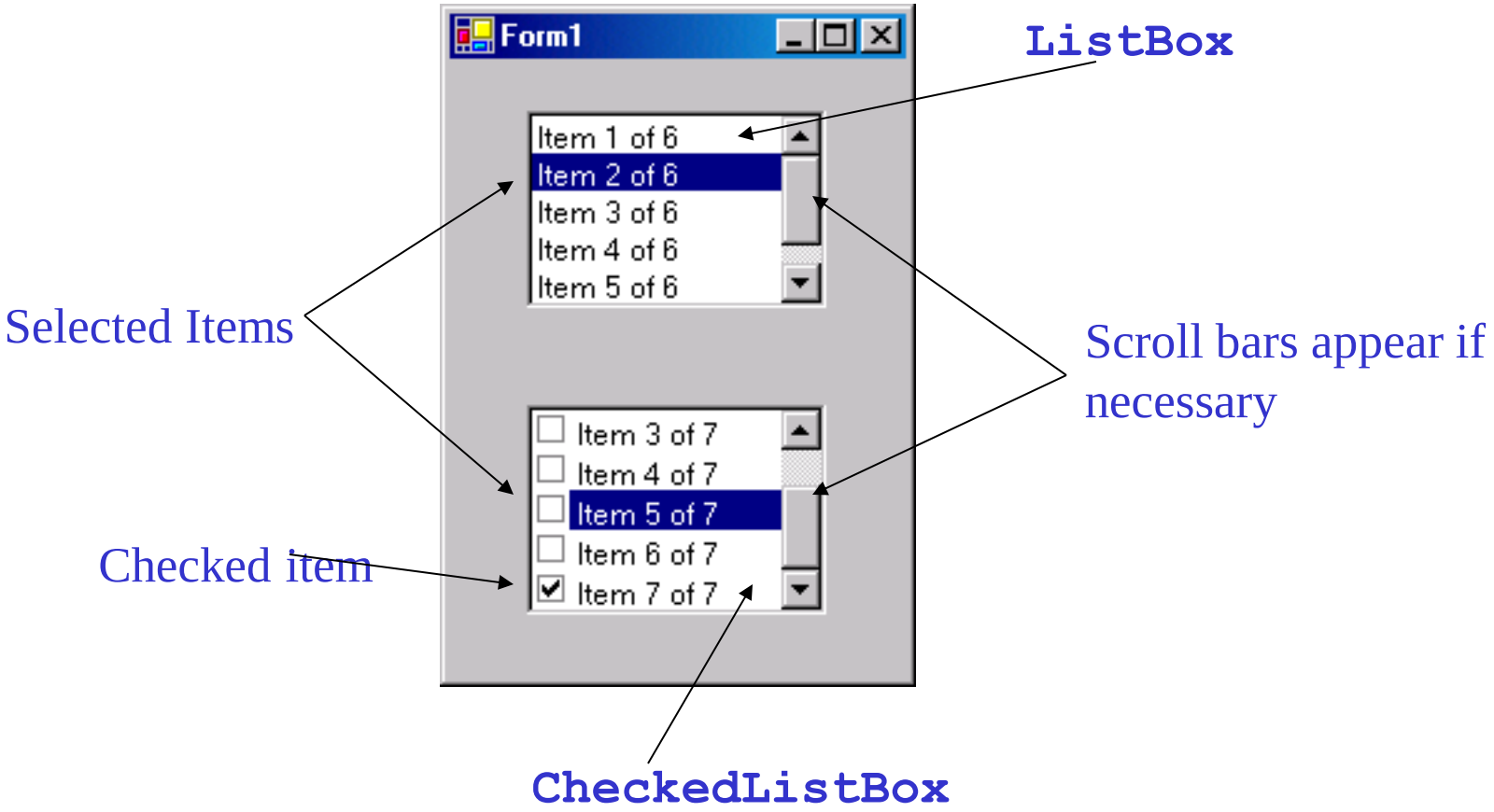
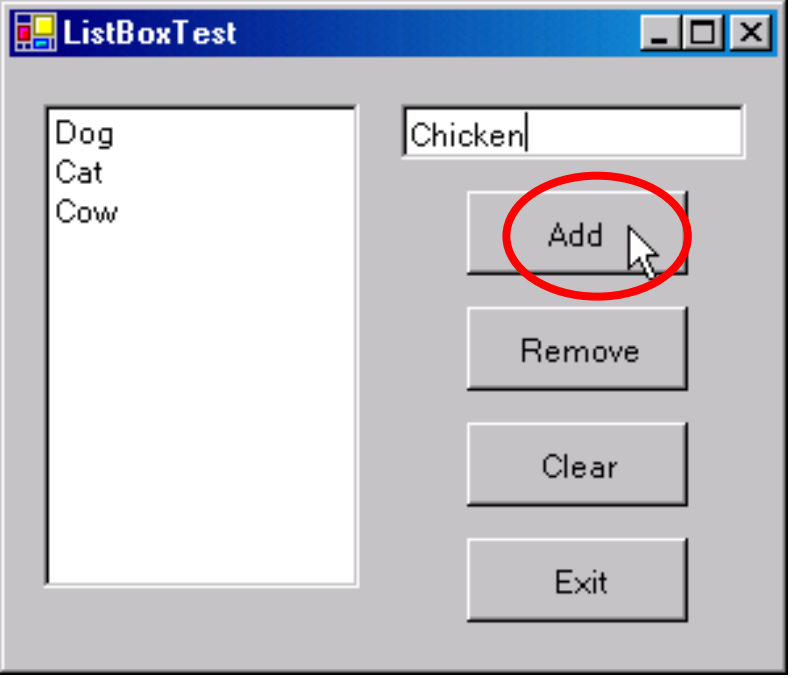


