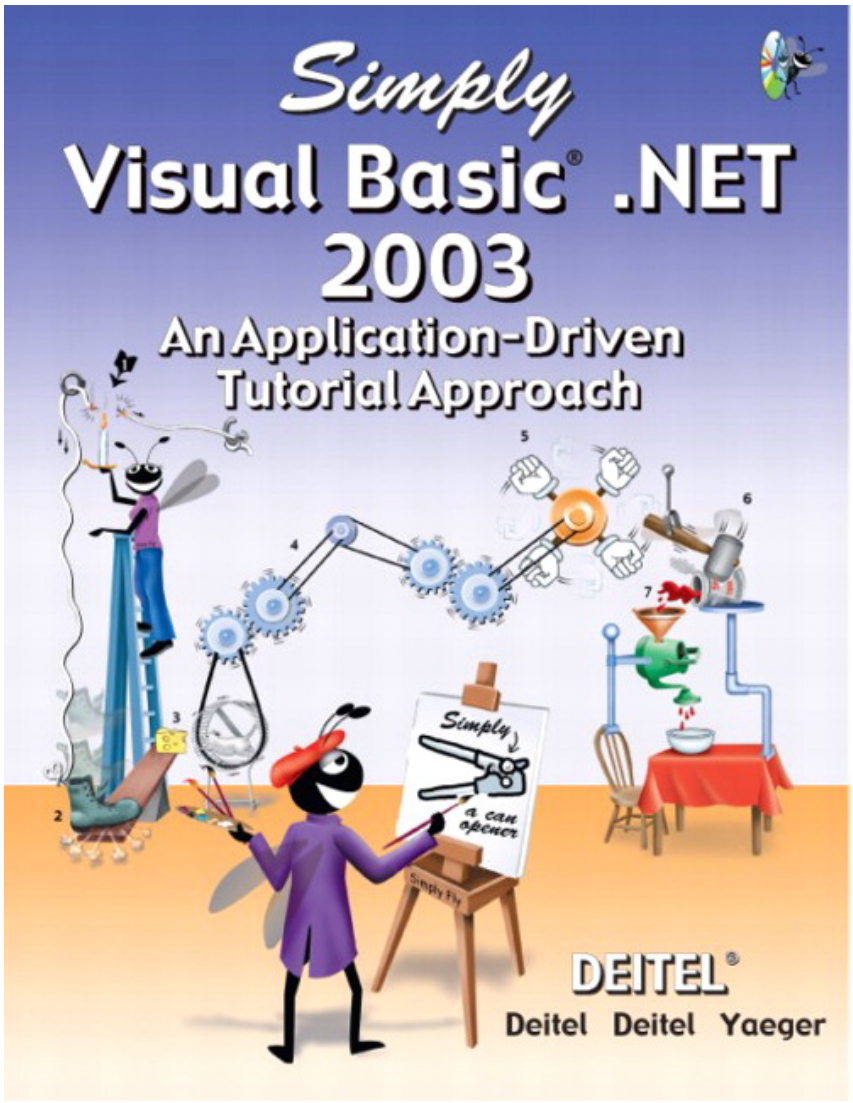
Instructor's Manual

for

Simply Visual Basic® .NET 2003



Deitel Deitel Yaeger

CONTENTS

Preface

Thank you for considering and/or adopting our text *Simply Visual Basic .NET*. If you have not read the preface to *Simply Visual Basic .NET*, we strongly encourage you to do so. The preface contains a careful walkthrough of book's key features. We have worked hard to produce a textbook and ancillaries that we hope you and your students will find valuable.

The following ancillary resources are available:

- *Simply Visual Basic .NET's program examples* are included on a CD-ROM in the back of the book. The examples help instructors prepare lectures faster and aid students in their study of the Visual Basic .NET. The examples are also available for download at **`www.deitel.com`**. When extracting the source code from the ZIP file, you must use a ZIP-file reader such as WinZip (**`www.winzip.com`**) or PKZIP (**`www.pkware.com`**) that understands directories. The file should be extracted into a separate directory such as **`simplyvb1_solutions`**.
- This *Simply Visual Basic .NET Instructor's Manual* contains answers to the exercises in *Simply Visual Basic .NET*. The *Instructor's Manual* CD also contains programming exercise solutions. The programs are separated into directories by tutorial and by project name.
- *Companion Web site* at **`www.prenhall.com/deitel`**.
- *Powerpoint Slide Show* which contains the source code for each program and key discussion points for the examples. Instructors can edit these slides for their own use in classroom discussions.
- *Test Item File* which contains hundreds of multiple-choice questions. Instructors can use these questions to create exams.

We would sincerely appreciate your comments, criticisms and corrections. Please send them to:

`deitel@deitel.com`

We will respond immediately. Please watch our Deitel & Associates, Inc. Web site and our Prentice Hall Web site for book and product updates:

`www.deitel.com`
`www.prenhall.com/deitel`

We would like to thank the extraordinary team of publishing professionals at Prentice Hall who made *Simply Visual Basic .NET* and its ancillaries possible. Our Computer Science editor, Petra Recter, worked closely with us to ensure the timely availability and professional quality of these ancillaries.

Harvey M. Deitel
Paul J. Deitel
Cheryl Yaeger

TUTORIAL 1

Graphing Application

Introducing Computers, the Internet and Visual Basic .NET

Solutions

Instructor's Manual Exercise Solutions Tutorial 1

MULTIPLE-CHOICE QUESTIONS

1.1 The World Wide Web was developed _____.

a) by ARPA
b) at CERN by Tim Berners-Lee
c) before the Internet
d) as a replacement for the Internet

1.2 Microsoft's _____ initiative integrates the Internet and the Web into software development.

a) .NET
b) BASIC
c) Windows
d) W3C

1.3 `TextBoxes`, `Buttons` and `RadioButtons` are examples of _____.

a) platforms
b) high-level languages
c) IDEs
d) controls

1.4 _____ is an example of primary memory.

a) TCP
b) RAM
c) ALU
d) CD-ROM

1.5 Visual Basic .NET is an example of a(n) _____ language, in which single program statements accomplish more substantial tasks.

a) machine
b) intermediate-level
c) high-level
d) assembly

1.6 Which protocol is primarily intended to create a "network of networks?"

a) TCP
b) IP
c) OOP
d) FCL

1.7 A major benefit of _____ programming is that it produces software that is more understandable and better organized than software produced with previously used techniques.

a) object-oriented
b) centralized
c) procedural
d) HTML

1.8 .NET's collection of prepackaged classes and methods is called the _____.

a) NCL
b) WCL
c) FCL
d) PPCM

1.9 The information-carrying capacity of communications lines is called _____.

a) networking
b) secondary storage
c) traffic
d) bandwidth

1.10 Which of these programming languages was specifically created for .NET?

a) C#
b) C++
c) BASIC
d) Visual Basic

Answers: 1.1) b. 1.2) a. 1.3) d. 1.4) b. 1.5) c. 1.6) b. 1.7) a. 1.8) c. 1.9) d. 1.10) a.

EXERCISES

1.11 Categorize each of the following items as either hardware or software:

a) CPU
b) Compiler
c) Input unit
d) A word-processor program
e) A Visual Basic .NET program

Answers: a) hardware. b) software. c) hardware. d) software. e) software.

1.12 Translator programs, such as assemblers and compilers, convert programs from one language (referred to as the source language) to another language (referred to as the target language). Determine which of the following statements are *true* and which are *false*:

a) A compiler translates high-level-language programs into target-language programs.

b) An assembler translates source-language programs into machine-language programs.

c) A compiler translates source-language programs into target-language programs.

d) High-level languages are generally machine dependent.

e) A machine-language program requires translation before it can be run on a computer.

Answers: a) True. b) True. c) True. d) False. High-level languages are generally machine independent. e) False. A machine language program is native to that specific machine and can be run without translation.

1.13 Computers can be thought of as being divided into six units.

a) Which unit can be thought of as "the boss" of the other units?

b) Which unit is the high-capacity "warehouse" and retains information even when the computer is powered off?

c) Which unit might determine whether two items stored in memory are identical?

d) Which unit obtains information from devices like the keyboard and mouse?

Answers: a) CPU. b) Secondary storage unit. c) ALU. d) Input unit.

1.14 Expand each of the following acronyms:

a) W3C

b) TCP/IP

c) OOP

d) FCL

e) HTML

Answers: a) World Wide Web Consortium. b) Transmission Control Protocol/Internet Protocol. c) Object-oriented programming. d) Framework Class Library. e) HyperText Markup Language.

1.15 What are the advantages to using object-oriented programming techniques?

Answer: Programs that use object-oriented programming techniques are easier to understand, correct and modify. The key advantage with using object-oriented programming is that it tends to produce software that is more understandable because it is better organized and has fewer maintenance requirements than software produced with earlier methodologies. OOP helps the programmer build applications faster by reusing existing software components. OOP also helps programmers create new software components that can be reused on future software-development projects.

TUTORIAL 2

Welcome Application

Introducing the Visual Studio® .NET IDE

Solutions

Instructor's Manual Exercise Solutions Tutorial 2

MULTIPLE-CHOICE QUESTIONS

2.1 The _____ integrated development environment is used for creating applications written in .NET programming languages such as Visual Basic.NET.

a) **Solution Explorer** b) Gates
c) Visual Studio .NET d) Microsoft

2.2 The `.vb` filename extension indicates a _____.

a) Visual Basic file b) dynamic help file
c) help file d) very big file

2.3 The pictures on toolbar `Buttons` are called _____.

a) prototypes b) icons
c) tool tips d) tabs

2.4 The _____ allows programmers to modify controls visually, without writing code.

a) **Properties** window b) **Solution Explorer**
c) menu bar d) **Toolbox**

2.5 The _____ hides the **Toolbox** when the mouse pointer is moved outside the **Toolbox**'s area.

a) component-selection feature b) Auto Hide feature
c) pinned command d) minimize command

2.6 A _____ appears when the mouse pointer is positioned over an IDE toolbar icon for a few seconds.

a) drop-down list b) menu
c) tool tip d) down arrow

2.7 The Visual Studio .NET IDE provides _____.

a) help documentation b) a toolbar
c) windows for accessing project files d) All of the above.

2.8 The _____ contains a list of helpful links, such as **Get Started** and **Online Community**.

a) **Solution Explorer** window b) **Properties** window
c) **Start Page** d) **Toolbox** link

2.9 The **Properties** window contains _____.

a) the component object box b) a **Solution Explorer**
c) menus d) a menu bar

2.10 A _____ can be enhanced by adding reusable components such as `Buttons`.

a) control b) `Form`
c) tab d) property

2.11 For Web browsing, Visual Studio .NET includes _____.

a) Web View b) Excel
c) a **Web** tab d) Internet Explorer

2.12 An application's GUI can include _____.

a) toolbars b) icons
c) menus d) All of the above.

2.13 The _____ does not contain a pin icon.

a) **Dynamic Help** window
b) **Solution Explorer** window
c) **Toolbox** window
d) active tab

2.14 When clicked, _____ in the **Solution Explorer** window will expand nodes and _____ will collapse nodes.

a) minus boxes; plus boxes
b) plus boxes; minus boxes
c) up arrows; down arrows
d) left arrows; right arrows

2.15 `Form` _____ specify attributes such as size and position.

a) nodes
b) inputs
c) properties
d) title bars

Answers: 2.1) c. 2.2) a. 2.3) b. 2.4) a. 2.5) b. 2.6) c. 2.7) d. 2.8) c. 2.9) a. 2.10) b. 2.11) d. 2.12) d. 2.13) d. 2.14) b. 2.15) c.

EXERCISES

2.16 ***(Closing and Opening the Start Page)*** In this exercise, you will learn how to close and reopen the **Start Page**. To accomplish this task, perform the following steps:

a) Close Visual Studio .NET if it is open by selecting **File > Exit** or by clicking its close box.

b) Start Visual Studio .NET.

c) Close the **Start Page** by clicking its close box (Fig. 2.30).

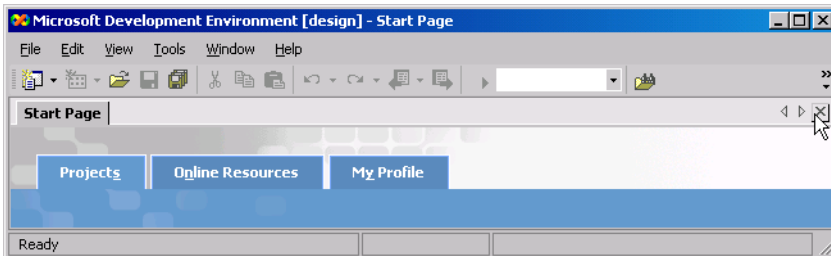


Figure 2.30 Closing the **Start Page**.

d) Select **Help > Show Start Page** to display the **Start Page**.

2.17 ***(Enabling Auto Hide for the Solution Explorer Window)*** In this exercise, you will learn how to use the **Solution Explorer** window's Auto Hide feature by performing the following steps:

a) Open the **Start Page**.

b) In the **Get Started** page (displayed by default), click the **Open Project** `Button` to display the **Open Project** dialog. You can skip to step e) if the **Welcome** application is already open.

c) In the **Open Project** dialog, navigate to `C:\SimplyVB\Welcome` and click **Open**.

d) In the **Open Project** dialog, select `Welcome.sln` and click **Open**.

e) Position the mouse pointer on the vertical pin icon in the **Solution Explorer** window's title bar. After a few seconds, a tool tip appears displaying the words **Auto Hide** (Fig. 2.31).

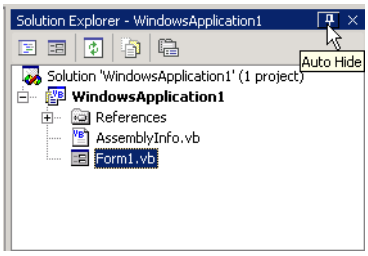


Figure 2.31 Enabling Auto Hide.

f) Click the vertical pin icon. This action causes a **Solution Explorer** tab to appear on the right side of the IDE. The vertical pin icon changes to a horizontal pin icon (Fig. 2.32). Auto Hide has now been enabled for the **Solution Explorer** window.

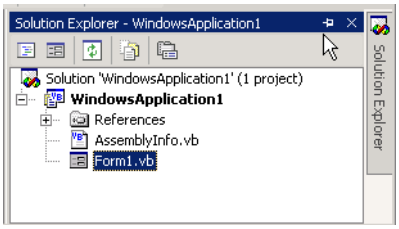


Figure 2.32 **Solution Explorer** window with Auto Hide enabled.

g) Position the mouse pointer outside the **Solution Explorer** window to hide the window.

h) Position the mouse pointer on the **Solution Explorer** tab to view the **Solution Explorer** window.

2.18 (***Sorting Properties Alphabetically in the Properties Window***) In this exercise, you will learn how to sort the **Properties** window's properties alphabetically by performing the following steps:

a) Open the **Welcome** application by performing steps a) through d) of Exercise 2.17. If the **Welcome** application is already open, you can skip this step.

b) Locate the **Properties** window. If it is not visible, select **View > Properties Window** to display the **Properties** window.

c) To sort properties alphabetically, click the **Properties** window's alphabetic icon (Fig. 2.33). The properties now display in alphabetic order.

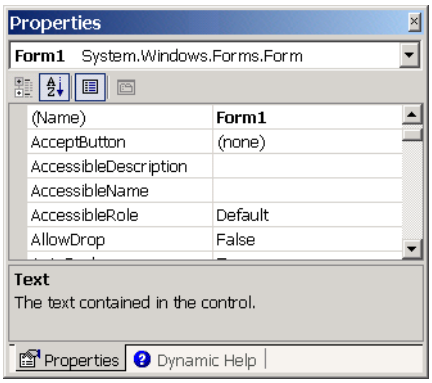


Figure 2.33 Sorting properties alphabetically.

TUTORIAL 3

Welcome Application

Introduction to Visual Programming
Solutions

Instructor's Manual Exercise Solutions Tutorial 3

MULTIPLE-CHOICE QUESTIONS

3.1 Property _____ determines the `Form`'s background color.

a) `BackColor` b) `BackgroundColor`
c) `RGB` d) `Color`

3.2 To save all the solution's files, select _____.

a) **Save > Solution > Save Files** b) **File > Save**
c) **File > Save All** d) **File > Save As...**

3.3 When the ellipsis `Button` to the right of the **Font** property value is clicked, the _____ is displayed.

a) **Font Property** dialog b) **New Font** dialog
c) **Font Settings** dialog d) **Font** dialog

3.4 `PictureBox` property _____ contains a preview of the image displayed in the `PictureBox`.

a) `Picture` b) `ImageName`
c) `Image` d) `PictureName`

3.5 The _____ tab allows you to create your own color.

a) **Custom** b) **Web**
c) **System** d) **User**

3.6 The `PictureBox` class has namespace _____.

a) `System.Windows.Forms` b) `System.Form.Form`
c) `System.Form.Font` d) `System.Form.Control`

3.7 A `Label` control displays the text specified by property _____.

a) `Caption` b) `Data`
c) `Text` d) `Name`

3.8 In _____ mode, the application is executing.

a) start b) run
c) break d) design

3.9 The _____ command prevents programmers from accidentally altering the size and location of the `Form`'s controls.

a) **Lock Controls** b) **Anchor Controls**
c) **Lock** d) **Bind Controls**

3.10 Pixels are _____.

a) picture elements b) controls in the **Toolbox**
c) a set of fonts d) a set of colors on the **Web** tab

Answers: 3.1) a. 3.2) c. 3.3) d. 3.4) c. 3.5) a. 3.6) a. 3.7) c. 3.8) b. 3.9) a. 3.10) a.

