6. The *current dividend yield* for common stock is calculated with the formula

$$\text{current dividend yield} = \frac{\text{most recent full-year dividend}}{\text{current share price}}.$$

Write a Web program to calculate the current dividend yield for a stock. See Fig. 12.9.

In Exercises 7 through 16, rework the example or exercise as a Web program.

7. Example 9 of Section 4.2
8. Example 3 of Section 5.1
9. Exercise 35 of Section 5.2
10. Exercise 36 of Section 5.2
11. Example 3 of Section 5.3
12. Exercise 11 of Section 5.3
13. Example 3 of Section 6.2
14. Exercise 30 of Section 6.1
15. Exercise 48 of Section 7.1
16. Example 9 of Section 7.1

The file States.txt contains the 50 U.S. states in the order in which they joined the union. Use this text file in Exercises 17 through 20.

17. Write a program whose page contains a button and a list box. When the user clicks on the button, the names of the states that end with "ia" should be displayed in the list box.
18. Write a program whose page contains a button and a list box. When the user clicks on the button, the program should determine the greatest length of the names of the states and then display (in the list box) all states having names of that length.
19. Write a program whose page contains a text box, a button, and a list box. When the user enters a letter into the text box and clicks on the button, the states beginning with that letter should be displayed in the list box.
20. Write a program whose page contains two buttons, a text box, and a list box. See Fig. 12.10. The user should be able to display a list of all states beginning with a vowel or all states beginning with a consonant (in alphabetical order) and determine the number of such states. ***Note:*** List boxes in VWD do not have a Sorted property.

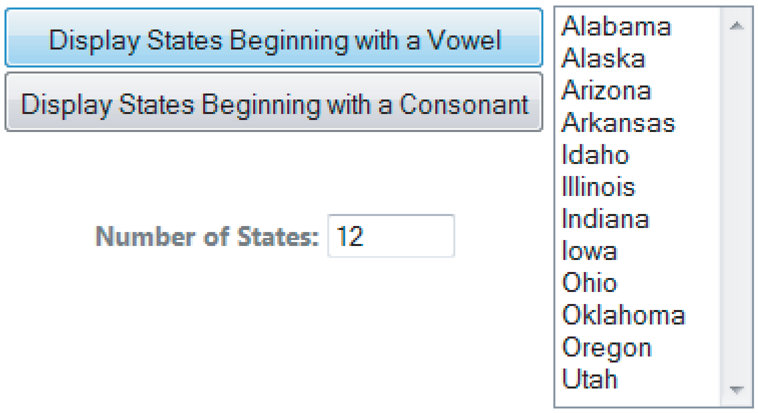


FIGURE 12.10 An outcome of Exercise 20.

12.2 Programming for the Web, Part II

In this section we explore some controls that are unique to VWD and some special features of Visual Web Developer.

Multiple Web Pages

VideoNote
Multiple Web pages

Web applications commonly use several Web pages. The pages contain links (called **hyperlinks**) that allow the user to navigate between pages. A hyperlink usually appears on a page as underlined text. Additional Web pages and hyperlinks can be easily added to Web programs.

The following steps add a new Web page to a program.

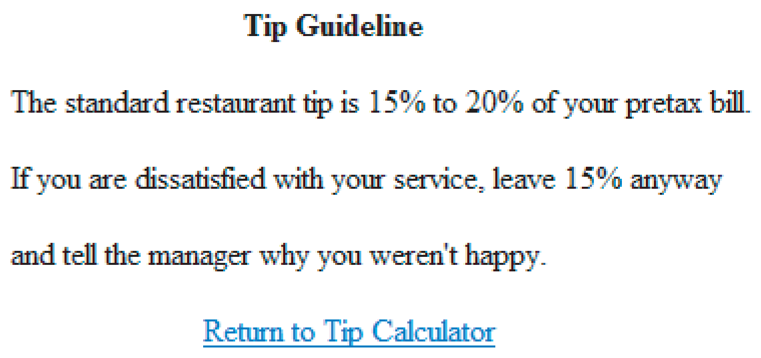
1. Click on the current Web page to ensure that it has the focus.
2. Click on *Add New Item* in the *Website* menu. (An Add New Item dialog box will appear.)
3. Click on *Web Form* in the center pane, type a name into the *Name* box, and click on the *Add* button. (The Designer for the new Web page will appear and the name of the page in the Solution Explorer will have the extension *aspx*.)