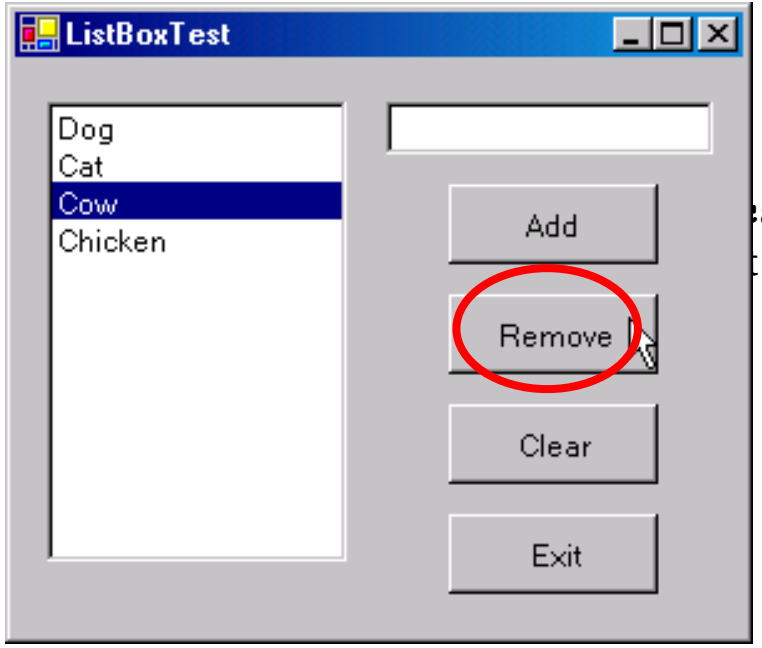
Fig. 13.8 **`ListBox`** and **`CheckedListBox`** on a form.

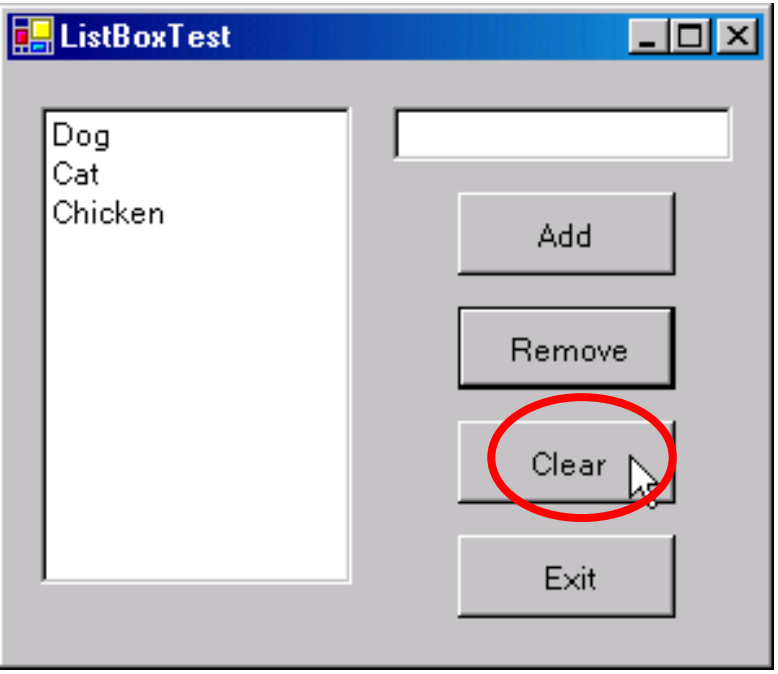
13.4 ListBoxes and CheckedListBoxes

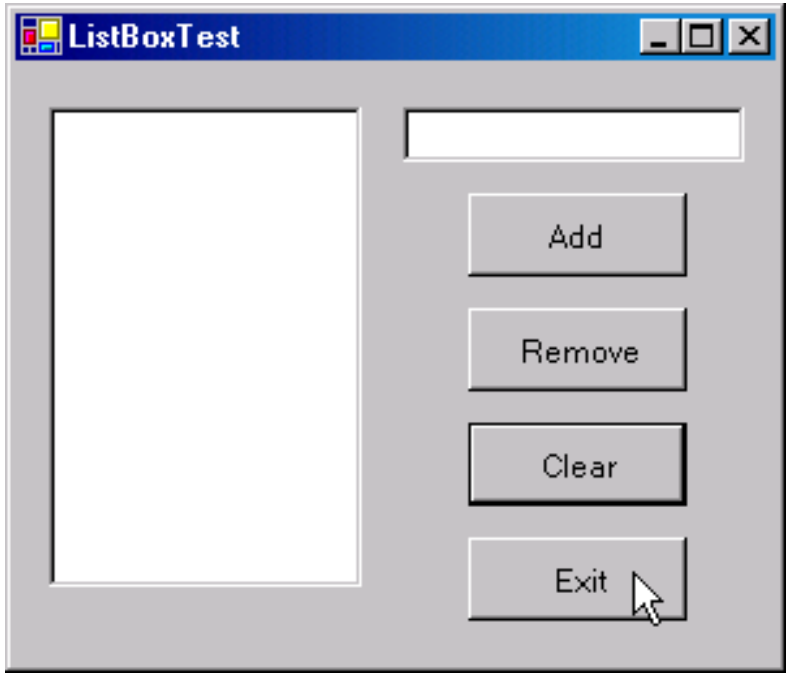
- **ListBoxes**
  - Allow users to view and select from items on a list
  - Static objects
  - **SelectionMode** property determines number of items that can be selected
  - Property **Items** returns all objects in list
  - Property **SelectedItem** returns current selected item
  - Property **SelectedIndex** returns index of selected item
  - Property **GetSelected** returns true if property at given index is selected
  - Use **Add** method to add to **Items** collection
    - myListBox.**Items.Add**("myListItem")