EXERCISES

For Exercises 3.11–3.16, you are asked to create the GUI shown in each exercise. You will use the visual programming techniques presented in this tutorial to create a variety of GUIs. Because you are creating only GUIs, your applications will not be fully operational. For example, the **Calculator** *GUI in Exercise 3.11 will not behave like a calculator when its* `Buttons` *are clicked. You will learn how to make your applications fully operational in later tutorials. Create each application as a separate project.*

3.11 ***(Calculator GUI)*** Create the GUI for the calculator shown in Fig. 3.33.

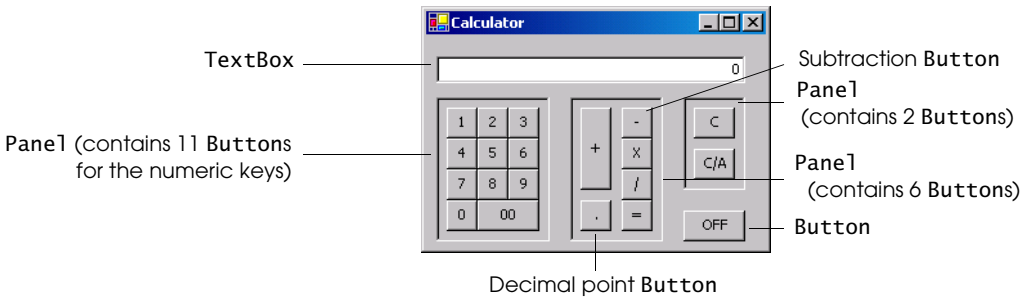
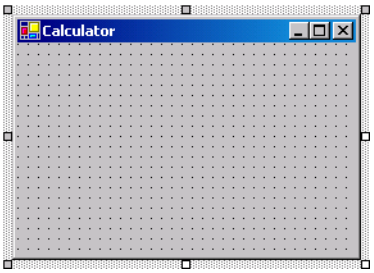


Figure 3.33 **Calculator** GUI.

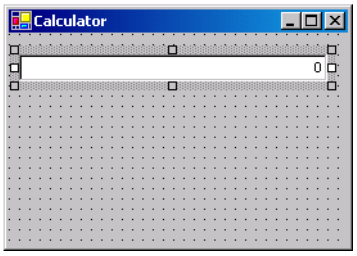
a) ***Creating a new project.*** Create a new **Windows Application** named `Calculator`.

b) ***Renaming the Form file.*** Name the `Form` file `Calculator.vb`.

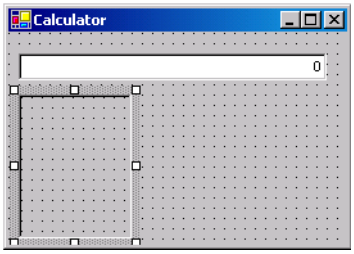
c) ***Manipulating the Form's properties.*** Change the `Size` property of the `Form` to `272, 192`. Change the `Text` property of the `Form` to `Calculator`. Change the `Font` property to `Tahoma`.



d) ***Adding a `TextBox` to the Form.*** Add a `TextBox` control by double clicking it in the **Toolbox**. A `TextBox` control is used to enter input into applications. Set the `TextBox`'s `Text` property in the **Properties** window to `0`. Change the `Size` property to `240, 21`. Set the `TextAlign` property to `Right`; this right aligns text displayed in the `TextBox`. Finally, set the `TextBox`'s `Location` property to `8, 16`.

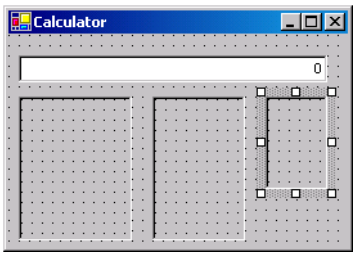


e) ***Adding the first `Panel` to the Form.*** `Panel` controls are used to group other controls. Double click the `Panel` icon (Panel) in the **Toolbox** to add a `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D` to make the inside of the `Panel` appear recessed. Change the `Size` property to `88, 112`. Finally, set the `Location` property to `8, 48`. This `Panel` contains the calculator's numeric keys.

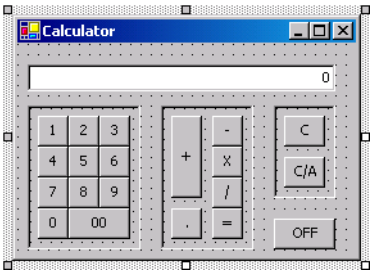


f) ***Adding the second `Panel` to the Form.*** Click the `Form`. Double click the `Panel` icon in the **Toolbox** to add another `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D`. Change the `Size` property to `72, 112`. Finally, set the `Location` property to `112, 48`. This `Panel` contains the calculator's operator keys.

g) ***Adding the third (and last) `Panel` to the Form.*** Click the `Form`. Double click the `Panel` icon in the **Toolbox** to add another `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D`. Change the `Size` property to `48, 72`. Finally, set the `Location` property to `200, 48`. This `Panel` contains the calculator's **C** (clear) and **C/A** (clear all) keys.



h) ***Adding `Buttons` to the Form.*** There are 20 `Buttons` on the calculator. To add a `Button` to a `Panel`, double click the `Button` control (ab Button) in the **Toolbox**. Then add the `Button` to the `Panel` by dragging and dropping it on the `Panel`. Change the `Text` property of each `Button` to the calculator key it represents. The value you enter in the `Text` property will appear on the face of the `Button`. Finally, resize the `Buttons`, using their `Size` properties. Each `Button` labelled `0`–`9`, `x`, `/`, `-`, `=` and `.` should have a size of `24, 24`. The **00** and **OFF** `Buttons` have size `48, 24`. The **+** `Button` is sized `24, 64`. The **C** (clear) and **C/A** (clear all) `Buttons` are sized `32, 24`.



i) ***Saving the project.*** Select **File > Save All** to save your changes.

3.12 ***(Alarm Clock GUI)*** Create the GUI for the alarm clock in Fig. 3.34.

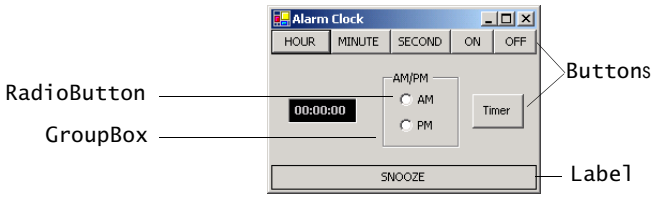
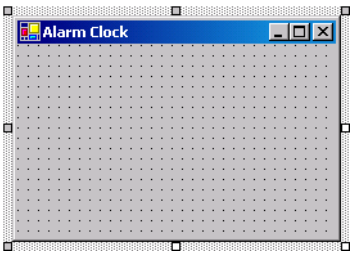


Figure 3.34 **Alarm Clock** GUI.

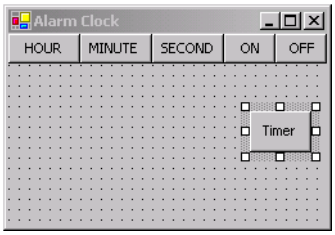
a) ***Creating a new project.*** Create a new **Windows Application** named `AlarmClock`.

b) ***Renaming the Form file.*** Name the `Form` file `AlarmClock.vb`.

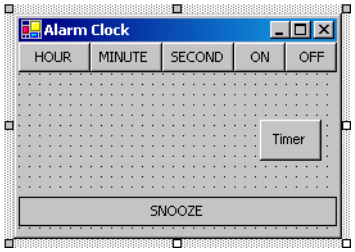
c) ***Manipulating the Form's properties.*** Change the `Size` property of the `Form` to `256, 176`. Change the `Text` property of the `Form` to `Alarm Clock`. Change the `Font` property to `Tahoma`.



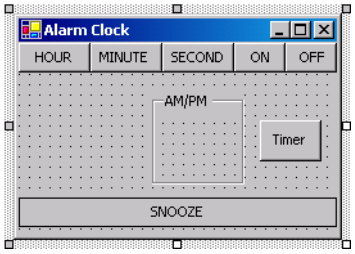
d) ***Adding `Buttons` to the `Form`.*** Add six `Buttons` to the `Form`. Change the `Text` property of each **Button** to the appropriate text. Change the `Size` properties of the **Hour**, **Minute** and **Second** `Buttons` to `56, 23`. The **ON** and **OFF** `Buttons` get size `40, 23`. The **Timer** `Button` gets size `48, 32`. Align the `Buttons` as shown in Fig. 3.34.



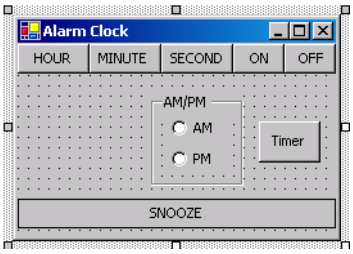
e) ***Adding a `Label` to the `Form`.*** Add a `Label` to the `Form`. Change the `Text` property to **Snooze**. Set its `Size` to `248, 23`. Set the `Label`'s `TextAlign` property to `MiddleCenter`. Finally, to draw a border around the edge of the **Snooze** `Label`, change the `BorderStyle` property of the **Snooze** `Label` to `FixedSingle`.



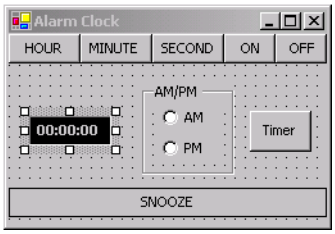
f) ***Adding a `GroupBox` to the `Form`. `GroupBoxes`*** are like `Panels`, except that `GroupBoxes` can display a title. To add a `GroupBox` to the `Form`, double click the `GroupBox` control (GroupBox) in the **Toolbox**. Change the `Text` property to **AM/PM**, and set the `Size` property to `72, 72`. To place the `GroupBox` in the correct location on the `Form`, set the `Location` property to `104, 38`.



g) ***Adding AM/PM `RadioButtons` to the `GroupBox`.*** Add two `RadioButtons` to the `Form` by dragging the `RadioButton` control (RadioButton) in the **Toolbox** and dropping it onto the `GroupBox` twice. Change the `Text` property of one `RadioButton` to `AM` and the other to `PM`. Then place the `RadioButtons` as shown in Fig. 3.34 by setting the `Location` of the **AM** `RadioButton` to `16, 16` and that of the **PM** `RadioButton` to `16, 40`. Set their `Size` properties to `48, 24`.



h) ***Adding the time `Label` to the `Form`.*** Add a `Label` to the `Form` and change its `Text` property to `00:00:00`. Change the `BorderStyle` property to `Fixed3D` and the `BackColor` to `Black`. Set the `Size` property to `64, 23`. Use the `Font` property to make the time bold. Change the `ForeColor` to `Silver` (located in the **Web** tab) to make the time stand out against the black background. Set `TextAlign` to `MiddleCenter` to center the text in the `Label`. Position the `Label` as shown in Fig. 3.34.



i) ***Saving the project.*** Select **File > Save All** to save your changes.

3.13 ***(Microwave Oven GUI)*** Create the GUI for the microwave oven shown in Fig. 3.35.

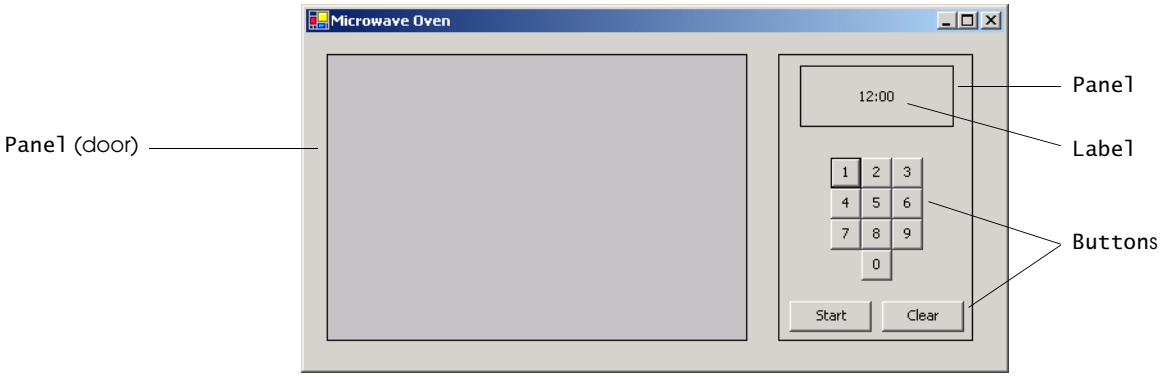
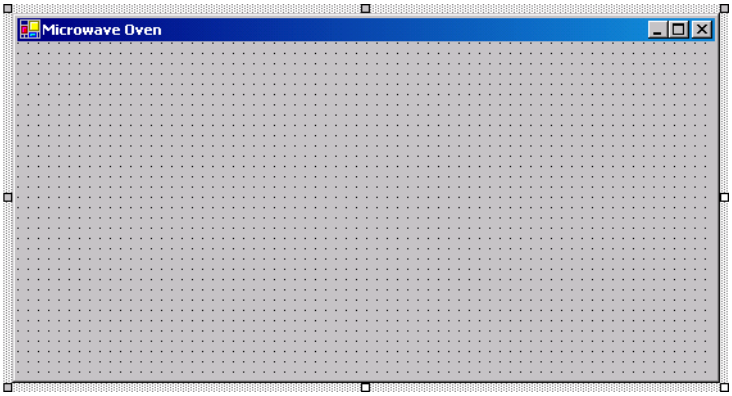


Figure 3.35 Microwave Oven GUI.

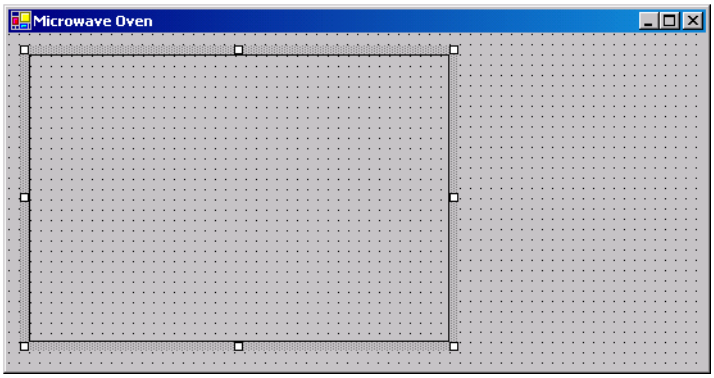
a) ***Creating a new project.*** Create a new **Windows Application** named `Microwave`.

b) ***Renaming the `Form` file.*** Name the `Form` file `Microwave.vb`.

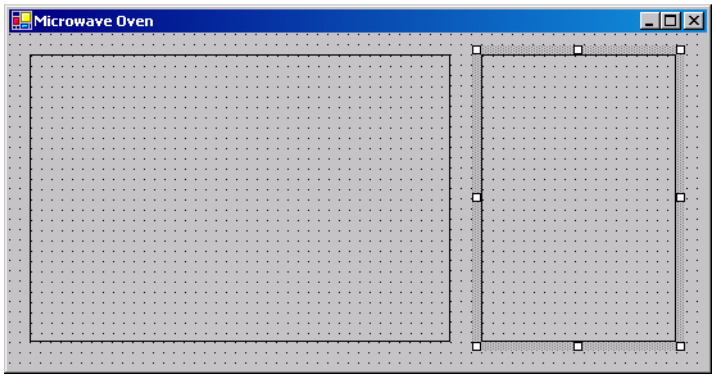
c) ***Manipulating the `Form`'s properties.*** Change the `Size` property of the `Form` to `552, 288`. Change the `Text` property of the `Form` to `Microwave Oven`. Change the `Font` property to `Tahoma`.



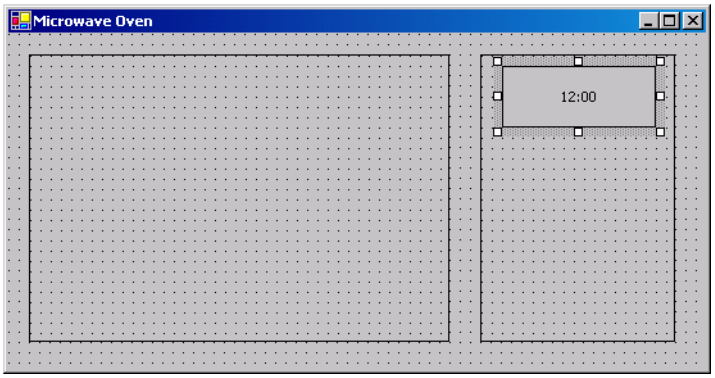
d) ***Adding the microwave oven door.*** Add a `Panel` to the `Form` by double clicking the `Panel` (Panel) in the **Toolbox**. Select the `Panel` and change the `BackColor` property to `Silver` (located in the **Web** tab) in the **Properties** window. Then change the `Size` to `328, 224`. Next, change the `BorderStyle` property to `FixedSingle`.



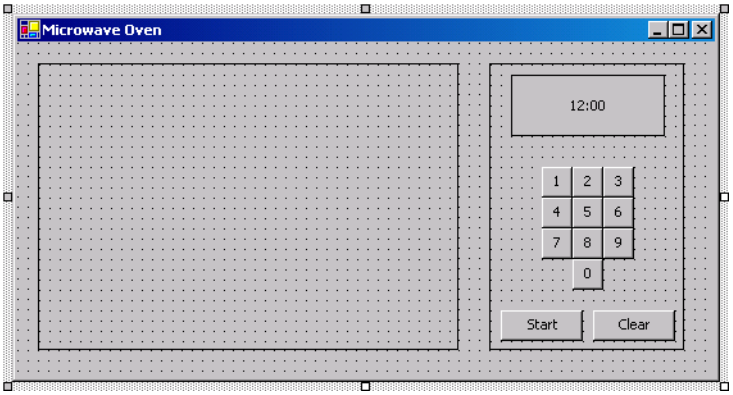
e) ***Adding another Panel.*** Add another `Panel` and change its `Size` to `152, 224` and its `BorderStyle` to `FixedSingle`. Place the `Panel` to the right of the door `Panel`, as shown in Fig. 3.35.



f) ***Adding the microwave oven clock.*** Add a `Label` to the right `Panel` by clicking the `Label` in the **Toolbox** once, then clicking once inside the right `Panel`. Change the `Label`'s `Text` to `12:00`, `BorderStyle` to `FixedSingle` and `Size` to `120, 48`. Change `TextAlign` to `MiddleCenter`. Place the clock as shown in Fig. 3.35.



g) ***Adding a keypad to the microwave oven.*** Place a `Button` in the right `Panel` by clicking the `Button` control in the `Toolbox` once, then clicking inside the `Panel`. Change the `Text` to `1` and the `Size` to `24, 24`. Repeat this process for nine more `Buttons`, changing the `Text` property in each to the next number in the keypad. Then add the **Start** and **Clear** `Buttons`, each of `Size 64, 24`. Do not forget to set the `Text` properties for each of these `Buttons`. Finally, arrange the `Buttons` as shown in Fig. 3.35. The **1** `Button` is located at `40, 80` and the **Start** `Button` is located at `8, 192`.



h) ***Saving the project.*** Select **File > Save All** to save your changes.