To switch from one Web page to another in Design mode, right-click on the name of the Web page in the Solution Explorer window, and click on either *View Designer* or *View Code* in the context menu that appears.

There are several ways to navigate from one page to another at run time. The most common way is to use a hyperlink control (found in the *General* group of the Toolbox). The following walkthrough adds an additional Web page to the tip-calculator program from Section 12.1.

Hyperlink walkthrough

1. Follow the steps in the "Building on an Existing Web Program" discussion at the end of Section 12.1 to create a program named TipWithHelp that builds on the program TipCalculator.
2. Add a new page (that is, Web Form) named Help.aspx to the program via the *Add New Item* entry in the *Website* menu. (The Designer for the new page will appear.)
3. Add text to the new Web page as shown in the first four lines of Fig. 12.11.


Tip Guideline

The standard restaurant tip is 15% to 20% of your pretax bill.

If you are dissatisfied with your service, leave 15% anyway

and tell the manager why you weren't happy.

Return to Tip Calculator

FIGURE 12.11 **The Help.aspx page.**

4. Press the Enter key and then double-click on the HyperLink control in the *Standard* group of the Toolbox to place a hyperlink control below the text.
5. Set the ID property of the hyperlink control to *lnkReturn* and set the Text property to *Return to Tip Calculator*. See Fig. 12.11.
6. Click on the ellipsis button in the hyperlink's NavigateUrl property Settings box. (A Select URL dialog box will appear.)
7. Double-click on *Default.aspx* in the right pane of the Select URL dialog box to set the hyperlink's NavigateUrl property to ~/Default.aspx.

8. Right-click on *Default.aspx* in the Solution Explorer window and then click on *View Designer* in the context menu to display the Designer for the original Web page.
9. Place the cursor on the line of the table containing the *Calculate Tip* button, hover the mouse over *Insert* in the *Table* menu, and click on *Row Below*. (A blank row will be created in the table with the insertion point in that row.)
10. Double-click on the HyperLink control in the Toolbox to place a hyperlink control in the new row.
11. Set the ID property to *lnkHelp*, set the Text property to *Tipping Help*, and set the NavigateUrl property to *~/Help.aspx*. (To specify the setting for the NavigateUrl property, click on the ellipsis in the Settings box and then click on *Help.aspx* in the right pane of the Select URL dialog box that appears.)
12. Set the Align property of the row containing the hyperlink control to *Center*. Figure 12.12 shows the Designer.

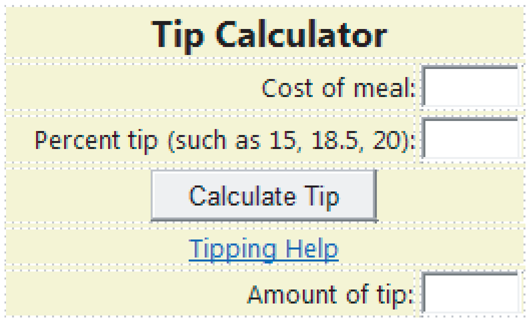


FIGURE 12.12 **The Default.aspx page.**

13. Run the program and click on the hyperlink. (The Help page appears.)
14. After you read the advice, click on the *Return to Tip Calculator* hyperlink to go back to the Tip Calculator Web page. (You can now calculate the amount of a tip.)

Validation Controls

The *Validation* group in the Toolbox contains five controls that can be used to check user input. In this section we use the RequiredFieldValidator control and the RangeValidator control to authenticate user input.

The RequiredFieldValidator control is used with text boxes to ensure that they contain data and is used with list boxes to check that an item has been selected. The two key properties of the RequiredFieldValidator control are ControlToVerify and ErrorMessage. Suppose you are adding validation for a page containing an input text box control named txtBox and a list box control named lstBox. Two RequiredFieldValidator controls will be needed; let's call the controls rfvText and rfvList. The setting for ControlToVerify in rfvText will be txtBox and in rfvList will be lstBox. If a button on the page is clicked on before data has been entered into the text box, or before an item in the list box has been selected, then the button's Click event will not be executed. In both of these situations, the setting for the ErrorMessage property will be displayed in the RequiredFieldValidator control.