ListBox properties, methods and events	Description / Delegate and Event Arguments
Common Properties	
Items	Lists the collection of items within the **ListBox**.
MultiColumn	Indicates whether the **ListBox** can break a list into multiple columns. Multiple columns are used to make vertical scroll bars unnecessary.
SelectedIndex	Returns the index of the currently selected item. If the user selects multiple items, this method arbitrarily returns one of the selected indices. If no items have been selected, the method returns **-1**.
SelectedIndices	Returns a collection of the indices of all currently selected items.
SelectedItem	Returns a reference to the currently selected item (if multiple items are selected, it returns the item with the lowest index number).
SelectedItems	Returns a collection of the currently selected item(s).
SelectionMode	Determines the number of items that can be selected and the means through which multiple items can be selected. Values **None**, **One**, **MultiSimple** (multiple selection allowed) and **MultiExtended** (multiple selection allowed via a combination of arrow keys, mouse clicks and *Shift* and *Control* buttons).
Sorted	Indicates whether items appear in alphabetical order. **True** causes alphabetization; default is **False**.









```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     public class ListBoxTest : System.Windows.Forms.Form {
12       private System.Windows.Forms.ListBox displayListBox;
14       private System.Windows.Forms.TextBox inputTextBox;
16       private System.Windows.Forms.Button addButton;
17       private System.Windows.Forms.Button removeButton;
18       private System.Windows.Forms.Button clearButton;
19       private System.Windows.Forms.Button exitButton;
20       [STAThread]
21       static void Main() {
23          Application.Run(
               new ListBoxTest());
24       }
```

```
27      private void addButton_Click(
28         object sender, System.EventArgs e ) {
30         displayListBox.Items.Add( inputTextBox.Text );
31         inputTextBox.Clear();
32      }
34      private void removeButton_Click(
35         object sender, System.EventArgs e ) {
38         if ( displayListBox.SelectedIndex != -1 )
39            displayListBox.Items.RemoveAt(
40               displayListBox.SelectedIndex );
41      }
43      private void clearButton_Click(
44         object sender, System.EventArgs e ) {
46         displayListBox.Items.Clear();
47      }
49      private void exitButton_Click(
50         object sender, System.EventArgs e ) {
52         Application.Exit();
53      }
54   }
```

Method: ListBox.ObjectCollection.RemoveAt(Int32)

Removes the item at the specified index within the collection.

13.4 ListBoxes and CheckedListBoxes

- **CheckedListBoxes**
  - Extends **ListBox** by placing check boxes next to items
  - Can select more than one object at one time

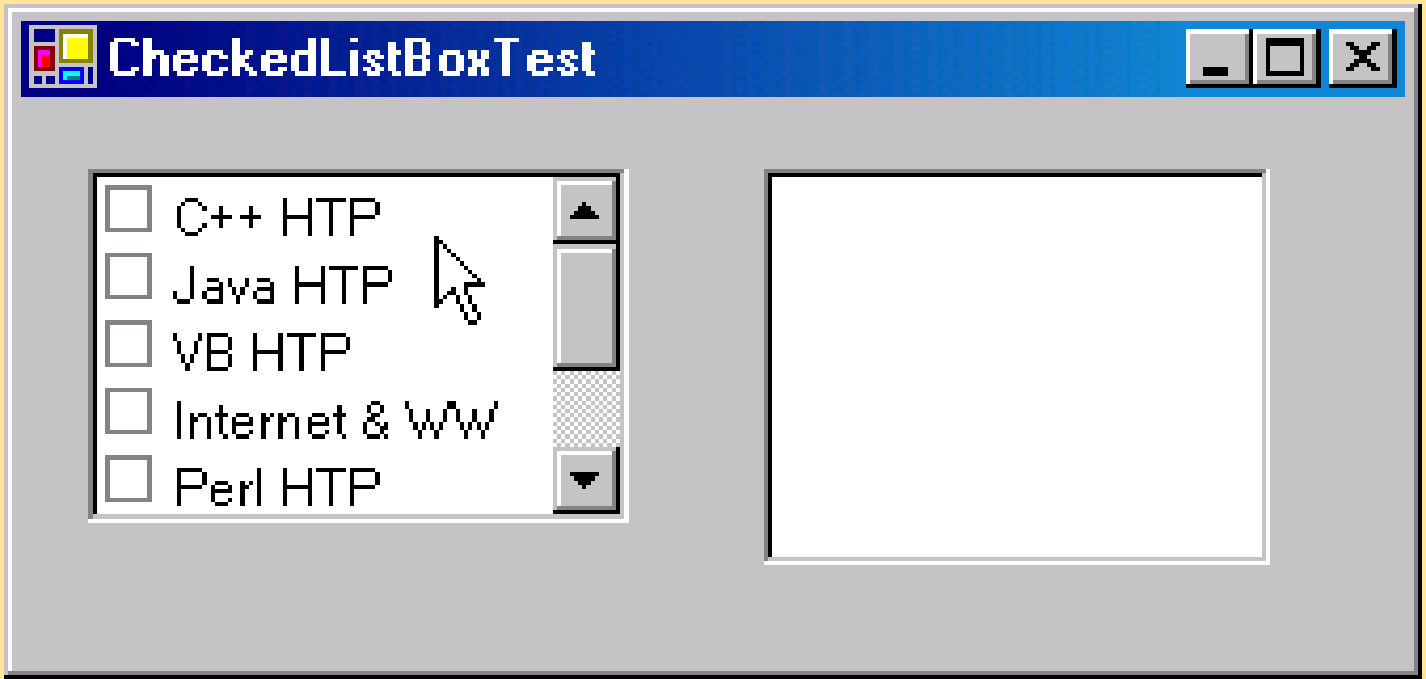
13.4.2 CheckedListBoxes

- **CheckedListBox** derives from class **ListBox**
  - Can add to, remove from or clear list
  - Can select multiple items from the list
  - Properties **CurrentValue** and **NewValue** return state of object selected
  - Properties **CheckedItems** and **CheckedIndices** return the objects and indices of selected items respectively

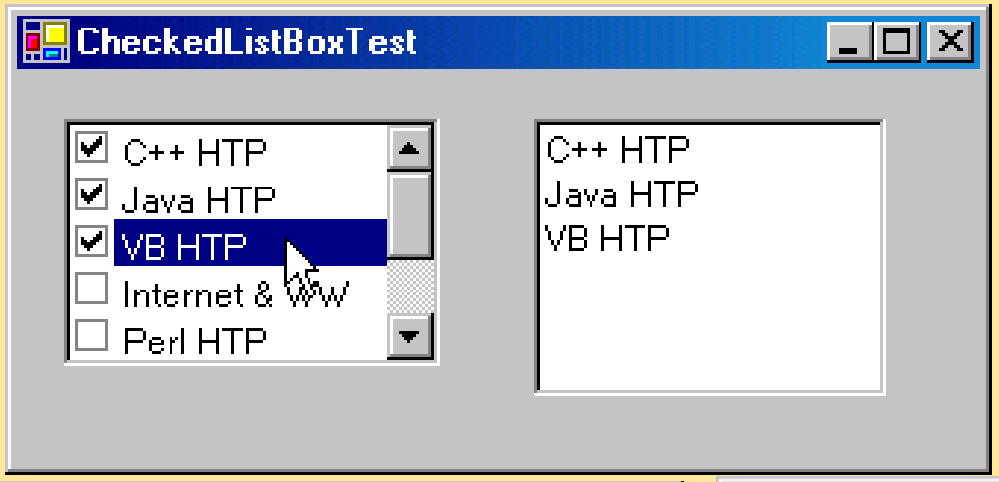
`CheckedListBox` properties, methods and events	Description / Delegate and Event Arguments
Common Properties	*(All the **`ListBox`** properties and events are inherited by **`CheckedListBox`**.)*
`CheckedItems`	Lists the collection of items that are checked. This is distinct from the selected items, which are highlighted (but not necessarily checked). *Note: There can be at most one selected item at any given time.*
`CheckedIndices`	Returns indices for the items that are checked. Not the same as the selected indices.
`SelectionMode`	Determines how many items can be checked. Only possible values are **`One`** (allows multiple checks to be placed) or **`None`** (does not allow any checks to be placed).
Common Method	
`GetItemChecked`	Takes an index, and returns **`True`** if corresponding item is checked.
Common Event	*(Delegate **`ItemCheckEventHandler`**, event arguments **`ItemCheckEventArgs`**)*
`ItemCheck`	Generated when an item is checked or unchecked.