3.14 ***(Cell Phone GUI)*** Create the GUI for the cell phone shown in Fig. 3.36.

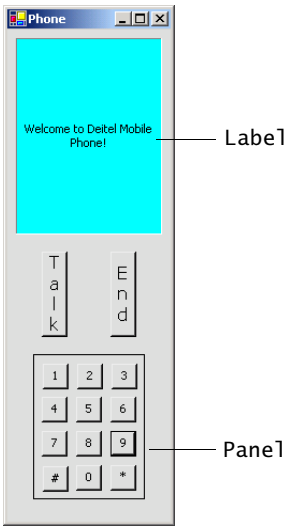
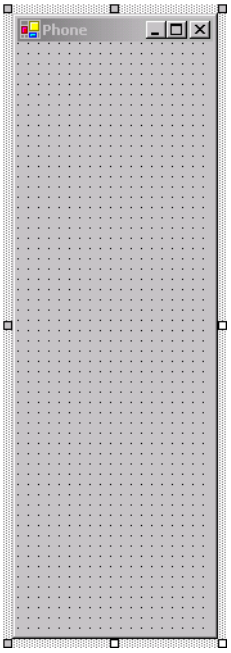


Figure 3.36 Cell **Phone** GUI.

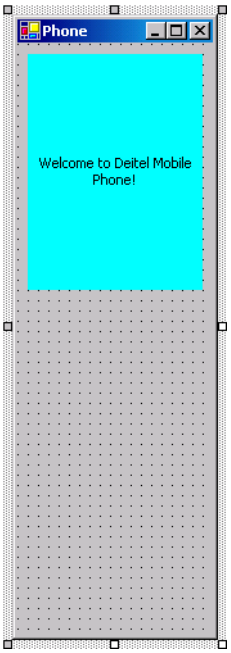
a) ***Creating a new project.*** Create a new **Windows Application** named `Phone`.

b) ***Renaming the Form file.*** Name the `Form` file `Phone.vb`.

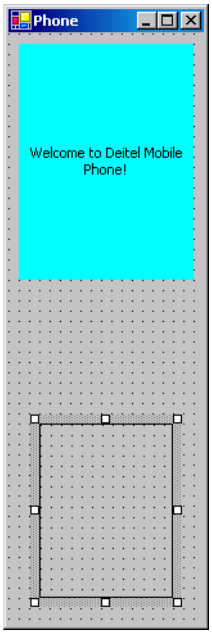
c) ***Manipulating the Form's properties.*** Change the `Form`'s `Text` property to `Phone` and the `Size` to `160, 488`. Change the `Font` property to `Tahoma`.



d) ***Adding the display Label.*** Add a `Label` to the `Form`. Change its `BackColor` to `Aqua` (in the **Web** tab palette), the `Text` to `Welcome to Deitel Mobile Phone!` and the `Size` to `136, 184`. Change the `TextAlign` property to `MiddleCenter`. Then place the `Label` as shown in Fig. 3.36.

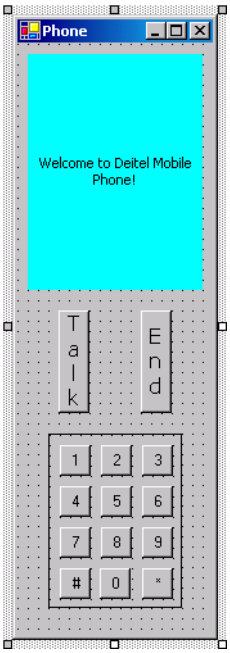


e) ***Adding the keypad Panel.*** Add a `Panel` to the `Form`. Change its `BorderStyle` property to `FixedSingle` and its `Size` to `104, 136`.



f) ***Adding the keypad `Buttons`.*** Add the keypad `Buttons` to the `Form` (12 `Buttons` in all). Each `Button` on the number pad should be of `Size` `24`, `24` and should be placed in the `Panel`. Change the `Text` property of each `Button` such that numbers `0`–`9`, the pound (`#`) and the star (`*`) keys are represented. Then add the final two `Buttons` such that the `Text` property for one is `Talk` and the other is `End`. Change the `Size` of each `Button` to `24`, `80`, and notice how the small `Size` causes the `Text` to align vertically. Also change each `Button`'s `Font` size to 12 points.

g) ***Placing the controls.*** Arrange all the controls so that your GUI looks like Fig. 3.36.



h) ***Saving the project.*** Select **File > Save All** to save your changes.

3.15 ***(Vending Machine GUI)*** Create the GUI for the vending machine in Fig. 3.37.

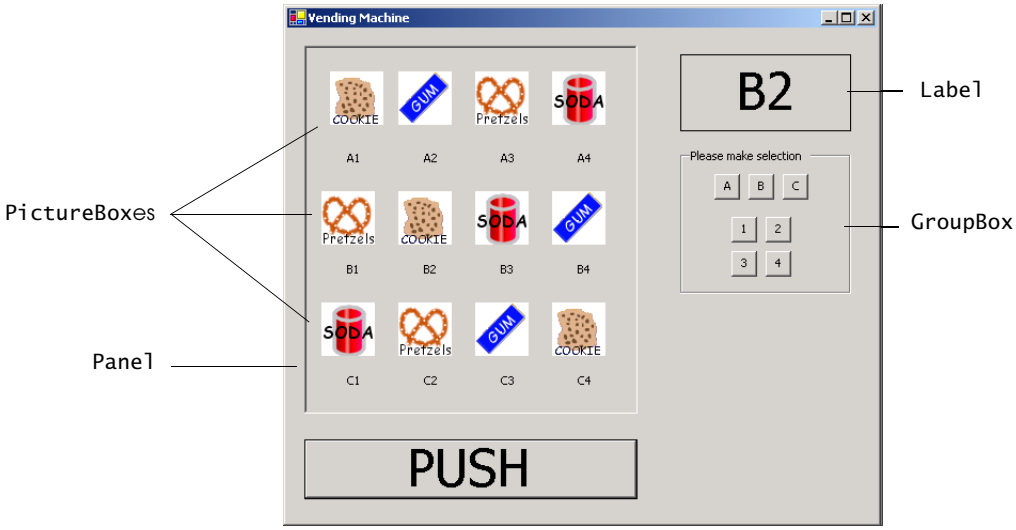
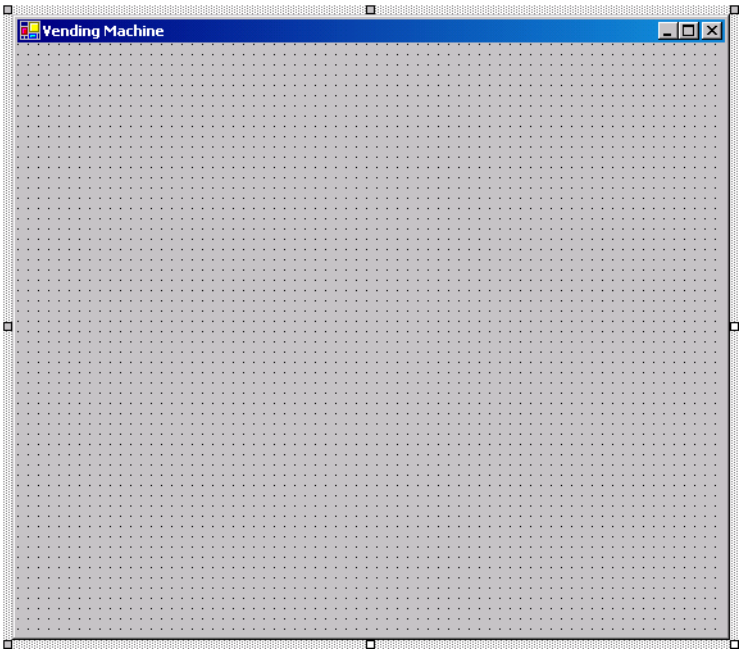


Figure 3.37 Vending Machine GUI.

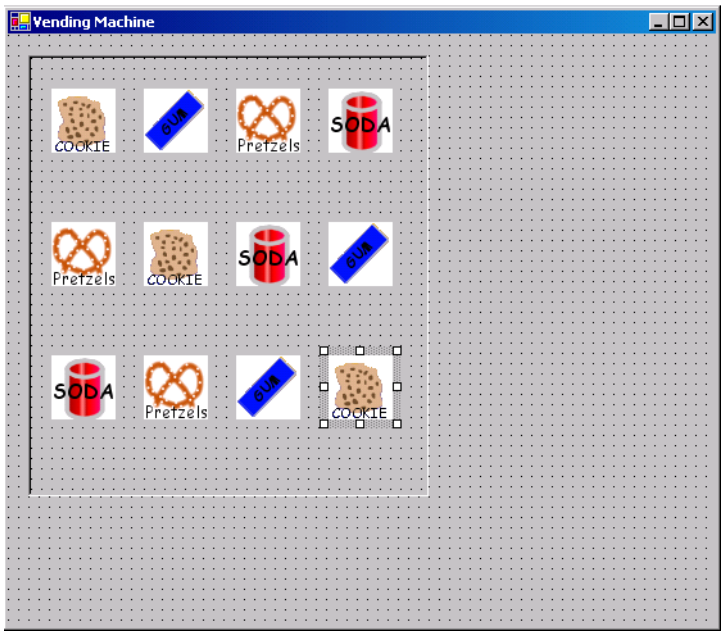
a) ***Creating a new project.*** Create a new **Windows Application** named `VendingMachine`.

b) ***Renaming the Form file.*** Name the `Form` file `VendingMachine.vb`.

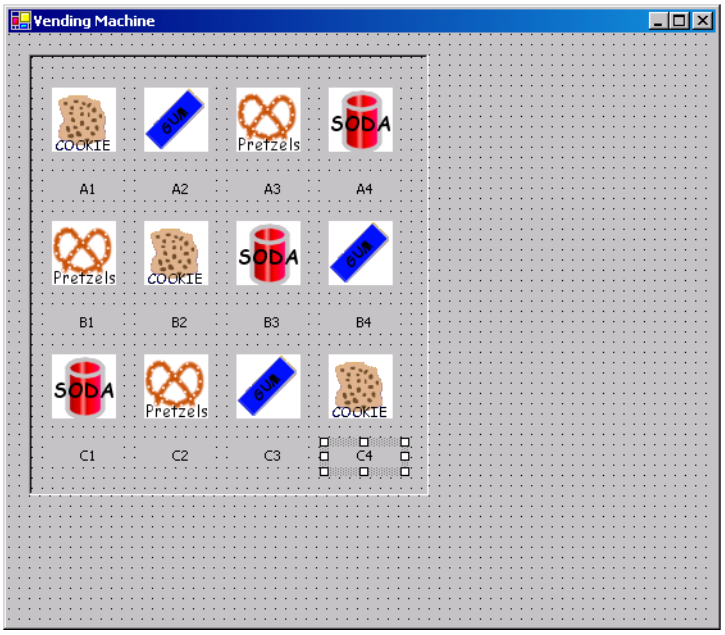
c) ***Manipulating the Form's properties.*** Set the `Text` property of the `Form` to `Vending Machine` and the `Size` to `560, 488`. Change the `Font` property to `Tahoma`.



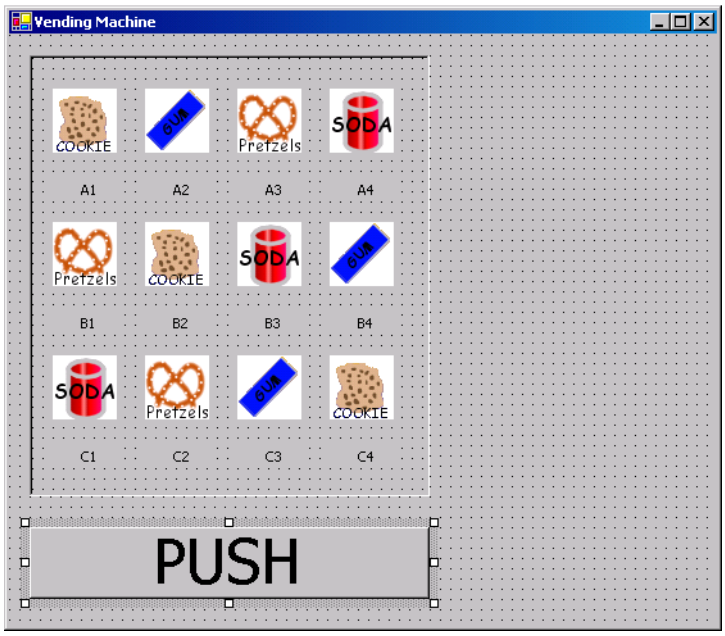
d) ***Adding the food selection Panel.*** Add a `Panel` to the `Form`, and change its `Size` to `312, 344` and `BorderStyle` to `Fixed3D`. Add a `PictureBox` to the `Panel`, and change its `Size` to `50, 50`. Then set the `Image` property by clicking the ellipsis `Button` and choosing a file from the `C:\Examples\Tutorial03\ExerciseImages\VendingMachine` directory. Repeat this process for 11 more `PictureBoxes`.



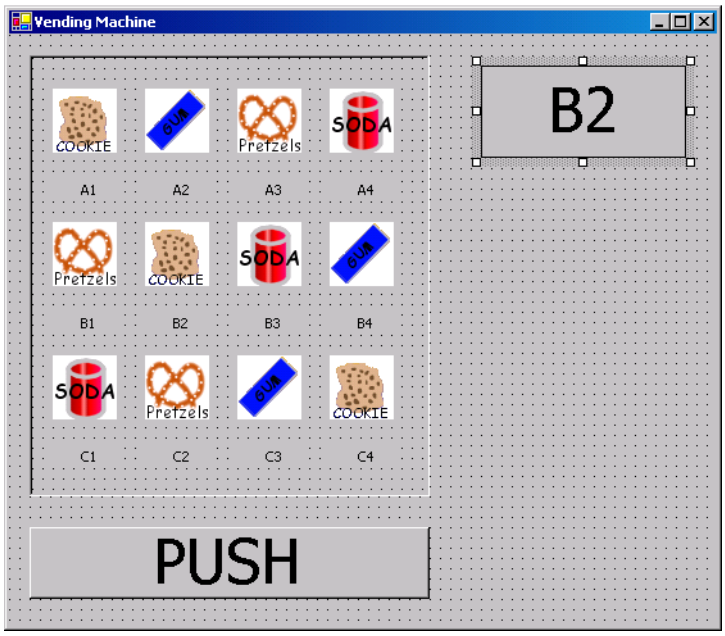
e) ***Adding `Labels` for each vending item.*** Add a `Label` under each `PictureBox`. Change the `Text` property of the `Label` to `A1`, the `TextAlign` property to `TopCenter` and the `Size` to `56, 16`. Place the `Label` so that it is located as in Fig. 3.37. Repeat this process for `A2` through `C4` (11 `Labels`).



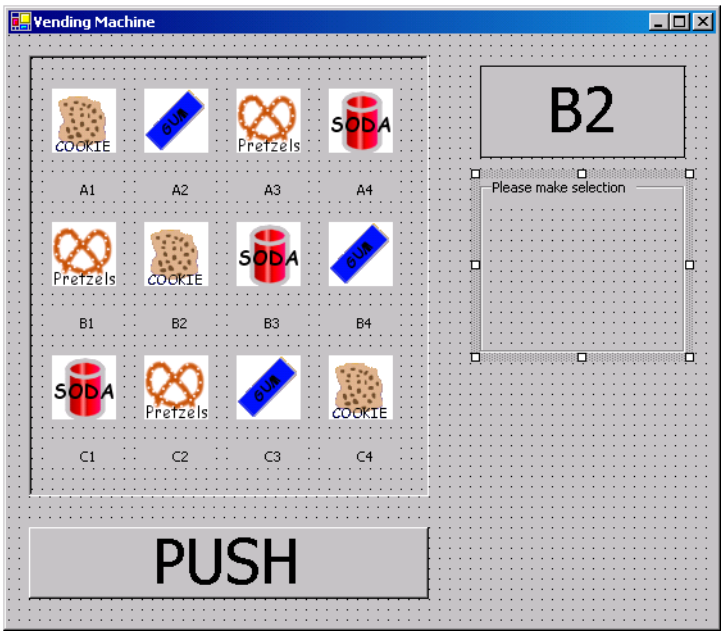
f) ***Creating the vending machine door (as a `Button`).*** Add a `Button` to the `Form` by dragging the `Button` control in the **Toolbox** and dropping it below the `Panel`. Change the `Button`'s `Text` property to `PUSH`, its `Font Size` to `36` and its `Size` to `312, 56`. Then place the `Button` on the `Form` as shown in Fig. 3.37.