The RangeValidator control checks that an entry in a text box falls within a set range of values. The five key properties of the RangeValidator control are ControlToVerify, ErrorMessage, Type, MinimumValue, and MaximumValue. The possible settings for the Type property are *String*, *Integer*, *Double*, *Date*, and *Currency*. The settings for the Minimum and Maximum properties should be suitable for the specified type, and the setting for the Minimum property should be less than or equal to the setting for the Maximum property. Suppose that the setting for the ControlToVerify property is a text box and that data outside of the specified range is typed into the text box. Then the error message will be displayed in the RangeValidator control as soon as the text box loses focus.

Example 1 The following program is an extension of the tip-calculator program from Section 12.1. The RequiredFieldValidator control is located to the right of the cost text box and the RangeValidator control is located to the right of the percent tip text box. When the user clicks on the button, the program checks to see that a cost has been entered and that the percentage for the tip is reasonable. (The assumption is made that a tip should never exceed 100%.) To be completely protected, an additional RequiredFieldValidator control for the *Percent* tip text box could have been added. Figure 12.13 shows the contents of the Designer and Fig. 12.14 shows the settings for the controls. The two red sentences will not appear initially in the Web browser when the program is run.

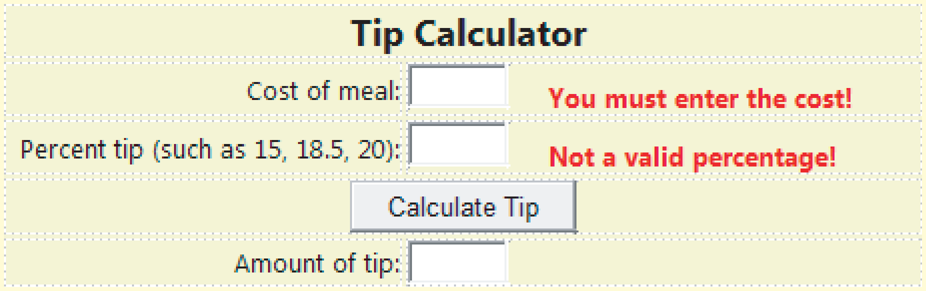


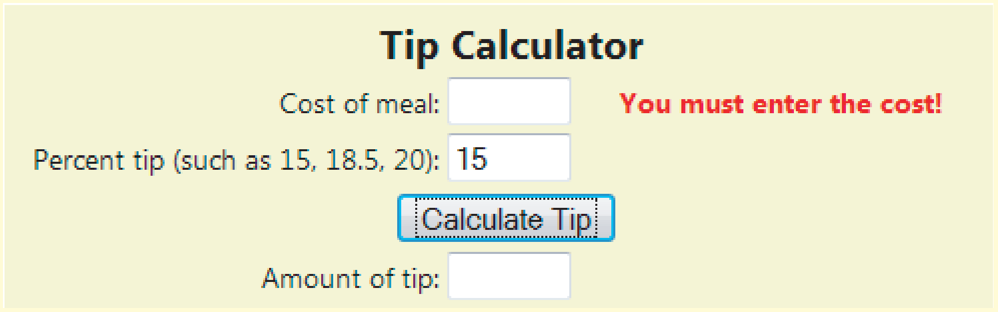
FIGURE 12.13 Design for Example 1.

OBJECT	PROPERTY	SETTING
txtCost		
CostRequiredFieldValidator	ControlToValidate	txtCost
	ErrorMessage	You must enter the cost!
	Font/Bold	True
	ForeColor	Red
txtPercent		
PercentRangeValidator	ControlToValidate	txtPercent
	ErrorMessage	Not a valid percentage!
	Type	Double
	MaximumValue	100
	MinimumValue	0
	Font/Bold	True
	ForeColor	Red
btnCalculate	Text	Calculate Tip
txtTip	ReadOnly	True

FIGURE 12.14 Property settings for Example 1.

```
Protected Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim cost As Double = CDbl(txtCost.Text)
  Dim percent As Double = CDbl(txtPercent.Text) / 100
  txtTip.Text = FormatCurrency(percent * cost)
End Sub
```

[Run, enter 15 for the tip percent, neglect to enter a cost, and click on the *Calculate Tip* button.]



Postback

Consider the tip-calculator program. When the user enters two proper values into the text boxes and presses the *Calculate* button, the values in the text boxes are sent back to the server, the tip calculation is made, and then an updated page is generated and sent to the Web browser. The page is said to be *posted back* to the server for processing, and a **postback** is said to have occurred.