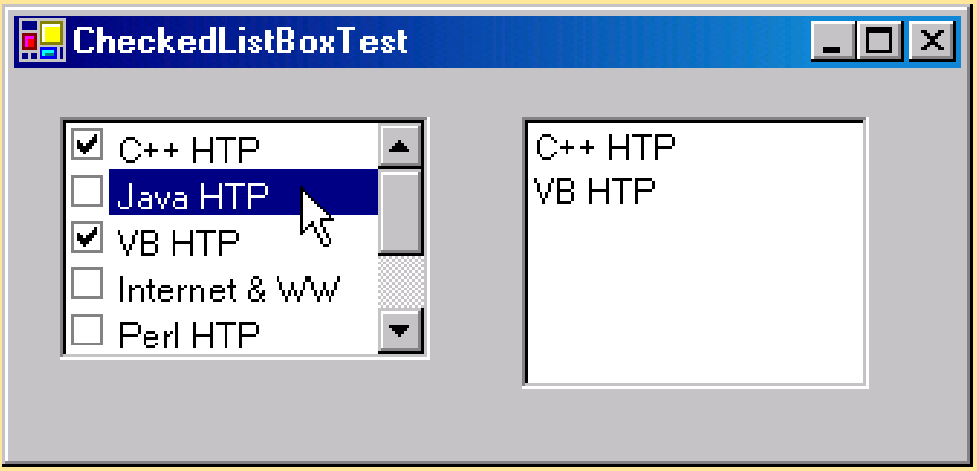
`CheckedListBox` properties, methods and events	Description / Delegate and Event Arguments
`ItemCheckEventArgs` *Properties*	
`CurrentValue`	Indicates whether current item is checked or unchecked. Possible values are **`Checked`**, **`Unchecked`** and **`Indeterminate`**.
`Index`	Returns index of the item that changed.
`NewValue`	Specifies the new state of item.

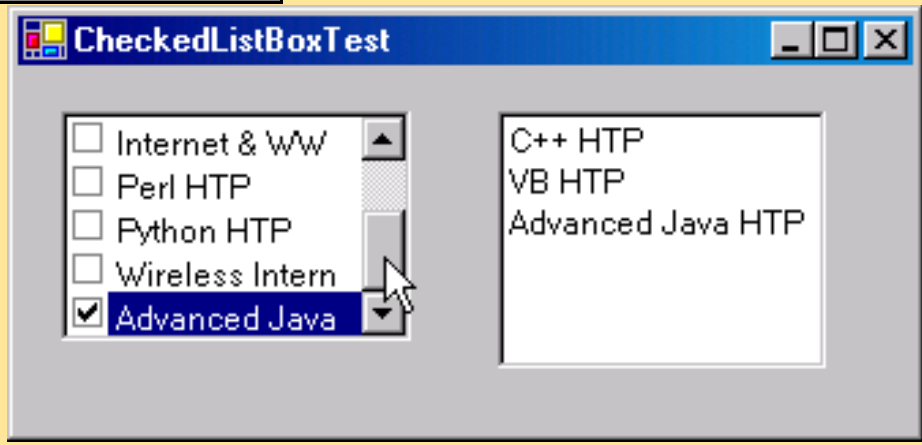
```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     public class CheckedListBoxTest: System.Windows.Forms.Form {
12       private System.Windows.Forms.CheckedListBox
                    inputCheckedListBox;
15       private System.Windows.Forms.ListBox displayListBox;
16       [STAThread]
17       static void Main() {
19          Application.Run( new CheckedListBoxTest() );
20       }
```



```
23      private void inputCheckedListBox_ItemCheck(object sender,
                System.Windows.Forms.ItemCheckEventArgs e ){
28       string item=inputCheckedListBox.SelectedItem.ToString();
32      if ( e.NewValue == CheckState.Checked )
33           displayListBox.Items.Add( item );
34        else
35           displayListBox.Items.Remove( item );
36      }
37  }
```