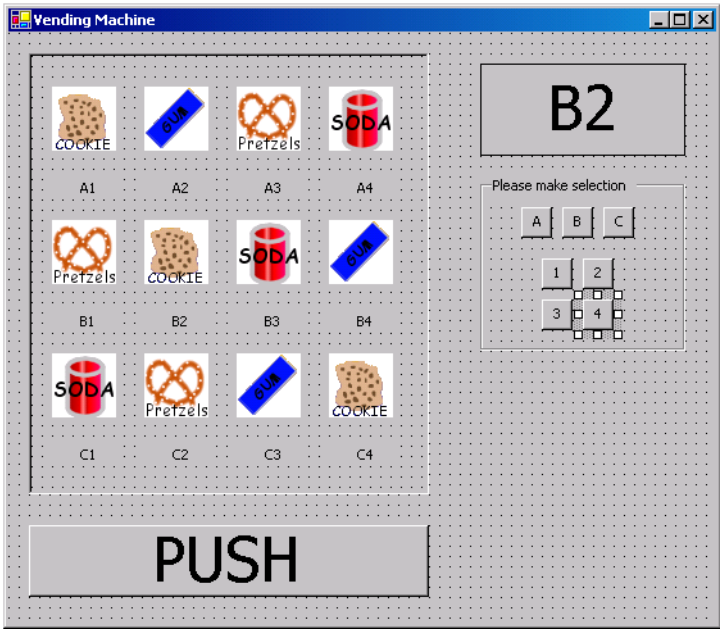
g) ***Adding the selection display Label.*** Add a `Label` to the `Form`, and change the `Text` property to `B2`, `BorderStyle` to `FixedSingle`, `Font Size` to `36`, `TextAlign` to `MiddleCenter` and `Size` to `160, 72`.



h) ***Grouping the input Buttons.*** Add a `GroupBox` below the `Label`, and change the `Text` property to `Please make selection` and the `Size` to `160, 136`.



i) ***Adding the input Buttons.*** Finally, add `Buttons` to the `GroupBox`. For the seven `Buttons`, change the `Size` property to `24, 24`. Then change the `Text` property of the `Buttons` such that each `Button` has one of the values `A`, `B`, `C`, `1`, `2`, `3` or `4`, as shown in Fig. 3.37. When you are done, move the controls on the `Form` so that they are aligned as shown in the figure.



j) ***Saving the project.*** Select **File > Save All** to save your changes.

Programming Challenge ▶ **3.16** ***(Radio GUI)*** Create the GUI for the radio in Fig. 3.38. [*Note:* All colors used in this exercises are from the **Web** palette.]

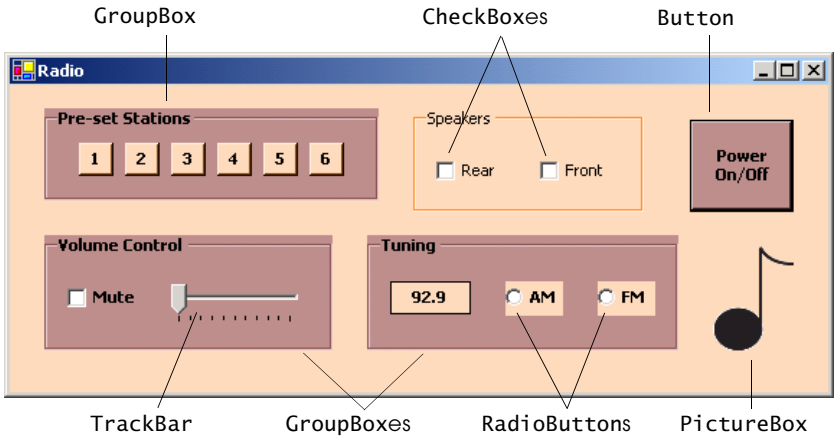


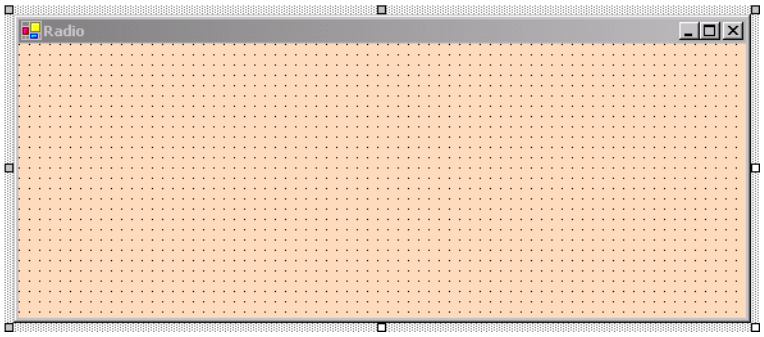
Figure 3.38 **Radio** GUI.

In this exercise, you will create this GUI on your own. Feel free to experiment with different control properties. For the image in the `PictureBox`, use the file (`MusicNote.gif`) found in the `C:\Examples\Tutorial03\ExerciseImages\Radio` directory.

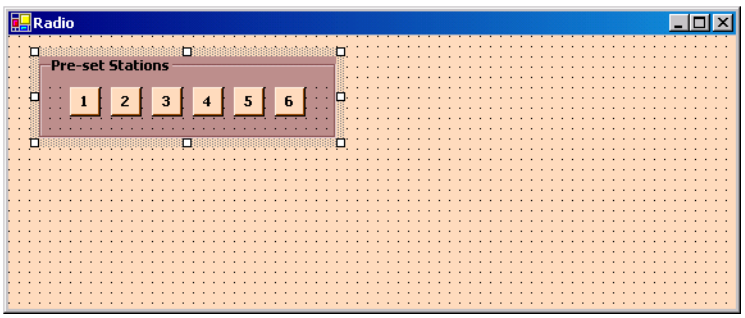
a) ***Creating a new project.*** Create a new **Windows Application** named `Radio`.

b) ***Renaming the Form file.*** Name the `Form` file `Radio.vb`.

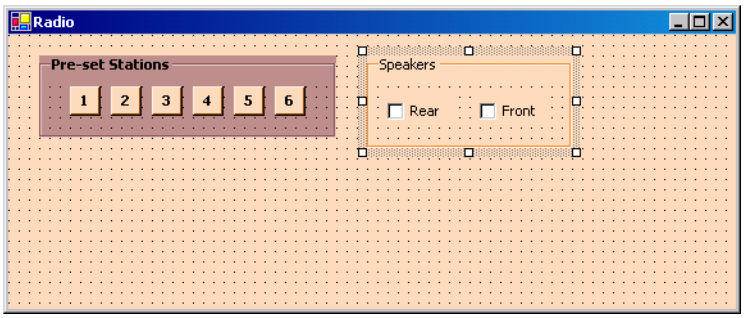
c) ***Manipulating the Form's properties.*** Change the `Form`'s `Text` property to `Radio` and the `Size` to `576, 240`. Change the `Font` property to `Tahoma`. Set `BackColor` to `PeachPuff`.



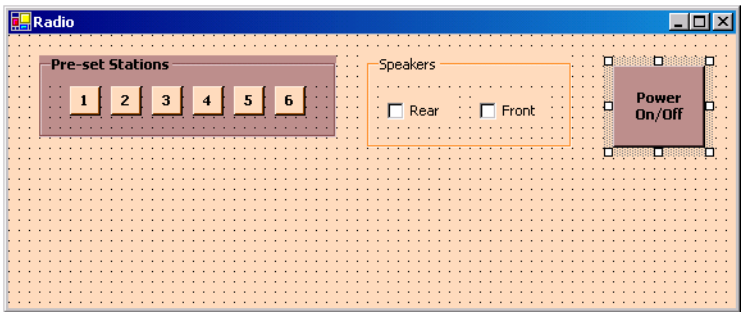
d) ***Adding the Pre-set Stations GroupBox and Buttons.*** Add a `GroupBox` to the `Form`. Set its `Size` to `232, 64`, its `Text` to `Pre-set Stations`, its `ForeColor` to `Black` and its `BackColor` to `RosyBrown`. Change its `Font` to `bold`. Finally, set its `Location` to `24, 16`. Add six `Buttons` to the `GroupBox`. Set each `BackColor` to `PeachPuff` and each `Size` to 24, 23. Change the `Buttons`' `Text` properties to `1`, `2`, `3`, `4`, `5`, `6`, respectively.



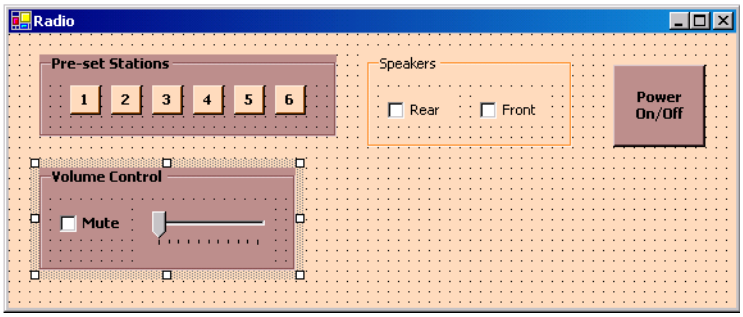
e) ***Adding the Speakers GroupBox and CheckBoxes.*** Add a `GroupBox` to the `Form`. Set its `Size` to `160, 72`, its `Text` to `Speakers` and its `ForeColor` to `Black`. Set its `Location` to `280, 16`. Add two `CheckBoxes` to the `Form`. Set each `CheckBox`'s `Size` to `56, 24`. Set the `Text` properties for the `CheckBoxes` to `Rear` and `Front`.



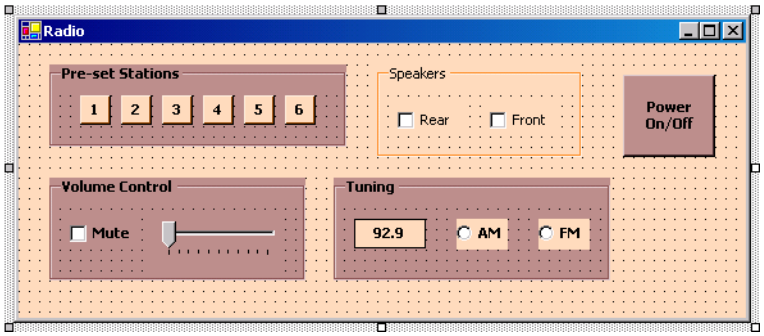
f) ***Adding the Power On/Off Button.*** Add a `Button` to the `Form`. Set its `Text` to `Power On/Off`, its `BackColor` to `RosyBrown`, its `ForeColor` to `Black` and its `Size` to `72, 64`. Change its `Font` style to `Bold`.



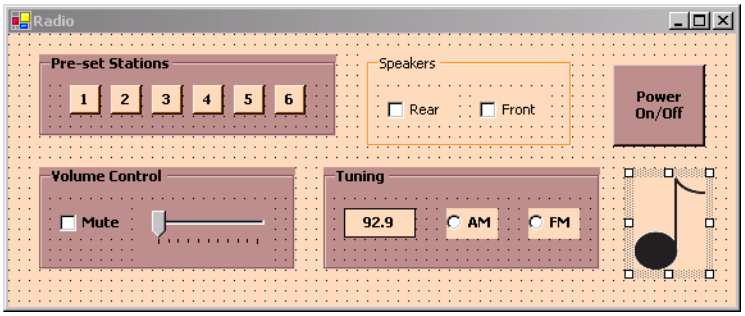
g) ***Adding the Volume Control GroupBox, the Mute CheckBox and the Volume TrackBar.*** Add a `GroupBox` to the `Form`. Set its `Text` to `Volume Control`, its `BackColor` to `RosyBrown`, its `ForeColor` to `Black` and its `Size` to `200, 80`. Set its `Font` style to `Bold`. Add a `CheckBox` to the `GroupBox`. Set its `Text` to `Mute` and its `Size` to `56, 24`. Add a `TrackBar` to the `GroupBox`.



h) ***Adding the Tuning GroupBox, the radio station Label and the AM/FM RadioButtons.*** Add a `GroupBox` to the `Form`. Set its `Text` to `Tuning`, its `ForeColor` to `Black` and its `BackColor` to `RosyBrown`. Set its `Font` style to `Bold` and its `Size` to `216, 80`. Add a `Label` to the `Form`. Set its `BackColor` to `PeachPuff`, its `ForeColor` to `Black`, its `BorderStyle` to `FixedSingle`, its `Font` style to `Bold`, its `TextAlign` to `MiddleCenter` and its `Size` to `56, 23`. Set its `Text` to `92.9`. Place the `Label` as shown in the figure. Add two `RadioButtons` to the `GroupBox`. Change the `BackColor` to `PeachPuff` and change the `Size` to `40,24`. Set one's `Text` to `AM` and the other's `Text` to `FM`.



i) ***Adding the image.*** Add a `PictureBox` to the `Form`. Set its `BackColor` to `Transparent`, its `SizeMode` to `StretchImage` and its `Size` to `56, 72`. Set its `Image` property to `C:\Examples\Tutorial03\ExerciseImages\Radio\MusicNote.gif`.



j) ***Saving the project.*** Select **File > Save All** to save your changes.

TUTORIAL 4

Designing the Inventory Application

Introducing `TextBoxes` and `Buttons`

Solutions

Instructor's Manual Exercise Solutions Tutorial 4

MULTIPLE-CHOICE QUESTIONS

4.1 A new Windows application is created by selecting _____ from the **File** menu.

a) **New > Program** b) **New > File...**
c) **New > Project...** d) **New > Application**

4.2 A `Label`'s `BorderStyle` property can be set to _____.

a) `Fixed3D` b) `Single`
c) `3D` d) All of the above.

4.3 When creating a `Label`, you can specify the _____ of that `Label`.

a) alignment of the text b) border style
c) size d) All of the above.

4.4 Changing the value stored in the _____ property will change the name of the `Form` file.

a) `Name` b) `File`
c) `File Name` d) `Full Path`

4.5 _____ should be used to prefix all `TextBox` names.

a) `txt` b) `tbx`
c) `Frm` d) `tbn`

4.6 A(n) _____ helps the user understand a control's purpose.

a) `Button` b) descriptive `Label`
c) output `Label` d) title bar

4.7 A _____ is a control in which the user can enter data from a keyboard.

a) `Button` b) `TextBox`
c) `Label` d) `PictureBox`

4.8 A descriptive `Label` uses _____.

a) sentence-style capitalization b) book-title capitalization
c) a colon at the end of its text d) Both a and c.

4.9 You should use the _____ font in your Windows applications.

a) Tahoma b) MS Sans Serif
c) Times d) Palatino

4.10 _____ should be used to prefix all `Button` names.

a) `but` b) `lbl`
c) `Frm` d) `btn`

Answers: 4.1) c. 4.2) a. 4.3) d. 4.4) c. 4.5) a. 4.6) b. 4.7) b. 4.8) d. 4.9) a. 4.10) d.

EXERCISES

At the end of each tutorial, you will find a summary of new GUI design tips listed in the GUI Design Guidelines section. A cumulative list of GUI design guidelines, organized by control appears in Appendix C. In these exercises, you will find Visual Basic .NET `Forms` that do not follow the GUI design guidelines presented in this tutorial. For each exercise, you must modify control properties so that your end result is consistent with the guidelines presented in the tutorial. Note that these applications do not provide any functionality.

4.11 ***(Address Book GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for an address book (Fig. 4.24).

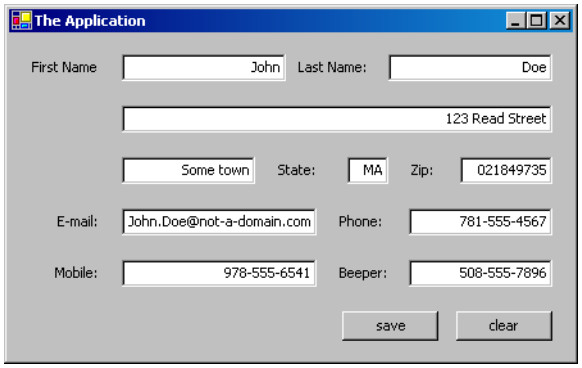


Figure 4.24 **Address Book** application without GUI design guidelines applied.

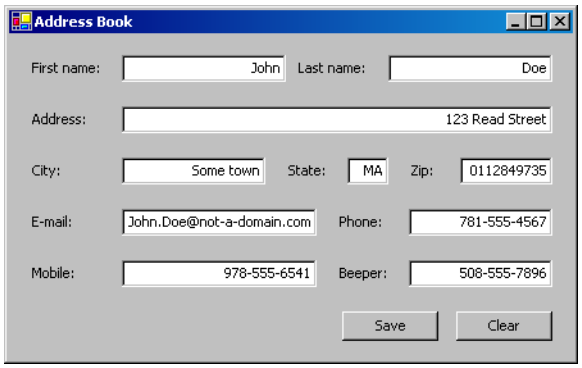
a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial04\Exercises\AddressBook` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AddressBook.sln` in the `AddressBook` directory to open the application.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

Answer:



1. Change the `Form`'s title (`Text` property).
2. All `TextBoxes` should have corresponding `Labels`.
3. `Labels` indicating control usage should use sentence-style capitalization.
4. `Buttons` should use book-title capitalization.
5. Each descriptive `Label` text should end with a colon.

4.12 ***(Mortgage Calculator GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for a mortgage calculator (Fig. 4.25).

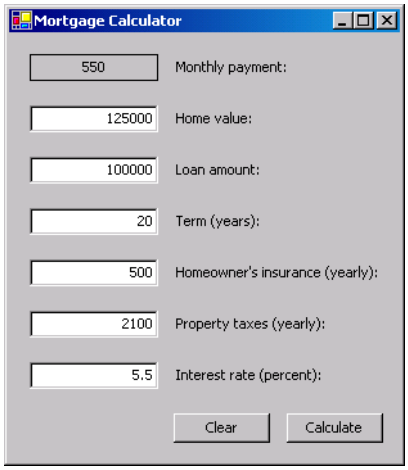


Figure 4.25 **Mortgage Calculator** application without GUI design guidelines applied.

