Chapter 7
Lists and Printing

Chapter Objectives

- Create and use list boxes and combo boxes
- Differentiate among the available types of combo boxes
- Add and remove items in a list at run time
- Determine which item in a list is selected
- Use the Items.Count property to determine the number of items in a list
- Display a selected item from a list
- Send information to the printer or the Print Preview window using the PrintDocument class

Chapter Objectives

- Use do, while, loops to execute a series of statements
- Use for loops to execute a series of statements
- Skip to the next iteration of a loop by using the continue statement
List Boxes and Combo Boxes

- List boxes and combo boxes
  - Have most of the same properties and operate in a similar fashion
  - One exception is that a combo box control has a DropDownStyle property
    - Determines whether list box has a text box for user entry and if the list will drop down
  - Provide the user with a list of items from which to select

List Boxes and Combo Boxes – cont’d

- Combo boxes have a Text property
  - Set or remove at design time
- List boxes have a Text property
  - Access it only at run time

The Items Collection

- The list of items in a list box or combo box
  - Collections are (an array of) objects that have properties and methods that allow:
    - Adding, removing, counting, and clearing (all) items
    - Referring to individual elements
  - Can refer to the individual items in a collection by an index
    - Zero based
    - For a collection of ten items, the indexes range from 0 to 9
    - To refer to the first item in the Items collection use Items[0]
    - Notice the square brackets
Filling a List

- Several methods to fill the Items collection of a list box or combo box
  - Fill the list at design time if the list is known and the list never changes
  - Define the Items collection in the Properties window
  - Fill the list during program execution
    - Items.Add or Items.Insert methods

Using the Items.Add Method

- Items can be added to a list at run time
- Alter placement of a new item by setting the control's Sorted property to true
  - General Form
    - `ObjectName.Items.Add(ItemValue);`
  - Examples
    ```csharp
    schoolsListBox.Items.Add("Harvard");
    schoolsListBox.Items.Add("Stanford");
    schoolsListBox.Items.Add(schoolsTextBox.Text);
    majorsComboBox.Items.Add(majorsComboBox.Text);
    majorsComboBox.Items.Add(majorString);
    coffeeComboBox.Items.Add(coffeeComboBox.Text);
    schoolsListBox.Items.Add(schoolsTextBox.Text);
    ```

Using the Items.Insert Method

- Choose location for new item added to list
  - `Items.Insert method`
    - Specify index position for new item
    - An existing position or end of list
    - Index position is zero based
      - First position, use index position = 0
      - Inserting an item beyond end of list throws an exception
  - Do not set list control's Sorted property to true
  - General form:
    ```csharp
    ObjectName.Items.Insert(IndexPosition, ItemValue);
    ```
The SelectedIndex Property

- Index number of the currently selected item is stored in the SelectedIndex property.
- If no list item is selected, the SelectedIndex property is negative 1 (-1).
- Set the SelectedIndex property in code to select an item in the list or deselect all items.

```csharp
// Select the fourth item in list.
coffeeTypesListBox.SelectedIndex = 3;

// Deselect all items in list.
coffeeTypesListBox.SelectedIndex = -1;
```

The Items.Count Property

- Use to determine the number of items in the list.
  ```csharp
  totalItemsInteger = itemsListBox.Items.Count;
  MessageBox.Show("The number of items in the list is " +
  ItemsListBox.Items.Count.ToString());
  ```
- Items.Count is always one more than the highest possible SelectedIndex because indexes begin with zero.
- Example: For a list of five items:
  - Items.Count = 5
  - Highest Index = 4
  - Possible values for SelectedIndex = 0, 1, 2, 3, 4

Referencing the Items Collection

- Combine the Items property and the SelectedIndex property to refer to a selected element of a list.
  ```csharp
  selectedFlavorString = flavorListbox.Items[flavorListbox.SelectedIndex].ToString();
  ```
- Retrieve the selected list item by referring to the Text property of the control.
  ```csharp
  selectedMajorLabel.Text = majorsComboBox.Text;
  ```
- Assigning a value to an item replaces previous contents of that position.
  ```csharp
  schoolsListBox.Items[0] = "My School";
  ```
Removing an Item from a List

- **Items.RemoveAt method**
  - Removes an item by index
  - First list element is 0, last element is Items.Count – 1
  - An invalid index will cause an IndexOutOfRangeException exception
  ```csharp
  namesListBox.Items.RemoveAt(0);
  schoolsComboBox.Items.RemoveAt(indexInteger);
  coffeeComboBox.Items.RemoveAt(coffeeComboBox.SelectedIndex);
  ```

- **Items.Remove method**
  - Looks for the specified string, removes item if string found
  - No exception is generated if the item is not found
  ```csharp
  namesListBox.Items.Remove("My School");
  schoolsComboBox.Items.Remove(schoolTextBox.Text);
  coffeeComboBox.Items.Remove(coffeeComboBox.Text);
  ```

Clearing a List

- **Items.Clear method**
  - Empty a combo box or list box
  - Clear all items from a list
  ```csharp
  // Confirm clearing the majors list.
  DialogResult responseDialogResult;
  responseDialogResult = MessageBox.Show("Clear the majors list?", "Clear Majors List", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
  if (responseDialogResult == DialogResult.Yes)
  {
    majorsComboBox.Items.Clear();
  }
  ```