When a validation control is triggered, the matter is handled entirely by the browser—no postback occurs. Also, no postback occurs when the browser requests a new page from the server.

The Page Load Event

The **page load event** is the Web analog of the form load event in a Windows program. They differ chiefly in that the form load event occurs only once, whereas the page load event occurs when a Web program is first run and also every time there is a postback to a page. The reason is that after a Web page is sent to a browser, the program is terminated until another postback occurs. At that point, for all practical purposes the program is being run for the first time. Therefore, the page load event occurs again.

There is a way for you to guarantee that the code inside the page load event procedure is executed only once while working with a single page. Place the code inside an If . . . Then block whose condition uses the IsPostBack property. The general form of the event procedure is as follows:

```
Protected Sub Page_Load(...) Handles Me.Load
  If Not Page.IsPostBack Then
    (code to execute just once)
  End If
End Sub
```

Class-Level Variables

Local variables act the same in Web programs as they do in Windows programs. They do not exist until a procedure is called, and they cease to exist after the procedure terminates. Class-level variables in a Web program, on the other hand, cease to exist whenever a page is sent to the Web browser. Therefore, whatever values were assigned to them will be gone when the next postback occurs.

There are ways to have a value persist between postbacks. For instance, a value can be placed in a HiddenField control. Some more advanced methods for retaining values involve the use of session variables and cookies.

The RadioButtonList Control

Figure 12.15 shows the form used in Example 3 of Section 4.4. The four radio buttons are inside a group box control with the caption *Age*.

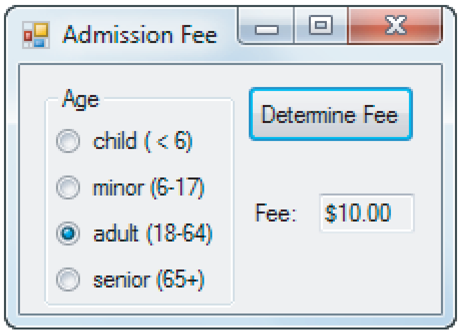


FIGURE 12.15 **Example 3 from Section 4.4.**

Visual Web Developer does not have a group box control capable of holding a list of radio buttons. The counterpart is the radio-button list control. The following VWD walkthrough creates an enhanced Web form of the program in Fig. 12.15. The Web program has the additional feature that a warning message is displayed if a radio button has not been selected when the *Determine Fee* button is clicked on.

A RadioButtonList Walkthrough

1. Start a VWD program with the name AdmissionFee and delete the text in the Main Content region.
2. Place a table having 3 rows, 2 columns, and width 275px onto the page.
3. Use the Table menu to merge the three cells in the left column of the table.
4. Type two spaces followed by the word *Age* into the left column.
5. Double-click on the word *Age* and make it boldface by clicking on the *Bold* button on the Toolbar.
6. Position the cursor to the right of the word *Age* and press the Enter key.
7. Double-click on *RadioButtonList* in the *Standard* Group of the Toolbox to place a radio-button list control on the page.
8. Set the control's ID property to *rblAges*.
9. Click on the control's *Tasks* button and then click on *Edit Items*. (The ListItem Collection Editor in Fig. 12.16 on the next page appears.)
10. Click on the *Add* button and then type "child (< 6)" into the setting for the Text property, as shown in Fig. 12.17.
11. Click on the *Add* button and type in "minor (6-17)" for the Text property.
12. Repeat Step 11 for "adult (18-64)" and for "senior (65+)".
13. Click on the *OK* button to indicate that all of the radio buttons have been added to the list.
14. Place a button with the name btnDetermine and Text setting "Determine Fee" into the first row of the right column.
15. Type the word "Fee:" into the second row of the right column, and then to the right of the word add a text box. Set its ID property to *txtFee* and its ReadOnly property to *True*.
16. Place a RequiredFieldValidator (with name rfvAge) in the third row of the right column.