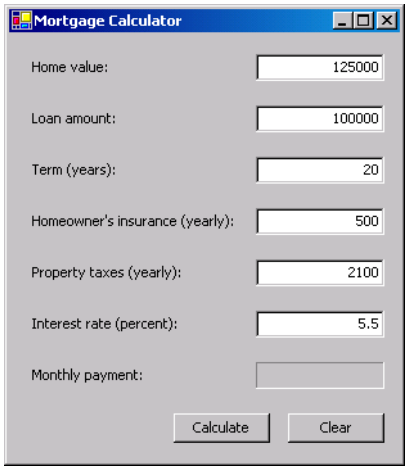
a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial04\Exercises\MortgageCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MortgageCalculator.sln` in the `MortgageCalculator` directory to open the application.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

Answer:



1. `Label` should be placed above or to the left of the control it is describing.
2. Output `Label`'s setting should be `BorderStyle` property `Fixed3D`.
3. Output `Label` initially should be blank.
4. Place an application's output below or to the right of the `Form`'s input control.

4.13 ***(Password GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for a password-protected message application (Fig. 4.26).

Figure 4.26 **Password** application without GUI design guidelines applied.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial04\Exercises\Password` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Password.sln` in the `Password` directory to open the application.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

Answer:

1. Keep the `Label` on the `Buttons` as short and descriptive as possible.
2. Arrange groups of controls approximately 2 grid units apart on a `Form`.
3. Leave approximately 2 grid units of space between the edges of the `Form` and controls nearest the edge. Increase the `Form`'s width.
4. `Buttons` use book-title capitalization.

Programming Challenge ▶ **4.14** ***(Monitor Invoice GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for an invoice application (Fig. 4.27).

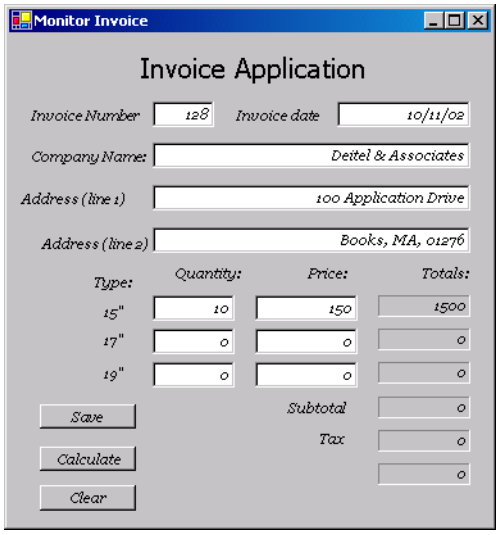


Figure 4.27 **Invoice** application without GUI design guidelines applied.

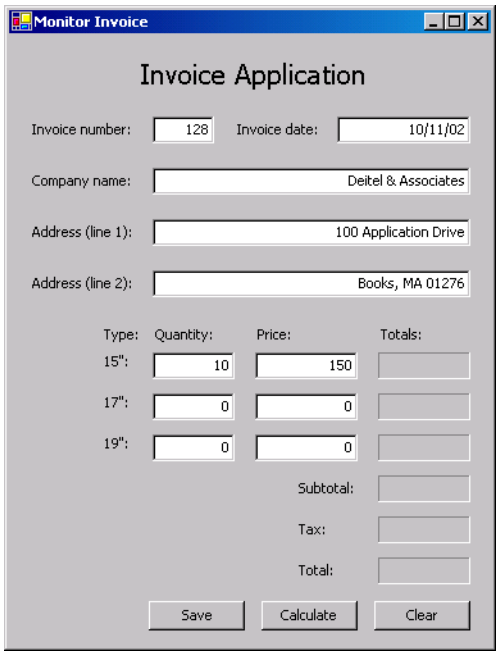
a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial04\Exercises\MonitorInvoice` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MonitorInvoice.sln` in the `MonitorInvoice` directory to open the application.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

Answer:



1. Use `Tahoma` font.
2. `Label`s indicating control usage should end with colon.
3. The `Label` and the control it describes should be aligned on the left if arranged vertically.
4. `Label` should use sentence-style capitalization.

5. `Buttons` should be placed in the top right or bottom right of a `Form`.
6. Each output `Label` must have a label that describes it.
7. Output `Labels` arranged vertically and used to display numbers in a calculation should have the `TextAlign` property set to `MiddleRight`.
8. Descriptive `Labels` that are in the same column vertically should be left aligned.

TUTORIAL 5

Completing the Inventory Application

Introducing Programming Solutions

Instructor's Manual Exercise Solutions Tutorial 5

MULTIPLE-CHOICE QUESTIONS

5.1 A(n) _____ represents a user action, such as clicking a `Button`.

a) statement b) event
c) application d) function

5.2 To switch to code view, select _____.

a) **Code > View** b) **Design > Code**
c) **View > Code** d) **View > File Code**

5.3 Code that performs the functionality of an application _____.

a) normally is provided by the programmer
b) can never be in the form of an event handler
c) always creates a graphical user interface
d) is always generated by the IDE

5.4 Comments _____.

a) help improve program readability
b) are preceded by the single-quote character
c) are ignored by the compiler d) All of the above.

5.5 The _____ allows a statement to continue past one line (when that character is preceded by one or more whitespace characters).

a) single-quote (') character b) hyphen (-) character
c) underscore (_) character d) plus (+) character

5.6 A(n) _____ causes an application to produce erroneous results.

a) logic error b) event
c) assignment statement d) syntax error

5.7 A portion of code that performs a specific task and returns a value is known as a(n) _____.

a) variable b) function
c) operand d) identifier

5.8 Visual Basic .NET keywords are _____.

a) identifiers b) reserved words
c) case sensitive d) properties

5.9 Visual Studio .NET allows you to organize code into _____, which you can expand or collapse to facilitate code editing.

a) statements b) operators
c) regions d) keywords

5.10 An example of a whitespace character is a _____ character.

a) space b) tab
c) newline d) All of the above.

Answers: 5.1) b. 5.2) c. 5.3) a. 5.4) d. 5.5) c. 5.6) a. 5.7) b. 5.8) b. 5.9) c. 5.10) d.

EXERCISES

5.11 ***(Inventory Enhancement)*** Extend the **Inventory** application to include a `TextBox` in which the user can enter the number of shipments received in a week. Assume every ship-

ment has the same number of cartons (each of which has the same number of items). Then modify the code so that the **Inventory** application uses that value in its calculation.

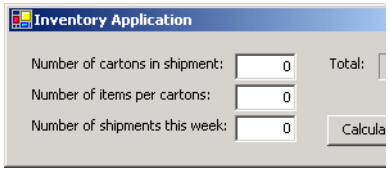


Figure 5.25 Enhanced **Inventory** application GUI.

a) ***Copying the template application to your working directory.*** Copy the `C:\Examples\Tutorial05\Exercises\InventoryEnhancement` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `InventoryEnhancment.sln` in the `InventoryEnhancement` directory to open the application.

c) ***Resizing the Form.*** Resize the `Form` you used in this tutorial by setting the `Size` property to `296, 144`. Move the `Button` toward the bottom of the `Form`, as shown in Fig. 5.25. Its new location should be `184, 78`.

d) ***Adding a `Label`.*** Add a `Label` to the `Form` and change the `Text` property to `Shipments this week:`. Set the `Location` property to `16, 80`. Resize the `Label` so that the entire text displays. Set the `Label`'s `Name` property to `lblShipments`.

e) ***Adding a `TextBox`.*** Add a `TextBox` to the right of the `Label`. Set its `Text` property to `0` and the `Location` property to `128, 80`. Set the `TextAlign` and `Size` properties to the same values as for the other `TextBoxes` in this tutorial's example. Set the `TextBox`'s `Name` property to `txtShipments`.

f) ***Modifying the code.*** Modify the **Calculate Total** `Click` event handler so that it multiplies the number of shipments per week with the product of the number of cartons in a shipment and the number of items in a carton.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter values for the number of cartons per shipment, items per carton and shipments in the current week. Click the **Calculate** `Button` and verify that the total displayed is equal to the result when the three values entered are multiplied together. Enter a few sets of input and verify the total each time.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 5.11 Solution
' Inventory.vb

Public Class FrmInventory
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Click event
   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      ' multiply values input and display result in Label
      lblTotalResult.Text = _
         Val(txtCartons.Text) * _
         Val(txtItems.Text) * _
         Val(txtShipments.Text)

   End Sub ' btnCalculate_Click
```

```
End Class ' FrmInventory
```

5.12 ***(Counter Application)*** Create a counter application. Your counter application will consist of a `Label` and `Button` on the `Form`. The `Label` initially displays `0`, but, each time a user clicks the `Button`, the value in the `Label` is increased by `1`. When incrementing the `Label`, you will need to write a statement such as `lblTotal.Text = Val(lblTotal.Text) + 1`.

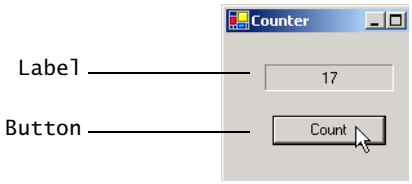


Figure 5.26 Counter GUI.

a) ***Creating the application.*** Create a new project named `Counter`.

b) ***Changing the name of the `Form` file.*** Change the name of `Form1.vb` to `Counter.vb`.

c) ***Modifying a new `Form`.*** Change your `Form`'s `Size` property to `168, 144`. Modify the `Form` so that the title reads **Counter**. Change the name of the `Form` to `FrmCounter`.

d) ***Changing the startup object.*** Change the startup object of your application to the form you modified in *Step c*.

e) ***Adding a `Label`.*** Add a `Label` to the `Form`, and place it as shown in Fig. 5.26. Make sure that the `Label`'s `Text` property is set to `0` and that `TextAlign` property is set so that any text will appear in the middle (both horizontally and vertically) of the `Label`. This can be done by using the `MiddleCenter` `TextAlign` property. Also set the `BorderStyle` property to `Fixed3D`. Set the `Label`'s `Name` property to `lblCountTotal`.

f) ***Adding a `Button`.*** Add a `Button` to the `Form` so that it appears as shown in Fig. 5.26. Set the `Button`'s `Text` property to contain the text **Count**. Set the `Button`'s `Name` property to `btnCount`.

g) ***Creating an event handler.*** Add an event handler to the **Count** `Button` such that the value in the `Label` increases by `1` each time the user clicks the **Count** `Button`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Count** `Button` several times and verify that the output value is incremented each time.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 5.12 Solution
' Counter.vb

Public Class FrmCounter
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Click event
   Private Sub btnCount_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCount.Click

      ' when button is clicked add one to lblCountTotal
      lblCountTotal.Text = Val(lblCountTotal.Text) + 1

   End Sub ' btnCount_Click
```

```
18
19  End Class ' FrmCounter
```

5.13 ***(Account Information Application)*** Create an application that allows a user to input a name, account number and deposit amount. The user then clicks the **Enter** `Button`, which causes the name and account number to be copied and displayed in two output `Labels`. The deposit amount entered will be added to the deposit amount displayed in another output `Label`. The result is displayed in the same output `Label`. Every time the **Enter** `Button` is clicked, the deposit amount entered is added to the deposit amount displayed in the output `Label`, keeping a cumulative total. When updating the `Label`, you will need to write a statement such as `lblDeposits.Text = Val(lblDeposits.Text) + Val(txtDepositAmount)`.

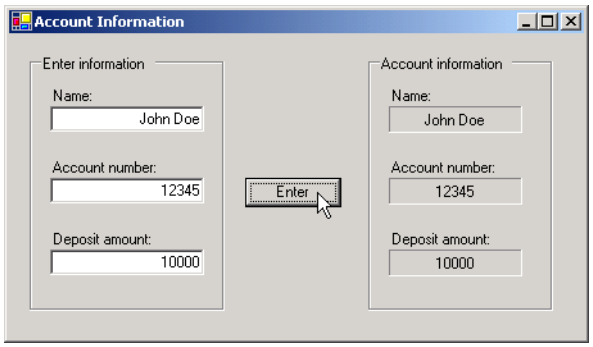


Figure 5.27 Account Information GUI.

a) ***Copying the template application to your working directory.*** Copy `C:\Examples\Tutorial05\Exercises\AccountInformation` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Creating an event handler.*** Add an event handler for the **Enter** `Button`'s `Click` event.

d) ***Coding the event handler.*** Code the event handler to copy information from the **Name:** and **Account number:** `TextBoxes` to their corresponding output `Labels`. Then add the value in the **Deposit amount:** `TextBox` to the **Deposit amount:** output `Label`, and display the result in the **Deposit amount:** output `Label`.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the values in Fig. 5.27 and click the **Enter** `Button`. Verify that the account information is displayed in the `Labels` on the right. Enter varying deposit amounts and click the **Enter** `Button` after each. Verify that the deposit amount on the right has the new values added.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1   ' Exercise 5.13 Solution
2   ' AccountInformation.vb
3
4   Public Class FrmAccountInformation
5      Inherits System.Windows.Forms.Form
6
7      ' Windows Form Designer generated code
8
9      ' handles Click event
10     Private Sub btnEnter_Click(ByVal sender As _
11        System.Object, ByVal e As System.EventArgs) _
12        Handles btnEnter.Click
```

```
13
14          ' copy user input
15          lblCopiedName.Text = txtName.Text
16          lblCopiedAccountNumber.Text = Val(txtAccountNumber.Text)
17          lblBalance.Text = Val(lblBalance.Text) + _
18             Val(txtDepositAmount.Text)
19
20       End Sub ' btnEnter_Click
21
22    End Class ' FrmAccountInformation
```

What does this code do? ▶

5.14 After entering 10 in the `txtPrice` `TextBox` and `1.05` in the `txtTax` `TextBox`, a user clicks the `Button` named `btnEnter`. What is the result of the click, given the following code?

```
1   Private Sub btnEnter_Click(ByVal sender As _
2      System.Object, ByVal e As System.EventArgs) _
3      Handles btnCalculate.Click
4
5      lblOutput.Text = Val(txtPrice.Text) * Val(txtTax.Text)
6
7   End Sub ' btnEnter_Click
```

Answer: This displays the number `10.5` in a `Label`. (This is the amount of the sale including the tax.)

What's wrong with this code? ▶

5.15 The following event handler should execute when the user clicks a **Calculate** `Button`. Identify the error(s) in its code.

```
1   Private Sub btnCalculate_Click(ByVal sender As
2      System.Object, ByVal e As System.EventArgs) _ ' second line
3      Handles btnCalculate.Click
4
5      lblResult.Text = txtPrice.Text * txtTax.Text
6   End Sub ' btnCalculate_Click
```

Answer: The first line of the event handler header is missing the line-continuation character, and the second line of the header has a comment after the line-continuation character; both are syntax errors. Also, the code should use the `Val` function. The corrected code should read:

```
1   Private Sub btnCalculate_Click(ByVal sender As _
2      System.Object, ByVal e As System.EventArgs) _
3      Handles btnCalculate.Click
4
5      lblResult.Text = Val(txtPrice.Text) * Val(txtTax.Text)
6   End Sub ' btnCalculate_Click
```

Using the Debugger ▶

5.16 ***(Account Information Debugging Exercise)*** Copy the folder from `C:\Examples\Tutorial05\Exercises\DebuggingExercise` to your work folder, `C:\SimplyVB`, and run the **Account Information** application. Remove any syntax errors, so that the application runs correctly.

Answer:

Line-continuation character was missing

`lblBalance` was mispelled

```
1  ' Exercise 5.16 Solution
2  ' AccountInformation.vb
3
4  Public Class FrmAccountInformation
5     Inherits System.Windows.Forms.Form
6
7     ' Windows Form Designer generated code
8
9     ' Enter button click event
10    Private Sub btnEnter_Click(ByVal sender As _
11       System.Object, ByVal e As System.EventArgs) _
12       Handles btnEnter.Click
13
14       lblBalance.Text = Val(txtDepositAmount.Text) _
15          - Val(txtWithdrawalAmount.Text) _
16          + Val(lblBalance.Text)
17
18    End Sub ' btnEnter_Click
19
20 End Class ' FrmAccountInformation
```

Programming Challenge ▶

5.17 ***(Account Information Enhancement)*** Modify Exercise 5.13 so that it no longer asks for the user's name and account number, but rather asks the user for a withdrawal or deposit amount. The user can enter both a withdrawal and deposit amount at the same time. When the **Enter** `Button` is clicked, the balance is updated appropriately.

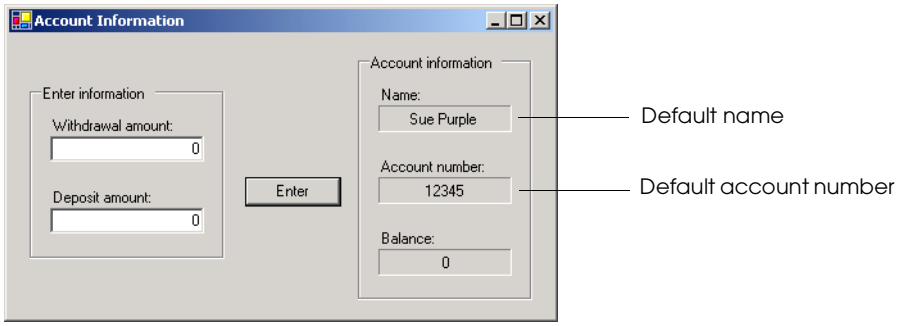


Figure 5.28 Enhanced **Account Information** GUI.

a) ***Copying the template application to your working directory.*** If you have not already done so, copy the `C:\Examples\Tutorial05\Exercises\AccountInformation` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Modifying the GUI.*** Modify the GUI so that it appears as in Fig. 5.28.

d) ***Setting the default values.*** Set the default name and account number to the values shown in Fig. 5.28 using the **Properties** window.

e) ***Writing code to add functionality.*** Update the account balance for every withdrawal (which decreases the balance) and every deposit (which increases the balance). When the balance is updated, reset the `TextBoxes` to zero.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter various withdrawal and deposit amounts, click the **Enter** `Button` after each. Verify that after each time the **Enter** `Button` is clicked, the balance on the right of the application is updated appropriately.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 5.17 Solution
' AccountInformation.vb

Public Class FrmAccountInformation
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' event handler for Enter button
   Private Sub btnEnter_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnEnter.Click

      lblBalance.Text = Val(txtDepositAmount.Text) _
         - Val(txtWithdrawalAmount.Text) _
         + Val(lblBalance.Text)

      ' reset TextBoxes
      txtWithdrawalAmount.Text = "0"
      txtDepositAmount.Text = "0"

   End Sub ' btnEnter_Click

End Class ' FrmAccountInformation
```

TUTORIAL 6

Enhancing the Inventory Application

Introducing Variables, Memory Concepts and Arithmetic

Solutions

Instructor's Manual Exercise Solutions Tutorial 6

MULTIPLE-CHOICE QUESTIONS

6.1 Parentheses that are added to an expression simply to make it easier to read are known as _____ parentheses.

a) necessary b) redundant
c) embedded d) nested

6.2 The _____ operator performs `Integer` division.

a) \ b) +
c) `Mod` d) ^

6.3 Every variable has a _____.

a) name b) value
c) type d) All of the above.