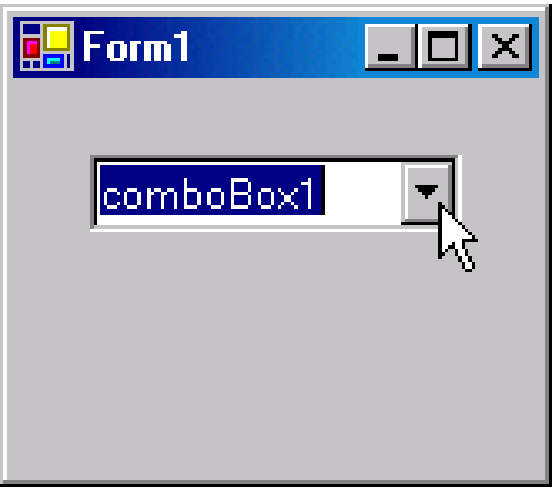


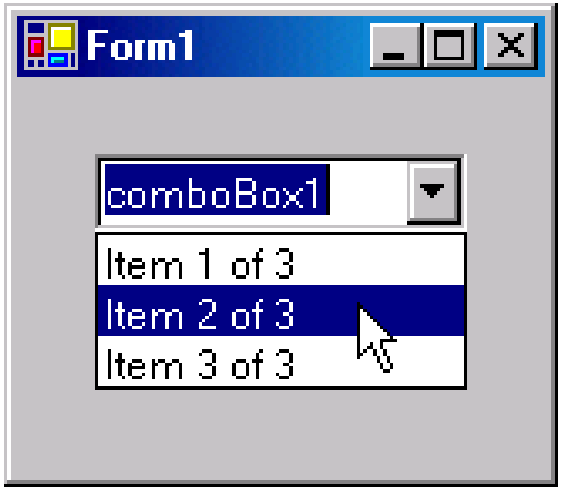


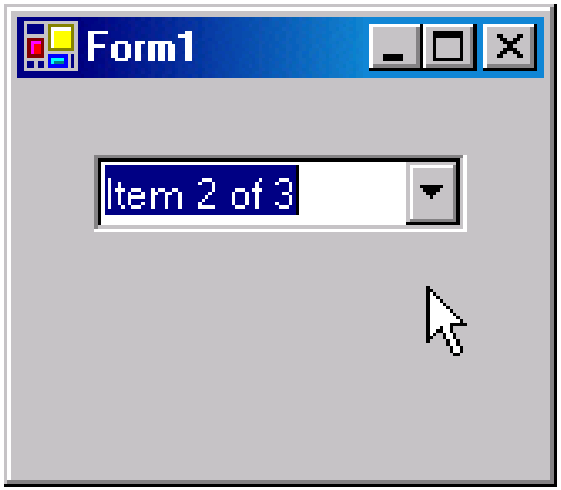


13.5 ComboBoxes

- Combine **TextBox** and drop-down list
- **Add** method adds object to collection
- Properties:
  - **DropDownStyle:** determines type of **ComboBox**
  - **Items:** returns objects in the list
  - **SelectedItem**: returns object selected
  - **SelectedIndex**: returns index of selected item