Sending Information to the Printer

- Use the .NET PrintDocument and PrintPreviewDialog components from the Printing tab of the toolbox
- Printing is done through the Windows environment
  - Allows use of fonts installed on any given system
  - Printing dialogs and page setup are the same as any other Microsoft product
- Several companies sell utilities that do a nice job of designing and printing reports
- C# Professional Edition and Team System Edition include Crystal Reports for creating reports from database files
The PrintDocument Component

- Set up output for the printer using methods and events of the PrintDocument component
  - Appears in component tray

The PrintDocument Print Method

- Print method sends command to print information according to the layout specified in the PrintPage event handler
- Place Print method in the Click event handler for a button or menu item

Setting Up the Print Output

- The code to set up the printed page belongs in the PrintDocument's PrintPage event handler
- The PrintPage event fires once for each page to be printed
  - This technique is referred to as a callback
  - In a callback the object notifies the program it needs to do something by firing an event
- The PrintDocument Print method executes the PrintPage event handler
  - Holds code that describes exactly how pages should print
- The PrintDocument object also fires events for BeginPrint and EndPrint
  - Code can be written for these events
- System.Drawing.Printing.PrintPageEventArgs e argument
  - Properties and methods of PrintPageEventArgs
  - Determines page margins and sends a string of text to the page
The Graphics Page

- Set up a graphics page in memory
  - Page is sent to the printer
- Page can contain strings of text and graphic elements
- Specify the exact X and Y coordinates of each element to be printed on the page
  - The X coordinate is the horizontal distance across the page
  - The Y coordinate is the vertical distance from the top of the page
  - Measurements are in pixels

Setting the X and Y Coordinates – Step 1

- For each print line, specify X and Y coordinates
- Create variables declared as float data type to hold the X and Y values
- The PrintPageEventArgs argument has several useful properties to determine the present settings
  - MarginBounds
  - PageBounds
  - PageSettings
- Example: Set the X coordinate to the current left margin and the Y coordinate to the top margin

```csharp
horizontalPrintLocationFloat = e.MarginBounds.Left;
verticalPrintLocationFloat = e.MarginBounds.Top;
```

Setting the X and Y Coordinates – Step 2

- To print multiple lines, increment the Y coordinate (forces next line down the page)
  - Add the height of a line to the previous Y coordinate to calculate the next line’s Y coordinate
  - Find height of a line in the current font using the font’s GetHeight method

```csharp
// Declarations at the top of the method.
Font printFont = new Font("Arial", 12);
float lineHeightFloat = printFont.GetHeight;
float horizontalPrintLocationFloat = e.MarginBounds.Left;
float verticalPrintLocationFloat = e.MarginBounds.Top;

// Print a line.
Graphics.DrawString(printLineString, printFont, Brushes.Black, horizontalPrintLocationFloat, verticalPrintLocationFloat);
verticalPrintLocationFloat += lineHeightFloat;
```
Printing Summary

- Easy steps for nearly all printing tasks
  - Place code in PrintPage event handler of the PrintDocument component
  1. At the top of the method, define the font(s), line height, and X and Y coordinates
  2. Set up the line to print
  3. Print the line
  4. Increment the Y coordinate if another line will be printed
  5. Place steps 2, 3, and 4 inside a loop if there are multiple lines to print

Printing the Contents of a List Box

- Combine the techniques for printing, looping, and the list box properties
  - The Items.Count property provides the number of iterations for the loop
  - The Items collection allows the actual values from the list to print

```csharp
// Print out all items in the coffeeComboBox list.
for (int indexInteger = 0; indexInteger < coffeeComboBox.Items.Count; indexInteger++)
{
    // Set up a line.
    printLineString = coffeeComboBox.Items[indexInteger];
    // Send the line to the graphics page object.
    e.Graphics.DrawString(printLineString, printFont, Brushes.Black,
                           horizontalPrintLocationFloat, verticalPrintLocationFloat);
    // Increment the Y position for the next line.
    verticalPrintLocationFloat += lineHeightFloat;
}
```

Printing the Selected Item from a List

- Use the Text property to print the selected item from a list box or combo box

```csharp
// Set up the line.
printLineString = "Coffee: " + coffeeComboBox.Text + " Syrup: " + syrupListBox.Text;
// Send the line to the graphics page object.
e.Graphics.DrawString(printLineString, printFont, Brushes.Black,
                       horizontalPrintLocationFloat, verticalPrintLocationFloat);
```
Displaying a Print Preview

- View printer’s output and choose to print or cancel
  - Helpful for testing and debugging
- `PrintPreviewDialog` component is key
- Add control to component tray

Displaying a Print Preview – cont’d

- The `PrintPreviewDialog` component uses a `PrintDocument` component declared for printer output
  - Assign the `PrintDocument` to the `Document` property of the `PrintPreviewDialog`
  - Execute the `ShowDialog` method
    - The same `PrintPage` event handler executes as for the `PrintDocument`

```csharp
private void filePrintPreviewMenu_Click(object sender, System.EventArgs e)
{
    // Begin the process for print preview.
    printPreviewDialog1.Document = printAllPrintDocument;
    printPreviewDialog1.ShowDialog();
}
```

Printing Multiple Pages

- The `PrintDocument`’s `PrintPage` event fires once for each page
- Set the `HasMorePages` property of the `PrintPageEventArgs` argument to `true` to print more than one page