6.4 In Visual Basic .NET, arithmetic expressions must be written in _____ form.

a) straight-line b) top-bottom
c) left-right d) right-left

6.5 Arithmetic expressions are evaluated _____.

a) from right to left b) from left to right
c) according to the rules of operator precedence
d) from the lowest level of precedence to the highest level of precedence

6.6 Variable declarations in event handlers begin with the keyword _____.

a) `Declare` b) `Dim`
c) `Sub` d) `Integer`

6.7 Entering a value in a `TextBox` raises the _____ event.

a) `TextAltered` b) `ValueChanged`
c) `ValueEntered` d) `TextChanged`

6.8 The _____ function converts user input from a `TextBox` to a variable of type `Double`.

a) `Convert` b) `MakeDouble`
c) `Val` d) `WriteDouble`

6.9 Variables to store integer values should be declared with keyword _____.

a) `Integer` b) `Int`
c) `IntVariable` d) None of the above.

6.10 Keyword _____ in a variable declaration indicates that the data type is the next word.

a) `IsA` b) `Type`
c) `Dim` d) `As`

Answers: 6.1) b. 6.2) a. 6.3) d. 6.4) a. 6.5) c. 6.6) b. 6.7) d. 6.8) c. 6.9) a. 6.10) d.

EXERCISES

6.11 ***(Simple Encryption Application)*** This application uses a simple technique to encrypt a number. Encryption is the process of modifying data so that only those intended to receive the data can undo the changes and view the original data. The user enters the number to be encrypted via a `TextBox`. The application then multiplies the number by `7` and adds `5`. The application displays the encrypted number in a `Label` as shown in Fig. 6.24.

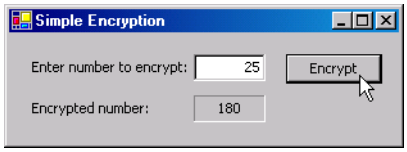


Figure 6.24 Result of completed **Simple Encryption** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial06\Exercises\SimpleEncryption` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SimpleEncryption.sln` in the `SimpleEncryption` directory to open the application.

c) ***Coding the `Click` event handler.*** Encrypt the number in the `Click` event handler by using the preceding technique. The user input should be stored in an `Integer` variable (`intNumber`) before it is encrypted. The event handler then should display the encrypted number.

d) ***Clearing the result.*** Add an event handler for the **Enter number to encrypt:** `TextBox`'s `TextChanged` event. This event handler should clear the **Encrypted number:** `TextBox` whenever the user enters new input.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the value `25` into the **Enter number to encrypt:** `TextBox` and click the **Encrypt** `Button`. Verify that the value `180` is displayed in the **Encrypted number:** output `Label`. Enter other values and click the **Encrypt** `Button` after each. Verify that the appropriate encrypted value is displayed each time.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 6.11 Solution
' SimpleEncryption.vb

Public Class FrmEncryption
   Inherits System.Windows.Forms.Form

   ' handles Click event
   Private Sub btnEncrypt_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEncrypt.Click

      Dim intNumber As Integer

      intNumber = Val(txtInput.Text)  ' obtain user input

      intNumber = intNumber * 7 + 5  ' encrypt number

      lblResult.Text = intNumber  ' display encrypted number
   End Sub ' btnEncrypt_Click

   ' handles TextChanged event
   Private Sub txtInput_TextChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles txtInput.TextChanged

      lblResult.Text = ""
   End Sub ' txtInput_TextChanged

End Class ' FrmEncryption
```

6.12 ***(Temperature Converter Application)*** Write an application that converts a Celsius temperature, *C*, to its equivalent Fahrenheit temperature, *F*. Figure 6.25 displays the completed application. Use the following formula:

$$F = \frac{9}{5}C + 32$$

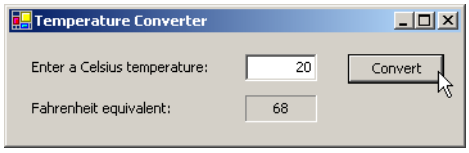


Figure 6.25 Completed **Temperature Converter**.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial06\Exercises\TemperatureConversion` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `TemperatureConversion.sln` in the `TemperatureConversion` directory to open the application.

c) ***Coding the `Click` event handler.*** Perform the conversion in the **Convert** `Button`'s `Click` event handler. Define `Integer` variables to store the user-input Celsius temperature and the result of the conversion. Display the Fahrenheit equivalent of the temperature conversion.

d) ***Clearing user input.*** Clear the result in the **Enter a Celsius temperature:** `TextBox`'s `TextChanged` event.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the value `20` into the **Enter a Celsius temperature:** `TextBox` and click the **Convert** `Button`. Verify that the value `68` is displayed in the output `Label`. Enter other Celsius temperatures, click the **Convert** `Button` after each. Use the formula provided above to verify that the proper Fahrenheit equivalent is displayed each time.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 6.12 Solution
' TemperatureConversion.vb

Public Class FrmTemperatureConverter
   Inherits System.Windows.Forms.Form

   ' handles Click event
   Private Sub btnConvert_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnConvert.Click

      ' temperature variables
      Dim intCelsius As Integer
      Dim intFahrenheit As Integer

      intCelsius = Val(txtInput.Text) ' obtain user input

      ' perform conversion
      intFahrenheit = (9 / 5) * intCelsius + 32

      lblResult.Text = intFahrenheit
   End Sub ' btnConvert_Click

   ' handles TextChanged event
   Private Sub txtInput_TextChanged(ByVal sender As _
```

```
        System.Object, ByVal e As System.EventArgs) _
        Handles txtInput.TextChanged

        lblResult.Text = ""
    End Sub ' txtInput_TextChanged

End Class ' FrmTemperatureConverter
```

6.13 ***(Simple Calculator Application)*** In this exercise, you will add functionality to a simple calculator application. The calculator will allow a user to enter two numbers in the `TextBoxes`. There will be four `Buttons` labeled `+`, `-`, `/` and `*`. When the user clicks the `Button` labeled as addition, subtraction, multiplication or division, the application will perform that operation on the numbers in the `TextBoxes` and displays the result. The calculator also should clear the calculation result when the user enters new input. Figure 6.26 displays the completed calculator.

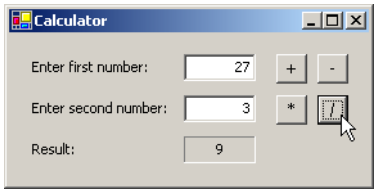


Figure 6.26 Result of **Calculator** application.

a) ***Copying the template to your working directory.*** Copy `C:\Examples\Tutorial06\Exercises\SimpleCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SimpleCalculator.sln` in the `SimpleCalculator` directory to open the application.

c) ***Coding the addition `Click` event handler.*** This event handler should add the two numbers and display the result.

d) ***Coding the subtraction `Click` event handler.*** This event handler should subtract the second number from the first number and display the result.

e) ***Coding the multiplication `Click` event handler.*** This event handler should multiply the two numbers and display the result.

f) ***Coding the division `Click` event handler.*** This event handler should divide the first number by the second number and display the result.

g) ***Clearing the result.*** Write event handlers for the `TextBoxes'` `TextChanged` events. Write code to clear the result `Label` (`lblResult`) after the user enters new input into either `TextBox`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a first number and a second number, then verify that each of the `Buttons` works by clicking each, and viewing the output. Repeat this process with two new values and again verify that the proper output is displayed based on which `Button` is clicked.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 6.13 Solution
' SimpleCalculator.vb

Public Class FrmCalculator
   Inherits System.Windows.Forms.Form

   ' handles addition Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click
```

```
10
11         lblResult.Text = Val(txtFirstNumber.Text) + _
12            Val(txtSecondNumber.Text)
13      End Sub ' btnAdd_Click
14
15      ' handles subtraction Button's Click event
16      Private Sub btnSubtract_Click(ByVal sender As System.Object, _
17         ByVal e As System.EventArgs) Handles btnSubtract.Click
18
19         lblResult.Text = Val(txtFirstNumber.Text) - _
20            Val(txtSecondNumber.Text)
21      End Sub ' btnSubtract_Click
22
23      ' handles multiplication Button's Click event
24      Private Sub btnMultiply_Click(ByVal sender As System.Object, _
25         ByVal e As System.EventArgs) Handles btnMultiply.Click
26
27         lblResult.Text = Val(txtFirstNumber.Text) * _
28            Val(txtSecondNumber.Text)
29      End Sub ' btnMultiply_Click
30
31      ' handles division Button's Click event
32      Private Sub btnDivide_Click(ByVal sender As System.Object, _
33         ByVal e As System.EventArgs) Handles btnDivide.Click
34
35         lblResult.Text = Val(txtFirstNumber.Text) / _
36            Val(txtSecondNumber.Text)
37      End Sub ' btnDivide_Click
38
39      ' handles TextChanged event
40      Private Sub txtFirstNumber_TextChanged(ByVal sender _
41         As System.Object, ByVal e As System.EventArgs) _
42         Handles txtFirstNumber.TextChanged
43
44         lblResult.Text = ""
45      End Sub ' txtFirstNumber_TextChanged
46
47      ' handles TextChanged event
48      Private Sub txtSecondNumber_TextChanged(ByVal sender _
49         As System.Object, ByVal e As System.EventArgs) _
50         Handles txtSecondNumber.TextChanged
51
52         lblResult.Text = ""
53      End Sub ' txtSecondNumber_TextChanged
54
55   End Class ' FrmCalculator
```

What does this code do? ▶

6.14 This code modifies values `intNumber1`, `intNumber2` and `intResult`. What are the final values of these variables?

```
1  Dim intNumber1 As Integer
2  Dim intNumber2 As Integer
3  Dim intResult As Integer
4
5  intNumber1 = 5 * (4 + 6)
6  intNumber2 = 2 ^ 2
7  intResult = intNumber1 \ intNumber2
```

Answer: `intNumber1` gets 50, `intNumber2` gets 4; `intResult` gets 12.

What's wrong with this code? ▶

6.15 Find the error(s) in the following code, which uses variables to perform a calculation.

```
Dim intNumber1 As Integer
Dim intNumber2 As Integer
Dim intResult As Integer

intNumber1 = (4 * 6 ^ 4) / (10 Mod 4 - 2)
intNumber2 = (16 \ 3) ^ 2 * 6 + 1
intResult = intNumber1 - intNumber2
```

Answer: `intNumber1`'s assignment statement contains a division by zero, which will cause a run-time error to occur.

```
Dim intNumber1 As Integer
Dim intNumber2 As Integer
Dim intResult As Integer

' intNumber1 = (4 * 6 ^ 4) / (10 Mod 4 - 2)
intNumber2 = (16 \ 3) ^ 2 * 6 + 1
intResult = intNumber1 - intNumber2
```

Using the Debugger ▶

6.16 ***(Average Three Numbers)*** You have just written an application that takes three numbers as input in `TextBoxes`, stores the three numbers in variables and then finds the average of the numbers (note that the average is rounded to the nearest integer value). The output is displayed in a `Label` (Fig. 6.27, which displays the incorrect output). You soon realize, however, that the number displayed in the `Label` is not the average, but rather a number that does not make sense given the input. Use the debugger to help locate and remove this error.

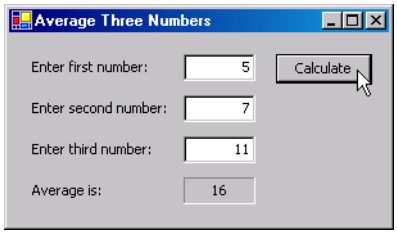


Figure 6.27 **Average Three Numbers** application for Exercise 6.16.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial06\Exercises\AverageDebugging` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AverageDebugging.sln` in the `AverageDebugging` directory to open the application.

c) ***Running the application.*** Select **Debug > Start** to run your application. View the output to observe that the output is incorrect.

d) ***Closing the application.*** Close the application, and view the `Average.vb` file in code view.

e) ***Setting breakpoints.*** Set a breakpoint in the `btnCalculate_Click` event handler. Run the application again, and use the debugger to help find the error(s).

f) ***Finding and correcting the error(s).*** Once you have found the error(s), modify the application so that it correctly calculates the average of three numbers.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter the three values from Fig. 6.27 into the input `TextBoxes` provided and click the **Calculate** `Button`. Verify that the output now accurately reflects the average of these values, which is 8.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 6.16 Solution
' Average.vb

Public Class FrmAverageDebugging
   Inherits System.Windows.Forms.Form

   ' handles Click event
   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      ' variables to store user inputs
      Dim intNumber1 As Integer
      Dim intNumber2 As Integer
      Dim intNumber3 As Integer
      Dim intAverage As Integer

      ' obtain user inputs
      intNumber1 = Val(txtFirstNumber.Text)
      intNumber2 = Val(txtSecondNumber.Text)
      intNumber3 = Val(txtThirdNumber.Text)

      ' average numbers
      intAverage = (intNumber1 + intNumber2 + intNumber3) / 3

      lblResult.Text = intAverage  ' display result
   End Sub ' btnCalculate_Click

End Class ' FrmAverageDebugging
```

Answer: The original code only divided the third number (`intNumber3`) by 3 when in fact the average ought to have been the sum of `intNumber1`, `intNumber2` and `intNumber3` divided by three. To correct the error, we included proper parentheses before `intNumber1` and after `intNumber3`.

Programming Challenge ▶

6.17 ***(Digit Extractor Application)*** Write an application that allows the user to enter a five-digit number into a `TextBox`. The application then separates the number into its individual digits and displays each digit in a `Label`. The application should look and behave similarly to Fig. 6.28. [*Hint:* You can use the `Mod` operator to extract the ones digit from a number. For instance, `12345 Mod 10` is `5`. You can use integer division (\) to "peel off" digits from a number. For instance, `12345 \ 100` is `123`. This allows you to treat the `3` in `12345` as a ones digit. Now you can isolate the `3` by using the `Mod` operator. Apply this technique to the rest of the digits.]

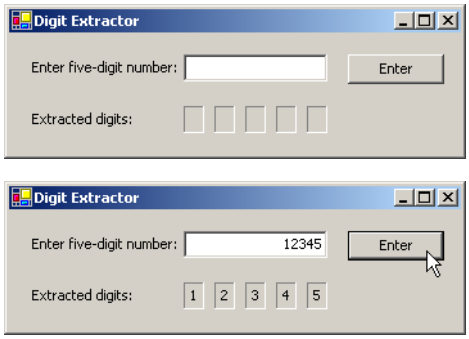


Figure 6.28 **Digit Extractor** application GUI.

a) ***Creating the application.*** Create a new project named `DigitExtractor`. Rename the `Form1.vb` file `DigitExtractor.vb`. Change the name of the `Form` to `FrmDigitExtractor` and set the startup object to `FrmDigitExtractor`. Add `Labels`, a `TextBox` and a `Button` to the application's `Form`. Name the `TextBox` `txtInput` and name the `Button` `btnEnter`. Name the other controls logically based on the tips provided in earlier tutorials.

b) ***Adding an event handler for `btnEnter`'s `Click` event.*** In design view, double click `btnEnter` to create the `btnEnter_Click` event handler. In this event handler, create five variables of type `Integer`. Use the `Mod` operator to extract each digit. Store the digits in the five variables created.

c) ***Adding an event handler for `txtInput`'s `TextChanged` event.*** In design view, double click `txtInput` to create the `txtInput_TextChanged` event handler. In this event handler, clear the five `Labels` used to display each digit. This event handler clears the output whenever new input is entered.

d) ***Running the application.*** Select **Debug > Start** to run your application. Enter a five-digit number and click the **Enter** `Button`. Enter a new five-digit number and verify that the previous output is cleared.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 6.17 Solution
' DigitExtractor.vb

Public Class FrmDigitExtractor
   Inherits System.Windows.Forms.Form

   ' handles Click event
   Private Sub btnEnter_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEnter.Click

      Dim intNumber As Integer  ' five-digit number

      ' five variables for five digits
      Dim intFirst As Integer
      Dim intSecond As Integer
      Dim intThird As Integer
      Dim intFourth As Integer
      Dim intFifth As Integer

      intNumber = Val(txtInput.Text)  ' obtain user input

      ' extract each digit
      intFirst = intNumber \ 10000
      intSecond = intNumber \ 1000 Mod 10
```