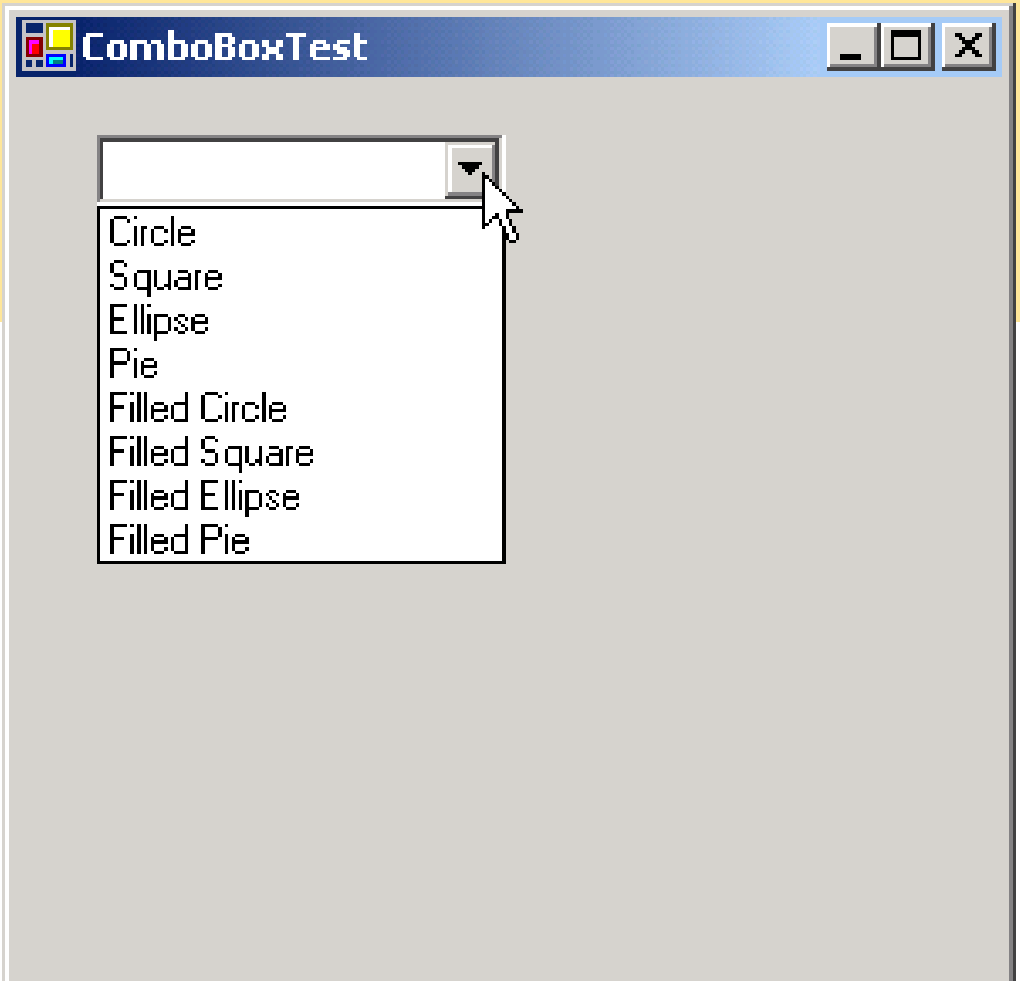


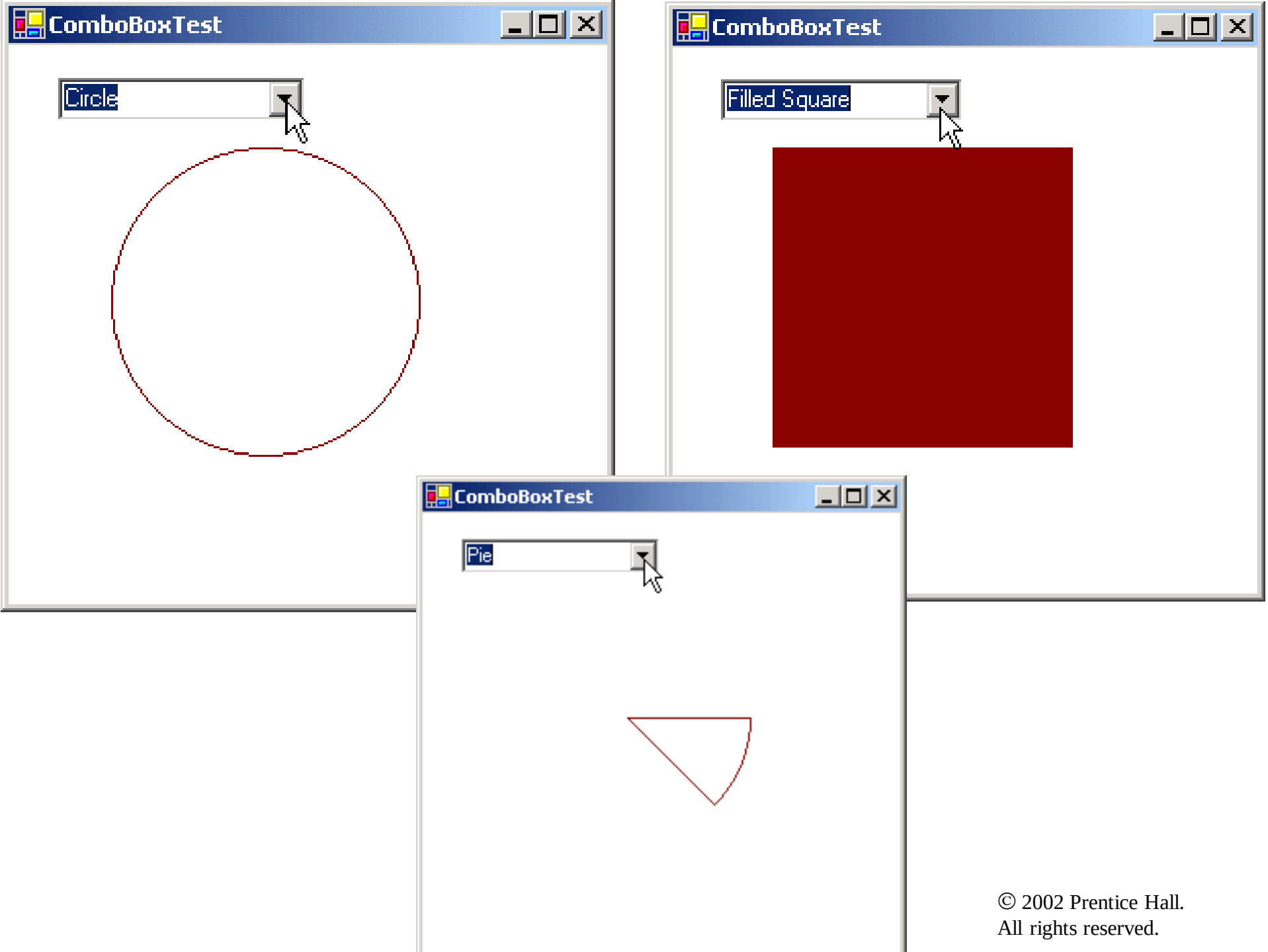
ComboBox events and properties	Description / Delegate and Event Arguments
Common Properties	
DropDownStyle	Determines the type of combo box. Value **Simple** means that the text portion is editable and the list portion is always visible. Value **DropDown** (the default) means that the text portion is editable, but the user must click an arrow button to see the list portion. Value **DropDownList** means that the text portion is not editable and the user must click the arrow button to see the list portion.
Items	The collection of items in the **ComboBox** control.
MaxDropDownItems	Specifies the maximum number of items (between **1** and **100**) that can display in the drop-down list. If the number of items exceeds the maximum number of items to display, a scroll bar appears.
SelectedIndex	Returns index of currently selected item. If there is no currently selected item, **-1** is returned.
SelectedItem	Returns a reference to the currently selected item.
Sorted	Specifies whether items in a list are alphabetized. If **True**, items appear in alphabetical order. Default is **False**.
Common Event	*(Delegate **EventHandler**; event arguments **EventArgs**)*
SelectedIndex-Changed	Generated when the selected index changes (such as when a check box has been checked or unchecked). Default when control is double-clicked in designer.

```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     public class ComboBoxTest : System.Windows.Forms.Form {
12       private System.Windows.Forms.ComboBox imageComboBox;
13       [STAThread]
14       static void Main() {
16          Application.Run( new ComboBoxTest() );
17       }
19       private void imageComboBox_SelectedIndexChanged(
20          object sender, System.EventArgs e ) {
23          Graphics myGraphics = base.CreateGraphics();
25          Pen myPen = new Pen( Color.DarkRed );
27          SolidBrush mySolidBrush =
               new SolidBrush( Color.DarkRed );
```

```
30          myGraphics.Clear( Color.White );
32          switch ( imageComboBox.SelectedIndex ) {
34             case 0:
35                myGraphics.DrawEllipse(myPen, 50, 50, 150, 150 );
37                break;
38             case 1:
38                myGraphics.DrawRectangle(myPen,50,50, 150, 150 );
41                break;
42             case 2:
43                myGraphics.DrawEllipse(myPen, 50, 85, 150, 115 );
45                break;
46             case 3:
47                myGraphics.DrawPie(myPen,50,50,150, 150, 0, 45 );
48                break;
50             case 4:
51                myGraphics.FillEllipse(mySolidBrush,50,50,150,150);
53                break;
54             case 5:
55                myGraphics.FillRectangle(mySolidBrush,50,50,150,
                                  150 );
57                break;
```

```
58      case 6:
59         myGraphics.FillEllipse(mySolidBrush, 50, 85, 150, 115 );
61         break;
62      case 7:
63         myGraphics.FillPie(mySolidBrush,50,50, 150, 150,0, 45 );
65         break;
66          }
67       }
68    }
```