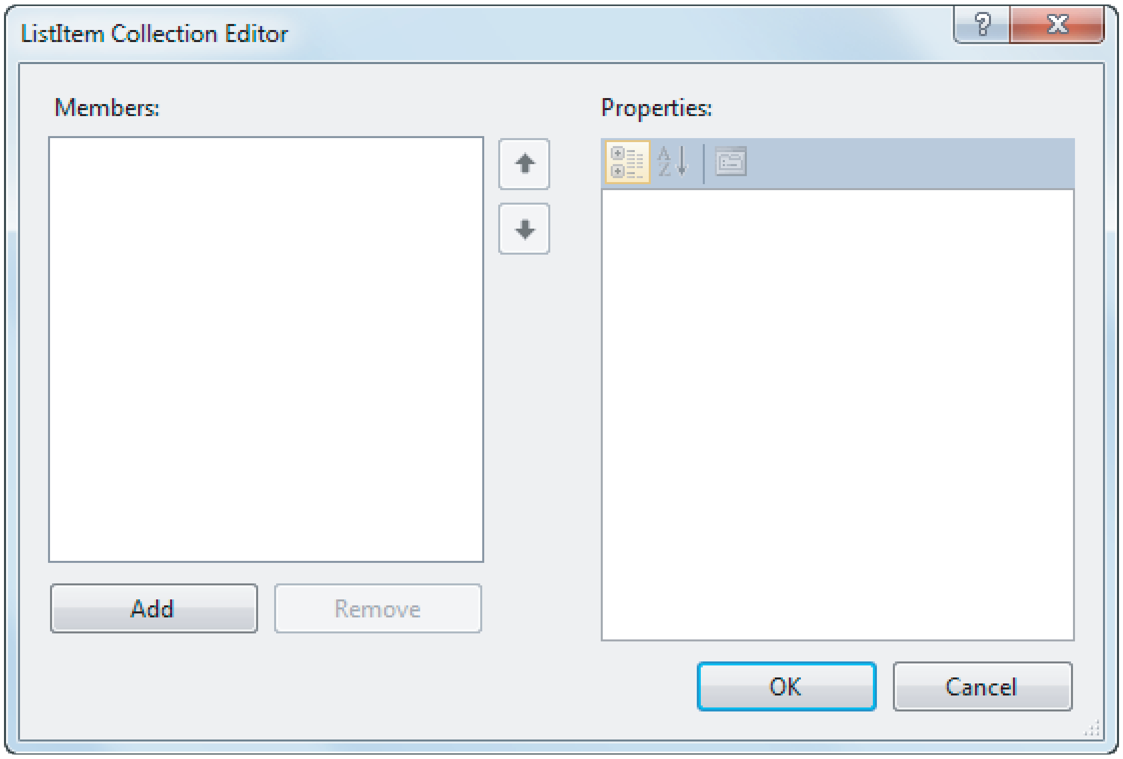
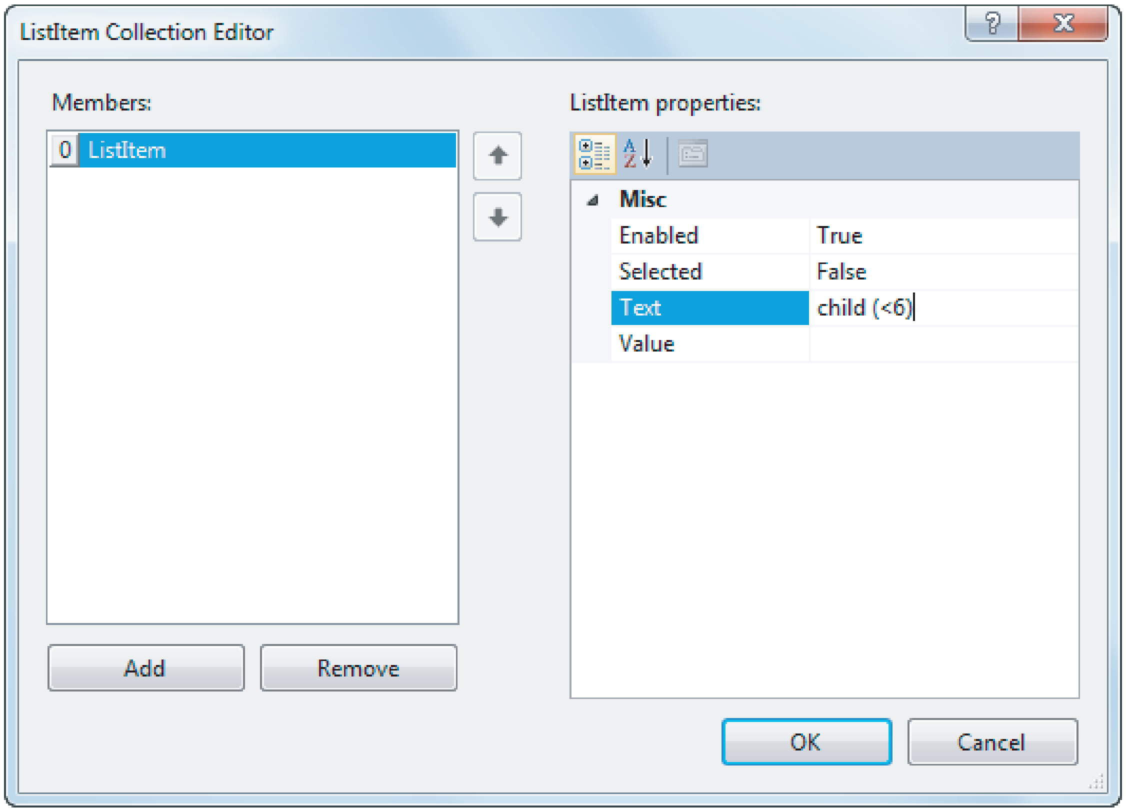