```
25         intThird = intNumber \ 100 Mod 10
26         intFourth = intNumber \ 10 Mod 10
27         intFifth = intNumber Mod 10
28
29         ' display extracted digits
30         lblFirstDigit.Text = intFirst
31         lblSecondDigit.Text = intSecond
32         lblThirdDigit.Text = intThird
33         lblFourthDigit.Text = intFourth
34         lblFifthDigit.Text = intFifth
35      End Sub ' btnEnter_Click
36
37      ' handles TextChanged event
38      Private Sub txtInput_TextChanged(ByVal sender As System.Object, _
39         ByVal e As System.EventArgs) Handles txtInput.TextChanged
40
41         ' clear Labels
42         lblFirstDigit.Text = ""
43         lblSecondDigit.Text = ""
44         lblThirdDigit.Text = ""
45         lblFourthDigit.Text = ""
46         lblFifthDigit.Text = ""
47      End Sub ' txtInput_TextChanged
48
49   End Class ' FrmDigitExtractor
```

TUTORIAL 7

Wage Calculator Application

Introducing Algorithms, Pseudocode and Program Control

Solutions

Instructor's Manual Exercise Solutions Tutorial 7

MULTIPLE-CHOICE QUESTIONS

7.1 The _____ operator returns `False` if the left operand is larger than the right operand.

a) =
b) <
c) <=
d) All of the above.

7.2 A _____ occurs when an executed statement does not directly follow the previously executed statement in the written program.

a) transition
b) flow
c) logical error
d) transfer of control

7.3 A variable or an expression that is added to the **Watch** window is known as a _____.

a) watched variable
b) watched expression
c) watch
d) watched value

7.4 The `If…Then` statement is called a _____ statement because it selects or ignores one action.

a) single-selection
b) multiple-selection
c) double-selection
d) repetition

7.5 The three types of control statements are the sequence statement, the selection statement and the _____ statement.

a) repeat
b) looping
c) redo
d) repetition

7.6 In an activity diagram, a rectangle with curved sides represents _____.

a) a complete algorithm
b) a comment
c) an action
d) the termination of the application

7.7 The `If…Then…Else` selection statement ends with the keywords _____.

a) `End If Then Else`
b) `End If Else`
c) `End Else`
d) `End If`

7.8 A variable of data type `Boolean` can be assigned keyword _____ or keyword _____.

a) `True`, `False`
b) `Off`, `On`
c) `True`, `NotTrue`
d) `Yes`, `No`

7.9 A variable whose value cannot be changed after its initial declaration is called a _____.

a) `Double`
b) constant
c) standard
d) `Boolean`

7.10 The _____ operator assigns the result of adding the left and right operands to the left operand.

a) +
b) =+
c) +=
d) + =

Answers: 7.1) d. 7.2) d. 7.3) c. 7.4) a. 7.5) d. 7.6) c. 7.7) d. 7.8) a. 7.9) b. 7.10) c.

EXERCISES

7.11 ***(Currency Converter Application)*** Develop an application that functions as a currency converter as shown in Fig. 7.33. Users must provide a number in the **Dollars:** `TextBox` and a currency name (as text) in the **Convert from Dollars to:** `TextBox`. Clicking the **Convert** `Button` will convert the specified amount into the indicated currency and display it in a

`Label`. Limit yourself to the following currencies as user input: Dollars, Euros, Yen and Pesos. Use the following exchange rates: **1 Dollar = 1.02 Euros, 120 Yen** and **10 Pesos.**

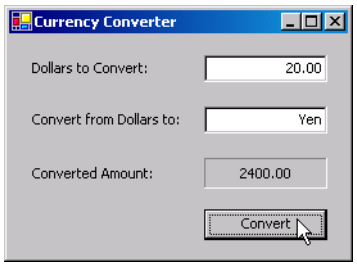


Figure 7.33 Currency Converter GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial07\Exercises\CurrencyConverter` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CurrencyConverter.sln` in the `CurrencyConverter` directory to open the application.

c) ***Add an event handler for the* Convert *`Button`'s `Click` event.*** Double click the **Convert** `Button` to generate an empty event handler for the `Button`'s `Click` event. The code for *Steps d–f* belongs in this event handler.

d) ***Obtaining the user input.*** Use the `Val` function to convert the user input from the **Dollars:** `TextBox` to a `Double`. Assign the `Double` to a `Decimal` variable `decAmount`. Visual Basic .NET implicitly performs this conversion from `Double` to `Decimal`.

e) ***Performing the conversion.*** Use an `If...ElseIf...ElseIf` statement to determine which currency the user entered. Assign the result of the conversion to `decAmount`.

f) ***Displaying the result.*** Display the result using method `String.Format` with format specifier F.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value in dollars to be converted and the name of the currency you wish to convert to. Click the **Convert** `Button` and, using the exchange rates above, verify that the correct output is displays.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 7.11 Solution
' CurrencyConverter.vb

Public Class FrmCurrencyConverter
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Click event
   Private Sub btnConvert_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnConvert.Click

      Dim decAmount As Decimal

      decAmount = Val(txtValue.Text)  ' obtain dollar amount

      ' perform currency conversion
      If txtCurrency.Text = "Euros" Then
         decAmount *= 1.02

      ElseIf txtCurrency.Text = "Yen" Then
```

```
            decAmount *= 120

         ElseIf txtCurrency.Text = "Pesos" Then
            decAmount *= 10
         End If

         lblConvertedResult.Text = String.Format("{0:F}", decAmount)
      End Sub ' btnConvert_Click

   End Class ' FrmCurrencyConverter
```

7.12 ***(Wage Calculator Application that Performs Tax Calculations)*** Develop an application that calculates an employee's wages as shown in Fig. 7.34. The user should provide the hourly wage and number of hours worked per week. When the **Calculate** `Button` is clicked, the gross earnings of the user should display in the **Gross earnings:** `TextBox`. The **Less FWT:** `TextBox` should display the amount deducted for Federal taxes and the **Net earnings:** `TextBox` displays the difference between the gross earnings and the Federal tax amount. Assume overtime wages are `1.5` times the hourly wage and Federal taxes are 15% of gross earnings. The **Clear** `Button` should clear all fields.

Figure 7.34 Wage Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial07\Exercises\ExpandedWageCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `WageCalculator.sln` in the `ExpandedWageCalculator` directory to open the application.

c) ***Modifying the Calculate `Button`'s `Click` event handler.*** Add the code for *Steps d–f* to `btnCalculate_Click`.

d) ***Adding a new variable.*** Declare `decFederalTaxes` to store the amount deducted for Federal taxes.

e) ***Calculating and displaying the Federal taxes deducted.*** Multiply the total earnings (`decEarnings`) by `0.15` (that is, 15%) to determine the amount to be removed for taxes. Assign the result to `decFederalTaxes`. Display this value using method `String.Format` with format specifier `C`.

f) ***Calculating and displaying the employee's net pay.*** Subtract `decFederalTaxes` from `decEarnings` to calculate the employee's net earnings. Display this value using method `String.Format` with format specifier `C`.

g) ***Creating an event handler for the Clear `Button`.*** Double click the **Clear** `Button` to generate an empty event handler for the `Click` event. This event handler should clear user input from the two `TextBoxes` and the results from the three `Labels`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter an hourly wage and the number of hours worked. Click the **Calculate** `Button` and verify that the appropriate output is displayed for gross earnings, amount taken out for federal taxes and the net earnings. Click the **Clear** `Button` and check that all fields are cleared.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 7.12 Solution
' WageCalculator.vb

Public Class FrmWageCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' declare variables
      Dim dblHours As Double
      Dim decWage As Decimal
      Dim decEarnings As Decimal
      Dim decFederalTaxes As Decimal
      Const intHOUR_LIMIT As Integer = 40  ' declare constant

      ' assign values from user input
      dblHours = Val(txtHours.Text)
      decWage = Val(txtWage.Text)

      ' determine wage amount
      If dblHours <= intHOUR_LIMIT Then

         ' if under or equal to 40 hours, regular wages
         decEarnings = dblHours * decWage
      Else

         ' if over 40 hours, regular wages for first 40
         decEarnings = intHOUR_LIMIT * decWage

         ' time and a half for the additional hours
         decEarnings += _
            (dblHours - intHOUR_LIMIT) * (1.5 * decWage)
      End If

      ' assign gross pay to the corresponding Label
      lblEarningsResult.Text = String.Format("{0:C}", decEarnings)

      ' assign federal taxes to the corresponding Label
      decFederalTaxes = decEarnings * 0.15
      lblFWTNumber.Text = String.Format("{0:C}", decFederalTaxes)

      ' assign net pay to the corresponding Label
      lblTotal.Text = String.Format("{0:C}", decEarnings - _
         decFederalTaxes)
   End Sub ' btnCalculate_Click

   ' handles Clear Button's Click event
   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      ' clear each TextBox and output Label
      txtWage.Text = ""
      txtHours.Text = ""
```

```
        lblEarningsResult.Text = ""
        lblFWTNumber.Text = ""
        lblTotal.Text = ""
    End Sub ' btnClear_Click

End Class ' FrmWageCalculator
```

7.13 ***(Customer Charge Account Analyzer Application)*** Develop an application (as shown in Fig. 7.35) that determines whether a department-store customer has exceeded the credit limit on a charge account. Each customer enters an account number (an `Integer`), a balance at the beginning of the month (a `Decimal`), the total of all items charged this month (a `Decimal`), the total of all credits applied to the customer's account this month (a `Decimal`), and the customer's allowed credit limit (a `Decimal`). The application should input each of these facts, calculate the new balance (= *beginning balance – credits + charges*), display the new balance and determine whether the new balance exceeds the customer's credit limit. If the customer's credit limit is exceeded, the application should display a message (in a `Label` at the bottom of the `Form`) informing the customer of this fact.

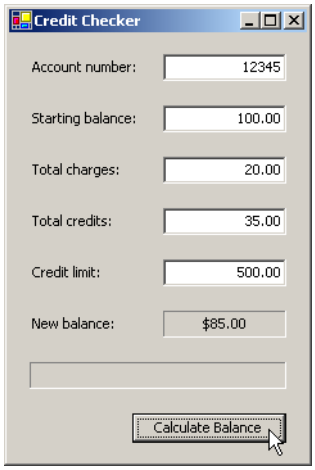


Figure 7.35 Credit Checker GUI.

a) ***Copying the template application to your working directory.*** Copy the `C:\Examples\Tutorial07\Exercises\CreditChecker` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Credit Checker.sln` in the `CreditChecker` directory to open the application.

c) ***Adding the Calculate `Button`'s `Click` event handler.*** Double click the **Calculate Balance** `Button` to generate the empty event handler for the `Click` event. The code for *Steps d–g* is added to this event handler.

d) ***Declaring variables.*** Declare an `Integer` variable to store the account number. Declare four `Decimal` variables to store the starting balance, charges, credits and credit limit. Declare a fifth `Decimal` variable to store the new balance in the account after the credits and charges have been applied.

e) ***Obtaining user input.*** Obtain the user input from the `TextBoxes'` `Text` properties.

f) ***Calculating and displaying the new balance.*** Calculate the new balance by adding the total credits to the starting balance and subtracting the charges. Assign the result to a variable. Display the result formatted as currency.

g) ***Determining if the credit limit has been exceeded.*** If the new balance exceeds the specified credit limit, a message should be displayed in `lblError`.

h) ***Handling the Account number: `TextBox`'s `TextChanged` event.*** Double click the **Account number:** `TextBox` to generate its `TextChanged` event handler. This event handler should clear the other `TextBoxes`, the error message `Label` and the result `Label`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter an account number, your starting balance, the amount charged to your account, the amount credited to your account and your credit limit. Click the **Calculate Balance** `Button` and verify that the new balance displayed is correct. Enter an amount charged that exceeds your credit limit. Click the **Calculate Balance** `Button` and ensure that a message is displayed in the lower `Label`.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 7.13 Solution
' CreditChecker.vb

Public Class FrmCreditChecker
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' declare variables
      Dim intAccountNumber As Integer
      Dim decStartBalance As Decimal
      Dim decTotalCharges As Decimal
      Dim decTotalCredits As Decimal
      Dim decCreditLimit As Decimal
      Dim decNewBalance As Decimal

      intAccountNumber = Val(txtAccountNumber.Text)
      decStartBalance = Val(txtStartBalance.Text)
      decTotalCharges = Val(txtTotalCharges.Text)
      decTotalCredits = Val(txtTotalCredits.Text)
      decCreditLimit = Val(txtCreditLimit.Text)

      ' calculate balance after credits and charges
      decNewBalance = decStartBalance - _
          decTotalCredits + decTotalCharges

      ' display new balance in corresponding Label
      lblNewBalanceNumber.Text = String.Format("{0:C}", _
         decNewBalance)

      ' determine if credit limit has been exceeded
      If decNewBalance > decCreditLimit Then

         ' if credit limit has been exceeded
         ' display an error message
         lblError.Text = "Credit Limit Exceeded!"

      End If

   End Sub ' btnCalculate_Click

   ' handles TextChanged event
   Private Sub txtAccountNumber_TextChanged(ByVal sender _
       As System.Object, ByVal e As System.EventArgs) _
       Handles txtAccounttNumber.TextChanged

```

```
        ' clear all fields when account number is changed
        txtStartBalance.Text = ""
        txtTotalCharges.Text = ""
        txtTotalCredits.Text = ""
        txtCreditLimit.Text = ""
        lblNewBalanceNumber.Text = ""
        lblError.Text = ""

    End Sub ' txtAccountNumber_TextChanged

End Class ' FrmCreditChecker
```

What does this code do? ▶

7.14 Assume that `txtAge` is a `TextBox` control and that the user has entered the value `27` into this `TextBox`. Determine the action performed by the following code:

```
Dim intAge As Integer

intAge = Val(txtAge.Text)

If intAge < 0 Then
   txtAge.Text = "Enter a value greater than or equal to zero."
ElseIf intAge < 13 Then
   txtAge.Text = "Child"
ElseIf intAge < 20 Then
   txtAge.Text = "Teenager"
ElseIf intAge < 30 Then
   txtAge.Text = "Young Adult"
ElseIf intAge < 65 Then
   txtAge.Text = "Adult"
Else
   txtAge.Text = "Senior Citizen"
End If
```

Answer: This code prints text when an age is inputted into the `txtAge` `TextBox`. In this case, the statement `txtAge.Text = "Young Adult"` executes, because the value of `intAge` is below `30`, but not less than `20`.

What's wrong with this code? ▶

7.15 Assume that `lblAMPM` is a `Label` control. Find the error(s) in the following code.

```
Dim intHour As Integer

intHour = 14

If intHour < 11 Then
   If intHour > 0 Then
       lblAMPM.Text = "AM"
   End If
Else
   lblAMPM.Text = "PM"
ElseIf intHour > 23 Then
   lblAMPM.Text = "Time Error."
End If
```

Answer: `ElseIf` cannot appear after an `Else` within the same `If...Then...Else` statement. The correct code should read:

```
Dim intHour As Integer

intHour = 14

If intHour < 11 Then
   If intHour > 0 Then
       lblAMPM.Text = "AM"
   End If
ElseIf intHour > 23 Then
   lblAMPM.Text = "Time Error."
Else
   lblAMPM.Text = "PM"
End If
```

Using the Debugger ▶

7.16 ***(Grade Calculator Application)*** Copy the `C:\Examples\Tutorial07\Debugger` directory into your working directory. This directory contains the `Grades` application, which takes a number from the user and displays the corresponding letter grade. For values 90–100 it should display **A**; for 80–89, **B**, for 70–79, **C**, for 60–69, **D** and for anything lower, an **F**. Run the application. Enter the value `85` in the `TextBox` and click **Calculate**. Notice that the application displays **D** when it ought to display **B**. Select **View > Code** to enter the code editor and set as many breakpoints as you feel necessary. Select **Debug > Start** to use the debugger to help you find the error(s). Figure 7.36 shows the incorrect output when the value `85` is input.

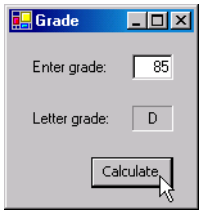


Figure 7.36 Incorrect output for **Grade** application.

Answer:

```
' Exercise 7.16 Solution
' Grades.vb

Public Class FrmGrade
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      Dim intGrade As Integer

      intGrade = Val(txtGrade.Text)

      ' display grade corresponding to number
      If intGrade >= 90 Then
         lblDisplay.Text = "A"
      ElseIf intGrade >= 80 Then
         lblDisplay.Text = "B"
      ElseIf intGrade >= 70 Then
         lblDisplay.Text = "C"
      ElseIf intGrade >= 60 Then
```

Individual `If`…`End If` statements replaced with one `If`…`ElseIf`…`Else` statement

```
            lblDisplay.Text = "D"
         Else
            lblDisplay.Text = "F"
         End If

      End Sub ' btnCalculate_Click

   End Class ' FrmGrade
```

Programming Challenge ▶

7.17 ***(Encryption Application)*** A company transmits data over the telephone, but it is concerned that its phones could be tapped. All its data is transmitted as four-digit `Integers`. The company has asked you to write an application that encrypts its data so that it may be transmitted more securely. Encryption is the process of transforming data for security reasons. Create a `Form` similar to Fig. 7.37. Your program should read four-digits entered by the user and encrypt the information as follows:

a) Replace each digit by *(the sum of that digit plus 7) modulo 10*. We use the term **modulo** to indicate you are to use the modulus (`Mod`) operator.

b) Swap the first digit with the third, and swap the second digit with the fourth.

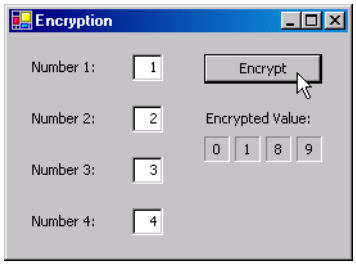


Figure 7.37 Encryption application.