13.6 TreeView

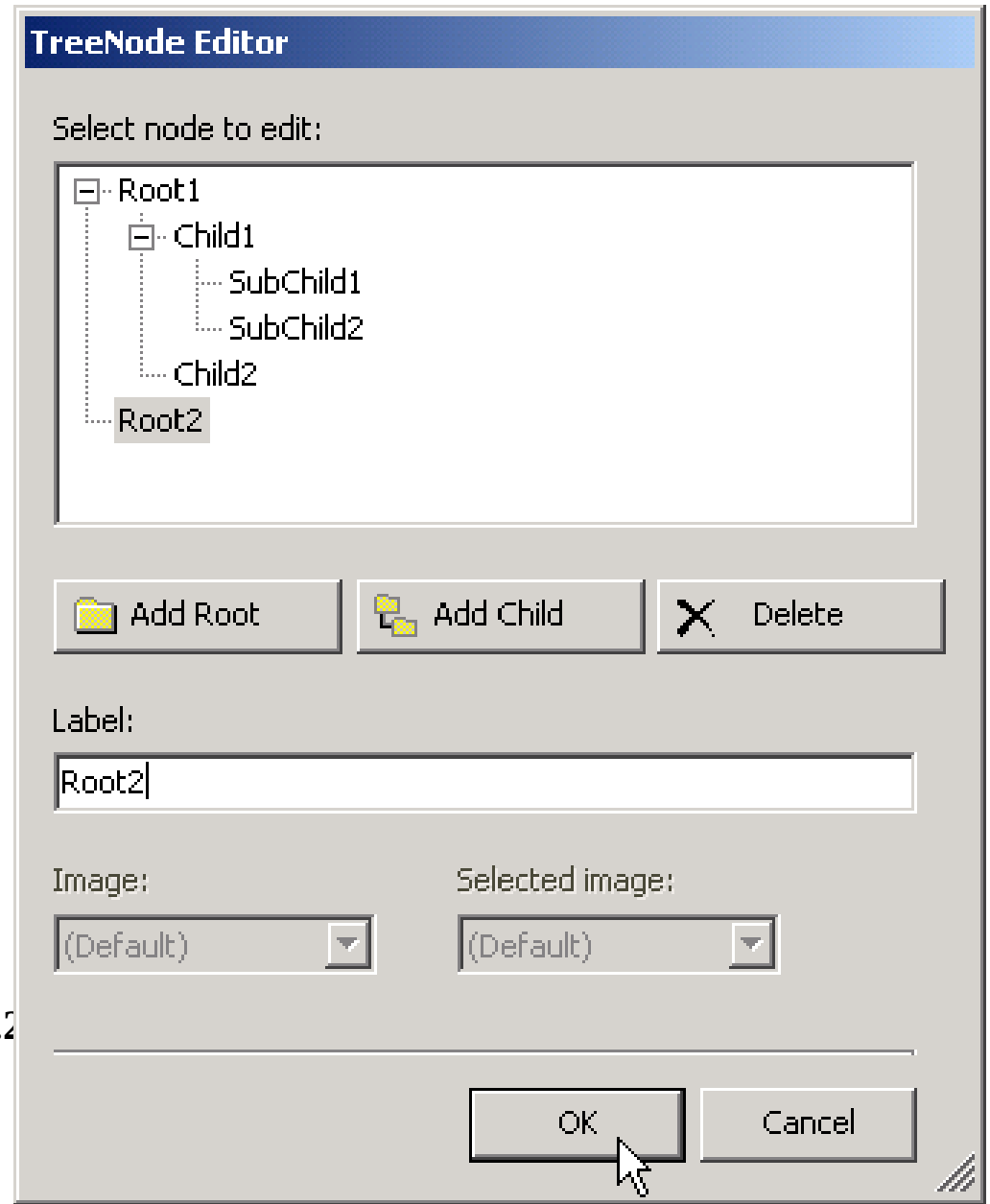
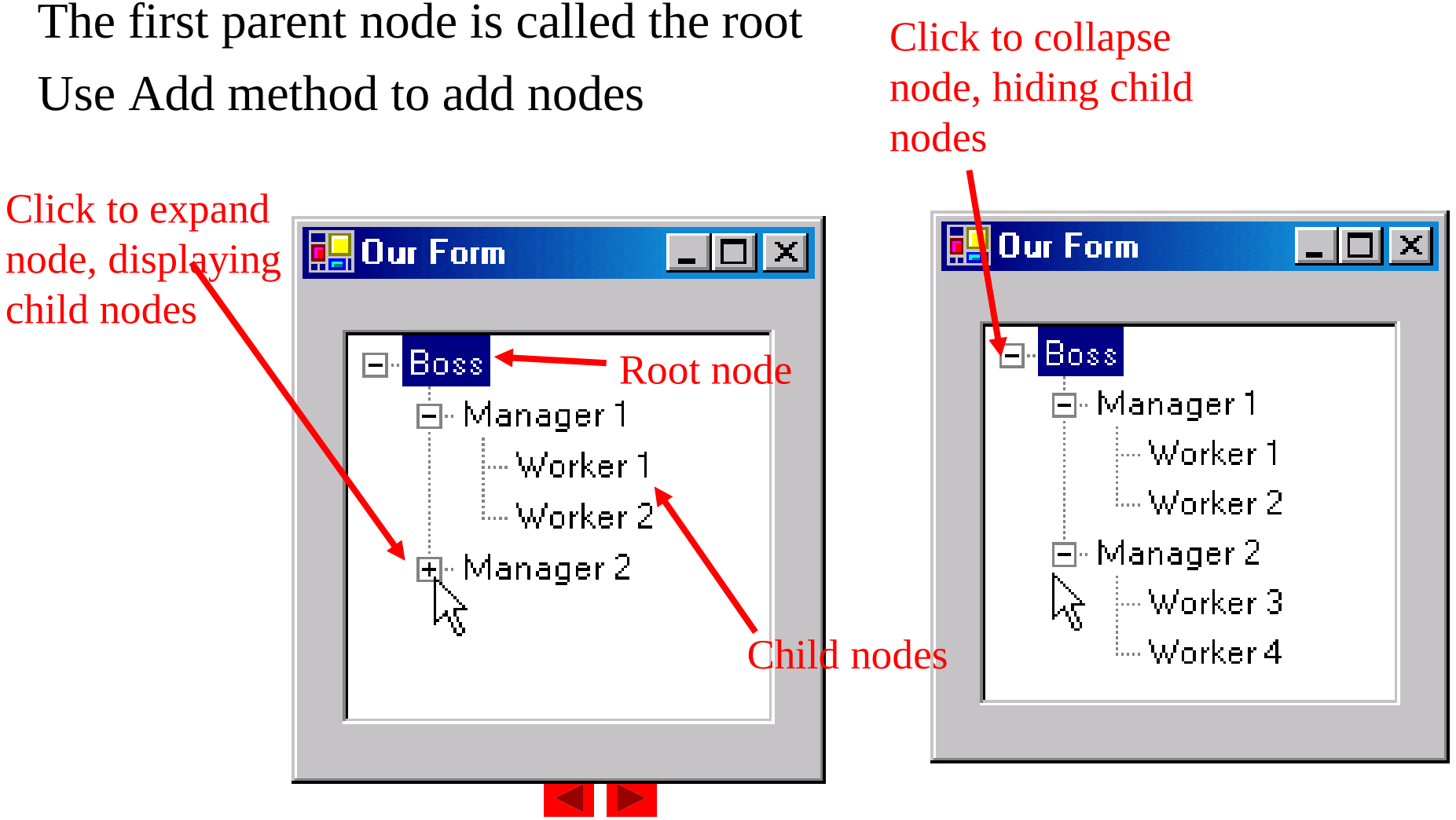