FIGURE 12.16 ListItem Collection Editor.



Misc

Enabled	True
Selected	False
Text	child (<6)
Value	

FIGURE 12.17 ListItem Collection Editor for a RadioButtonList control.

17. Set the ControlToValidate property of rfvAge to *rblAges*, set the ErrorMessage property to "You must select an age!", set the ForeColor to *Red*, and set the Font Bold property to *True*. (See Fig. 12.18.)

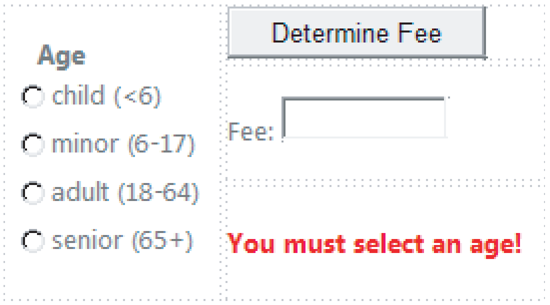


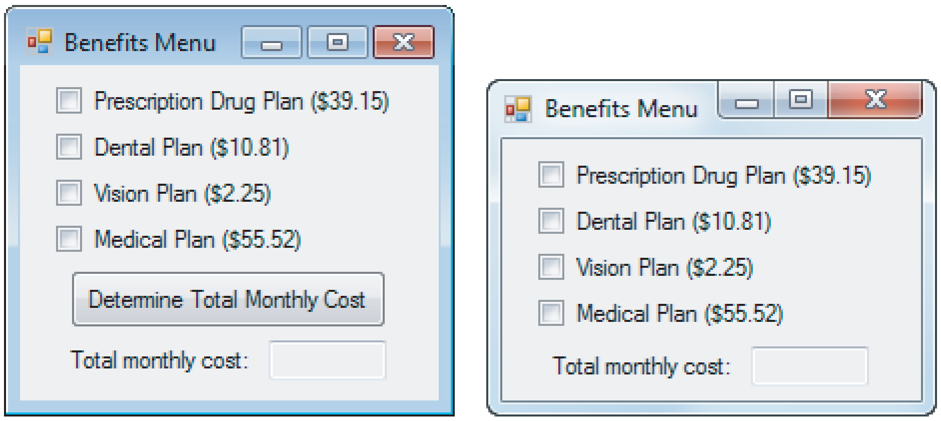
FIGURE 12.18 Designer from walkthrough.

18. The radio-button list control is similar to the Visual Basic list box control in that the value of its Text property is the selected item represented as a string. Double-click on the button and enter the following code:

```
Protected Sub btnDetermine_Click(...) Handles btnDetermine.Click
  Select Case rblAges.Text
    Case "child (<6)"
      txtFee.Text = FormatCurrency(0)
    Case "minor (6-17)"
      txtFee.Text = FormatCurrency(5)
    Case "adult (18-64)"
      txtFee.Text = FormatCurrency(10)
    Case "senior (65+)"
      txtFee.Text = FormatCurrency(7.5)
  End Select
End Sub
```

The Check Box Control

Figure 12.19 shows the forms from Examples 4 and 5 in Section 4.4. In Example 4 the user first selects check boxes and then clicks on the *Determine* button to obtain the total cost. Nothing happens until the button is clicked on. In Example 5, an event procedure is declared that is



(a) Example 4 (b) Example 5

FIGURE 12.19 Two programs from Section 4.4.

raised whenever one of the check boxes is clicked on. That event procedure updates the total cost. The difference between the two examples is that in Example 5 a computation is made every time the user clicks on a check box, whereas in Example 4 clicking on a check box does not cause a computation.