Fig. 13.2

13.6 TreeViews

- Displays nodes hierarchically
- Parent nodes have children
- The first parent node is called the root
- Use Add method to add nodes



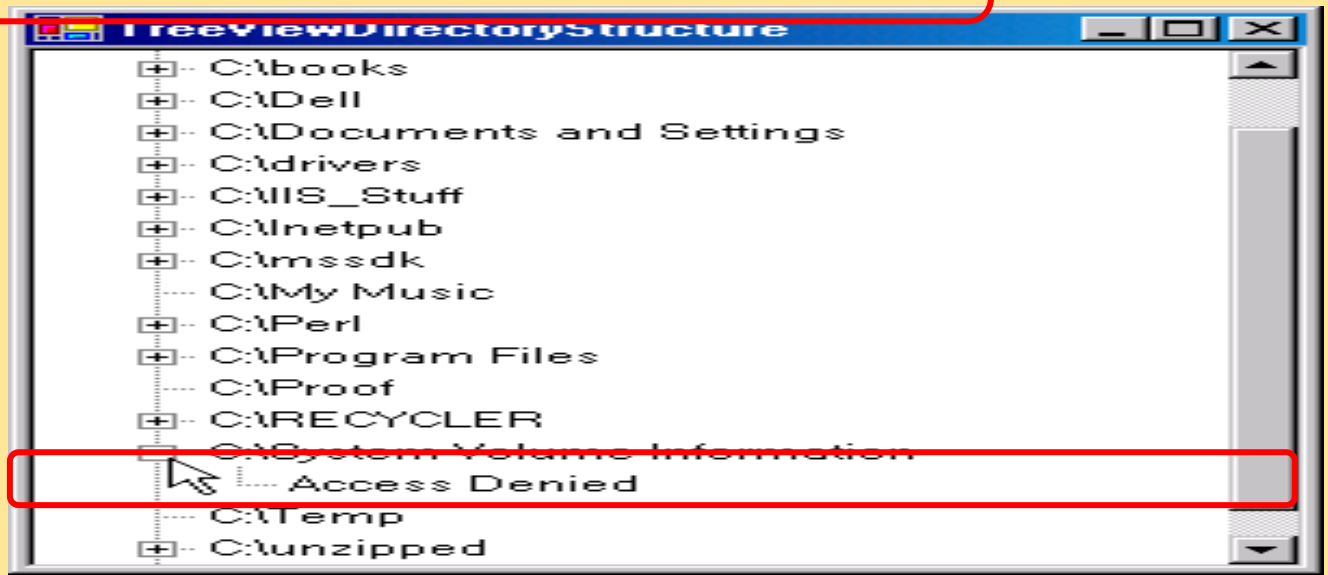
13.6 TreeView

TreeView properties and events	Description / Delegate and Event Arguments
Common Properties	
CheckBoxes	Indicates whether checkboxes appear next to nodes. **True** displays checkboxes. Default is **False**.
ImageList	Indicates the **ImageList** used to display icons by the nodes. An ***ImageList*** is a collection that contains a number of **Image** objects.
Nodes	Lists the collection of **TreeNodes** in the control. Contains methods **Add** (adds a **TreeNode** object), **Clear** (deletes the entire collection) and **Remove** (deletes a specific node). Removing a parent node deletes all its children.

TreeNode properties and methods	Description / Delegate and Event Arguments
Common Properties	
Checked	Indicates whether the **TreeNode** is checked. (**CheckBoxes** property must be set to **True** in parent **TreeView**.)
FirstNode	Specifies the first node in the **Nodes** collection (i.e., first child in tree).
FullPath	Indicates the path of the node, starting at the root of the tree.
ImageIndex	Specifies the index of the image to be shown when the node is deselected.
LastNode	Specifies the last node in the **Nodes** collection (i.e., last child in tree).
NextNode	Next sibling node.
Nodes	The collection of **TreeNode**s contained in the current node (i.e., all the children of the current node). Contains methods **Add** (adds a **TreeNode** object), **Clear** (deletes the entire collection) and **Remove** (deletes a specific node). Removing a parent node deletes all its children.
PrevNode	Indicates the previous sibling node.
SelectedImageIndex	Specifies the index of the image to use when the node is selected.
Text	Specifies the text to display in the **TreeView**.
Common Methods	
Collapse	Collapses a node.
Expand	Expands a node.
ExpandAll	Expands all the children of a node.
GetNodeCount	Returns the number of child nodes.

```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
9     using System.IO;
10    public class TreeViewDirectoryStructureTest
11       : System.Windows.Forms.Form {
14       private System.Windows.Forms.TreeView directoryTreeView;
15       [STAThread]
16       static void Main() {
18          Application.Run(
19             new TreeViewDirectoryStructureTest() );
20       }
```

```
21      public void PopulateTreeView(
22          string directoryValue, TreeNode parentNode ) {
25          string[] directoryArray =
26              Directory.GetDirectories( directoryValue );
28          try {
30              if ( directoryArray.Length != 0 ) {
35                  foreach ( string directory in directoryArray ) {
37                      TreeNode myNode = new TreeNode( directory );
39                      parentNode.Nodes.Add( myNode );
41                      PopulateTreeView( directory, myNode );
42                  }
43              }
44          }
46          catch ( UnauthorizedAccessException ) {
48              parentNode.Nodes.Add( "Access denied" );
49          }
50      }
```



```
52      private void TreeViewDirectoryStructureTest_Load(
53          object sender, System.EventArgs e) {
57          directoryTreeView.Nodes.Add( "C:\\" );
58          PopulateTreeView("C:\\", directoryTreeView.Nodes[0] );
60      }
61  }
```