Both examples can be converted to Web programs with the same controls, settings, and code as in Section 4.4. The Web version of Example 4 will work just fine, but Example 5 will not work as expected. This happens because computations can be made only if there is a postback to the server. Clicking on a button triggers a postback, but clicking on a check box does not. The situation in the Web version of Example 5 is easily remedied. The check box control has an AutoPostBack property that is set to *False* by default. However, if it is changed to *True*, then clicking on the check box will trigger a postback and thus will allow a computation to be made. Therefore, all that is required to make the Web version of Example 5 work as intended is to change the AutoPostBack setting for each of the four check boxes.

By default, clicking on a button causes VWD to check for validation. Such is not the case with a check box. However, if you change a check box's CausesValidation property to True, clicking on it will invoke a validation check.

Comments

1. As an alternative to using a hyperlink control, a line of code of the form

```
Response.Redirect("WebPageName.aspx")
```

   can be used to navigate to another Web page.
2. In Windows programs, a list box's String Collection Editor can be used to fill the contents of the list box at design time. In Web programs this task is accomplished with a ListItem Collection Editor that is identical to the one for the radio-button list control.

Practice Problems 12.2

A statement such as

```
Dim states() As String = IO.File.ReadAllLines(MapPath("App_Data/States.txt"))
```

cannot be placed in the Declarations section of a Web program as is commonly done in Windows programs.

1. Why did we often avoid placing ReadAllLines statements inside event procedures in Windows programs?
2. Why is the answer to Practice Problem 1 not relevant to Web programs?

EXERCISES 12.2

1. Write a program that uses the page design in Fig. 12.20. The list box should be populated with the names of the 50 states in alphabetical order when the page is loaded. (The file States.txt contains the names of the U.S. states in the order in which they joined the union.) When the user clicks on the button, the selected state should be deleted from the list box. A validation control should be used to assure that a state has been selected before the button is clicked on. At all times the read-only text box should display the number of states remaining in the list.

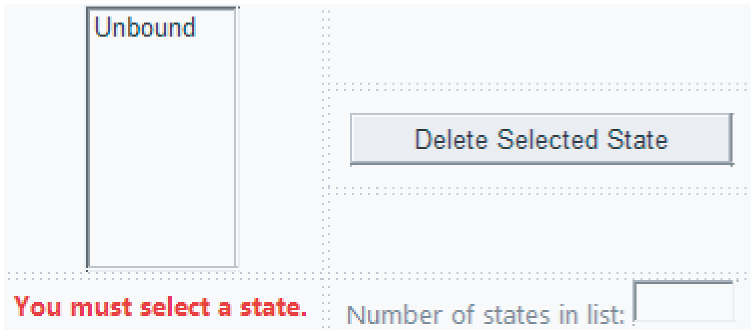


FIGURE 12.20 Web page design for Exercise 1.

2. Rework Exercise 20 from Section 4.4 (Presidential Eligibility) as a Web program. Add two validation controls for the "Date of birth" text box. One validation control should check that the date entered is 1/21/1978 or earlier (to guarantee that the person will be 35 by inauguration day in 2013) and the other should require that a date has been entered. The code for the button's Click event procedure can be shortened, since the date would have already been validated.
3. Rework Exercise 17 of Section 4.4 (Cost of a Computer) as a Web program. Use a RequiredFieldValidator control to assure that one of the radio buttons is selected. ***Note 1:*** If you set the radio-button list's RepeatDirection property to *Horizontal*, then the radio buttons will be displayed in a row. ***Note 2:*** You must set each check box's CausesValidation property to *True*.
4. Rework Exercise 25 of Section 4.4 (Health Club Fees) as a Web program. Use a RequiredFieldValidator control to assure that one of the radio buttons is selected before the fee is calculated.
5. Rework Example 1 of Section 5.1 (Convert Fahrenheit to Celsius) as a Web program. Require that the input text box contain a value when the button is clicked on and that the value is in the range from −459.67 °F (absolute zero) to 11,000 °F (approximate temperature of the surface of the sun).
6. Consider the program in the hyperlink walkthrough. Replace the hyperlink control in the Help page with a button having the caption "Return to Tip Calculator".
7. Rework Exercise 35 of Section 5.2 (Highest Two Grades) as a Web program. Require that data be entered into each input text box before the button's Click event is processed and validate that each grade is between 0 and 100. See Fig. 12.21.

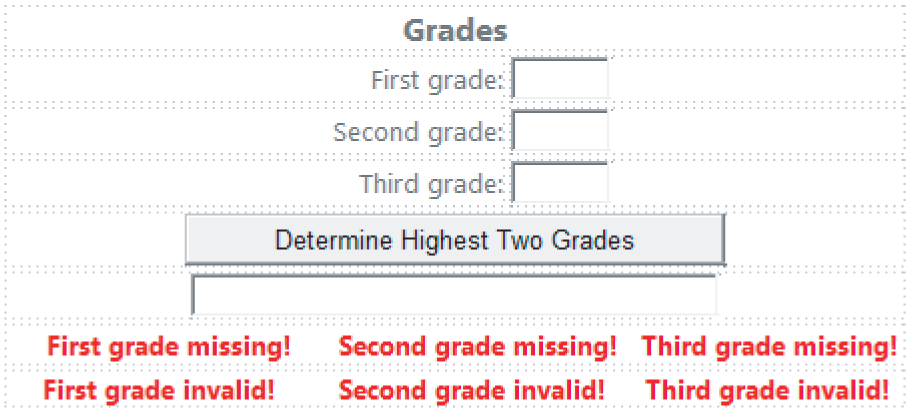


FIGURE 12.21 Page design for Exercise 7.

8. Rework Example 3 of Section 5.1 (Weekly Pay) as a Web program. Require that data be entered into each input text box before the button's Click event is processed and validate that the number of hours worked is at most 168 (the number of hours in a week).
9. Rework Exercise 37 of Section 5.2 (Alphabetize Two Words) as a Web program with the additional provision that both words must begin with lower-case letters. Use a RequiredFieldValidator and a RangeValidator control with each text box.
10. Rework Example 3 of Section 7.3 (Display Countries by Continent and Area) as a Web program without using a structure.
11. Rework Exercise 29 of Section 7.3 (Display Justices from a Specified State) as a Web program without using a structure.
12. Rework Example 1 of Section 7.4 (Intercity Distances) as a Web program. Use a RequiredFieldValidator and a RangeValidator control with each input text box.

Solutions to Practice Problems 12.2

1. This was done to avoid reading a text file from a disk into an array more than once.
2. With a Web program, any array loses its values between postbacks.

12.3 Using Databases in Web Programs

Databases play a prominent role in Web applications. In this section we show how to use LINQ to manipulate information retrieved from databases. The information will be displayed both in bar charts and in grids. Our bar charts will be generated by the Chart control, which is new to Visual Web Developer 2010.

This section uses the same types of databases as Chapter 10. However, we will use them in a different format called a Microsoft SQL Server format. (SQL is pronounced *sequel.*) The databases will have the extension *mdf*. In order to open these databases when using VWD Express, you must have Microsoft SQL Server installed on your computer. (Microsoft SQL Server Express is contained on the DVD accompanying this book and is usually installed when you install Visual Basic Express or Visual Web Developer Express.)

Creating a Bar Chart from a Database

The following walkthrough uses the Megacities database discussed in Section 10.2 and displays a bar chart showing the cities and their 2010 populations. The cities will be displayed in descending order of their 2010 populations. **Note:** VWD refers to a *bar chart* as a *column chart*.

The walkthrough proceeds in four stages.

Stage 1: Design the Web page.

Stage 2: Add a database connection.

Stage 3: Create an *object model* for the database. (The object model is used to enable LINQ queries to be performed on data retrieved from relational databases.)

Stage 4: Use a LinqDataSource control to display data in a Chart control.

Stage 1: Design the web page

1. Start a VWD program with the name PopBarChart and delete the text in the Main Content region.
2. Place a button control on the form, set its ID property to btnDisplay, set its Width property to 300px, and set its Text property to "Display City Populations in Descending Order".