Answer:

```
' Exercise 7.17 Solution
' Encryption.vb

Public Class FrmEncryption
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Click event
   Private Sub btnEncrypt_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEncrypt.Click

      ' clear previous output
      lblEncryptedNumber1.Text = ""
      lblEncryptedNumber2.Text = ""
      lblEncryptedNumber3.Text = ""
      lblEncryptedNumber4.Text = ""

      Dim intNumber1 As Integer
      Dim intNumber2 As Integer
      Dim intNumber3 As Integer
      Dim intNumber4 As Integer

      ' retrieve numbers from TextBoxes
      intNumber1 = Val(txtNumber1.Text)
      intNumber2 = Val(txtNumber2.Text)
```

```
27         intNumber3 = Val(txtNumber3.Text)
28         intNumber4 = Val(txtNumber4.Text)
29
30         ' convert to 1-digit numbers
31         If intNumber1 > 9 Then
32            intNumber1 = intNumber1 Mod 10
33         ElseIf intNumber2 > 9 Then
34            intNumber2 = intNumber2 Mod 10
35         ElseIf intNumber3 > 9 Then
36            intNumber3 = intNumber3 Mod 10
37         ElseIf intNumber4 > 9 Then
38            intNumber4 = intNumber4 Mod 10
39         End If
40
41         ' display using the following:
42         ' 1st number and third number are swapped
43         ' 2nd number and 4th number are swapped
44         lblEncryptedNumber1.Text = (intNumber3 + 7) Mod 10
45         lblEncryptedNumber2.Text = (intNumber4 + 7) Mod 10
46         lblEncryptedNumber3.Text = (intNumber1 + 7) Mod 10
47         lblEncryptedNumber4.Text = (intNumber2 + 7) Mod 10
48
49      End Sub ' btnEncrypt_Click
50
51   End Class ' FrmEncryption
```

TUTORIAL 8

Dental Payment Application

Introducing CheckBoxes and Message Dialogs

Solutions

Instructor's Manual Exercise Solutions Tutorial 8

MULTIPLE-CHOICE QUESTIONS

8.1 How many `CheckBoxes` in a GUI can be selected at once?

a) `0` b) `1`
c) `4` d) any number

8.2 The text that appears alongside a `CheckBox` is referred to as the _____.

a) `CheckBox` label b) `CheckBox` name
c) `CheckBox` value d) `CheckBox` data

8.3 The first argument passed to method `MessageBox.Show` is _____.

a) the text displayed in the dialog's title bar
b) a constant representing the `Buttons` displayed in the dialog
c) the text displayed inside the dialog
d) a constant representing the icon that appears in the dialog

8.4 You can specify the `Button`(s) and icon to be displayed in a message dialog by using the `MessageBoxButtons` and _____ constants.

a) `MessageIcon` b) `MessageBoxImages`
c) `MessageBoxPicture` d) `MessageBoxIcon`

8.5 _____ are used to create complex conditions.

a) Assignment operators b) Activity diagrams
c) Logical operators d) Formatting codes

8.6 Operator `AndAlso` _____.

a) performs short-circuit evaluation b) is not a keyword
c) is a comparison operator
d) evaluates to false if both operands are true

8.7 A `CheckBox` is selected when its `Checked` property is set to _____.

a) `On` b) `True`
c) `Selected` d) `Checked`

8.8 The condition *expression1* `AndAlso` *expression2* evaluates to `True` when _____.

a) *expression1* is `True` and *expression2* is `False`
b) *expression1* is `False` and *expression2* is `True`
c) both *expression1* and *expression2* are `True`
d) both *expression1* and *expression2* are `False`

8.9 The condition *expression1* `OrElse` *expression2* evaluates to `False` when _____.

a) *expression1* is `True` and *expression2* is `False`
b) *expression1* is `False` and *expression2* is `True`
c) both *expression1* and *expression2* are `True`
d) both *expression1* and *expression2* are `False`

8.10 The condition *expression1* `Xor` *expression2* evaluates to `True` when _____.

a) *expression1* is `True` and *expression2* is `False`
b) *expression1* is `False` and *expression2* is `True`
c) both *expression1* and *expression2* are `True`
d) Both a and b.

Answers: 8.1) d. 8.2) a. 8.3) c. 8.4) d. 8.5) c. 8.6) a. 8.7) b. 8.8) c. 8.9) d. 8.10) d.

EXERCISES

8.11 (***Enhanced Dental Payment Application***) Modify the **Dental Payment** application from this tutorial to include additional services, as shown in Fig. 8.21. Add the proper functionality (using `If…Then` structures) to determine whether any of the new `CheckBoxes` are selected and, if so, add the price of the service to the total bill.

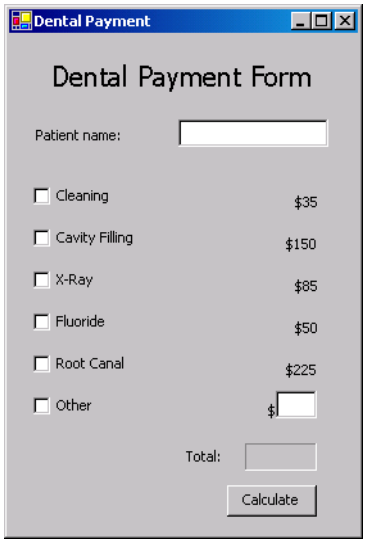


Figure 8.21 Enhanced **Dental Payment** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial08\Exercises\DentalPaymentEnhanced` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `DentalPaymentEnhanced.sln` in the `DentalPaymentEnhanced` directory to open the application.

c) ***Adding `CheckBoxes` and `Labels` and a `TextBox`.*** Add two `CheckBoxes` and two `Labels` to the `Form`. The new `CheckBoxes` should be labelled **Fluoride** and **Root Canal**, respectively. Add these `CheckBoxes` and `Labels` beneath the `X-Ray CheckBox` and its price `Label`. The price for a Fluoride treatment is $50; the price for a root canal is $225. Add a `CheckBox` labelled **Other** and a `Label` containing a dollar sign (**$**) to the `Form`, as shown in Fig. 8.21. Then add a `TextBox` to the right of the **$** `Label` in which the user can enter the cost of the service performed.

d) ***Modifying the `Click` event handler code.*** Add code to the `btnCalculate_Click` event handler that determines whether the new `CheckBoxes` have been selected. This can be done using `If…Then` statements that are similar to the ones already in the event handler. Use the `If…Then` statements to update the bill amount.

e) ***Running the application.*** Select **Debug > Start** to run your application. Test your application by checking one or more of the new services. Click the **Calculate** `Button` and verify that the proper total is displayed. Test the application again by checking some of the services, then checking the Other `CheckBox` and entering a dollar value for this service. Click the **Calculate** `Button` and verify that the proper total is displayed, and that it includes the price for the "other" service.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 8.11 Solution
' DentalPayment.vb

Public Class FrmDentalPayment
   Inherits System.Windows.Forms.Form

```

```
    ' Windows Form Designer generated code

    ' event handler calculates bill
    Private Sub btnCalculate_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnCalculate.Click

       ' if no CheckBox checked, display message
       If (txtName.Text = "") OrElse _
          (chkClean.Checked = False AndAlso _
          chkXRay.Checked = False AndAlso _
          chkCavity.Checked = False AndAlso _
          chkFluoride.Checked = False AndAlso _
          chkRootCanal.Checked = False AndAlso _
          chkOther.Checked = False) Then

          ' display message in dialog
          MessageBox.Show( _
             "Please enter a name and check at least one item", _
             "Missing information", MessageBoxButtons.OK, _
             MessageBoxIcon.Warning)

       Else ' add prices

          ' intTotal contains amount to bill patient
          Dim intTotal As Integer

          ' if patient had a cleaning
          If chkClean.Checked = True Then
             intTotal += 35
          End If

          ' if patient had cavity filled
          If chkCavity.Checked = True Then
             intTotal += 150
          End If

          ' if patient had x-ray taken
          If chkXRay.Checked = True Then
             intTotal += 85
          End If

          ' if patient had Fluoride treatment
          If chkFluoride.Checked = True Then
             intTotal += 50
          End If

          ' if patient had root canal
          If chkRootCanal.Checked = True Then
             intTotal += 225
          End If

          ' if patient had some other service performed
          If chkOther.Checked = True Then
             If txtOtherCost.Text = "" Then
                MessageBox.Show("Please enter cost of service", _
                   "No Cost Entered", MessageBoxButtons.OK, _
                   MessageBoxIcon.Warning)

              Else

             ' add cost entered
```

```
            intTotal += Val(txtOtherCost.Text)
        End If

        ' display total
        lblTotalResult.Text = String.Format("{0:C}", intTotal)

    End If

  End Sub ' btnCalculate_Click

End Class ' FrmDentalPayment
```

8.12 (*Fuzzy Dice Order Form Application*) Write an application that allows users to process orders for fuzzy dice as shown in Fig. 8.22. The application should calculate the total price of the order, including tax and shipping. `TextBoxes` for inputting the order number, the customer name and the shipping address are provided. Initially, these fields contain text that describes their purpose. Provide `CheckBoxes` for selecting the fuzzy-dice color and `TextBoxes` for inputting the quantities of fuzzy dice to order. The application should also contain a `Button` that, when clicked, calculates the subtotals for each type of fuzzy dice ordered and the total of the entire order (including tax and shipping). Use 5% for the tax rate. Shipping charges are $1.50 for up to 20 pairs of dice. If more than 20 pairs of dice are ordered, shipping is free.

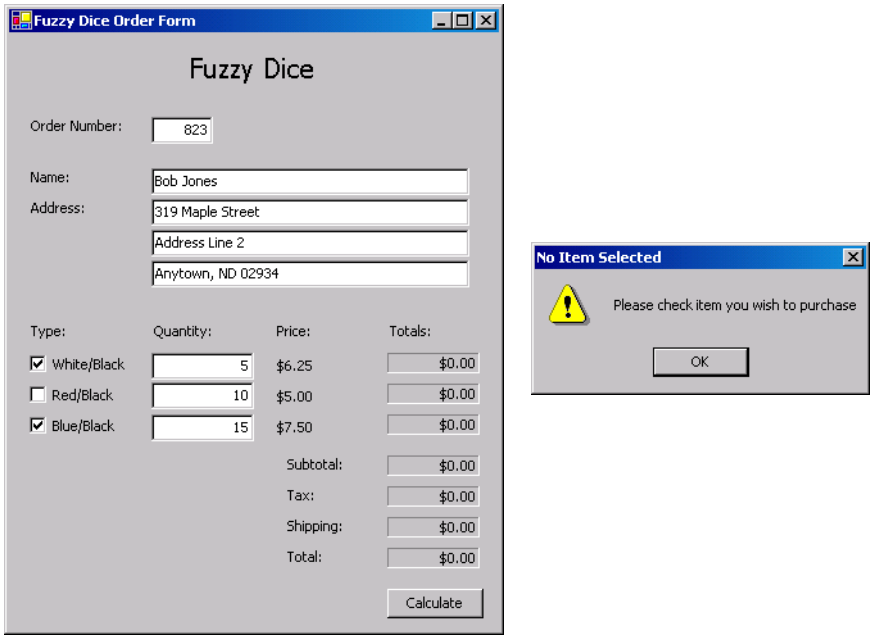


Figure 8.22 Fuzzy Dice Order Form application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial08\Exercises\FuzzyDiceOrderForm` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `FuzzyDiceOrderForm.sln` in the `FuzzyDiceOrderForm` directory to open the application.

c) ***Adding CheckBoxes to the Form.*** Add three `CheckBoxes` to the `Form`. Label the first `CheckBox` **White/Black**, the second one **Red/Black** and the third **Blue/Black**.

d) ***Adding a `Click` event handler and its code.*** Create the `Click` event handler for the **Calculate** `Button`. For this application, users should not be allowed to specify an item's quantity unless the item's corresponding `CheckBox` is checked. For the total to be calculated, the user must enter an order number, a name and a shipping address. Use logical operators to ensure that these terms are met. If they are not, display a message in a dialog.

e) ***Calculating the total cost***. Calculate the subtotal, tax, shipping and total, and display the results in their corresponding `Labels`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Test the application by providing quantities for checked items. For instance, ensure that your application is calculating 5% sales tax. If more than 20 pairs of dice are ordered, verify that shipping is free. Also, determine whether your code containing the logical operators works correctly by specifying a quantity for an item that is not checked. For instance, in Fig. 8.22, a quantity is specified for **Red/Black** dice, but the corresponding `CheckBox` is not selected. This should cause the message dialog in Fig. 8.22 to appear.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 8.12 Solution
' FuzzyDiceOrderForm.vb

Public Class FrmFuzzyDiceOrderForm
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' check validity of order before calculating totals
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' display message if user does not check box
      If (Val(txtWhiteBlackQuantity.Text) > 0 AndAlso _
         chkWhiteBlack.Checked = False) OrElse _
         (Val(txtRedBlackQuantity.Text) > 0 AndAlso _
         chkRedBlack.Checked = False) OrElse _
         (Val(txtBlueBlackQuantity.Text) > 0 AndAlso _
         chkBlueBlack.Checked = False) Then

         ' display message in dialog
         MessageBox.Show( _
            "Please check item you wish to purchase", _
            "No Item Selected", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      ' display message if order number, name or address fields
      ' are empty
      ElseIf txtOrderNumber.Text = "" _
         OrElse txtName.Text = "" _
         OrElse txtAddressLine1.Text = "" _
         OrElse txtCityStateZip.Text = "" Then

         ' display message in dialog
         MessageBox.Show( _
            "Please fill out all information fields", _
            "Empty Fields", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      Else ' calculate totals

         ' individual totals
         ' total of white/black dice ordered
         Dim decWhiteBlackTotals As Decimal = _
            Val(txtWhiteBlackQuantity.Text) * _
            lblWhiteBlackPrice.Text
```

```
47
48          ' total of red/black dice ordered
49          Dim decRedBlackTotals As Decimal = _
50             Val(txtRedBlackQuantity.Text) * _
51             lblRedBlackPrice.Text
52
53          ' total of blue/black dice ordered
54          Dim decBlueBlackTotals As Decimal = _
55             Val(txtBlueBlackQuantity.Text) * _
56             lblBlueBlackPrice.Text
57
58          ' display totals of dice ordered
59          lblWhiteBlackTotals.Text = _
60             String.Format("{0:C}", decWhiteBlackTotals)
61          lblRedBlackTotals.Text = _
62             String.Format("{0:C}", decRedBlackTotals)
63          lblBlueBlackTotals.Text = _
64             String.Format("{0:C}", decBlueBlackTotals)
65
66          ' calculate and display subtotal
67          Dim decSubtotal As Decimal = decWhiteBlackTotals + _
68             decRedBlackTotals + decBlueBlackTotals
69
70          lblSubtotalResult.Text = _
71             String.Format("{0:C}", decSubtotal)
72
73          ' calculate and display tax
74          Dim decTax As Decimal = decSubtotal * 0.05
75
76          lblTaxResult.Text = String.Format("{0:C}", decTax)
77
78          ' shipping
79          ' $1.50 for up to 20 items
80          ' free after 20 items
81          Dim intNumberOfItems As Integer = _
82             (Val(txtWhiteBlackQuantity.Text) + _
83             Val(txtRedBlackQuantity.Text) + _
84             Val(txtBlueBlackQuantity.Text))
85
86          Dim decShippingCost As Decimal = 0.0
87
88          ' shipping is $1.50 if less than 20 pairs ordered
89          If intNumberOfItems <= 20 Then
90
91             decShippingCost = 1.5
92
93          End If
94
95          ' display shipping cost
96          lblShippingResult.Text = _
97             String.Format("{0:C}", decShippingCost)
98
99          ' calculate and display total
100         Dim decTotalCharge As Decimal = decSubtotal + _
101            decTax + decShippingCost
102
103         lblTotalResult.Text = _
104            String.Format("{0:C}", decTotalCharge)
105
106      End If
107
```

```
    End Sub ' btnCalculate_Click

End Class ' FrmFuzzyDiceOrderForm
```

8.13 (***Modified Fuzzy Dice Order Form Application***) Modify the **Fuzzy Dice Order Form** application from Exercise 8.12 to determine whether customers should receive a 7% discount off their purchase. Customers ordering more than $500 (before tax and shipping) in fuzzy dice are eligible for this discount.

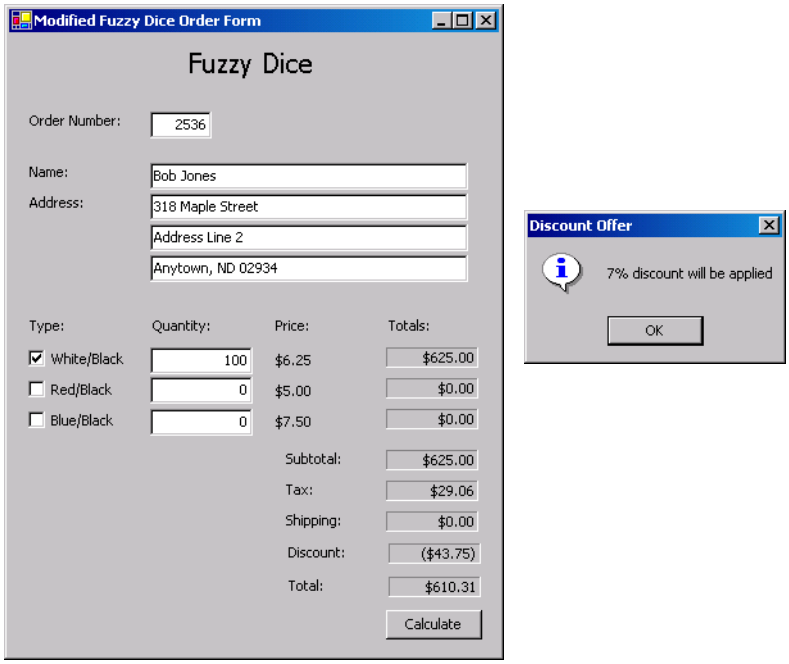


Figure 8.23 Modified **Fuzzy Dice Order Form** application.

a) ***Opening the application***. Open the application you created in Exercise 8.12.

b) ***Determining whether the total cost is over $500***. Use an `If…Then` statement to determine if the amount ordered is greater than $500.

c) ***Displaying the discount and subtracting the discount from the total***. If a customer orders more than $500, display a message dialog as shown in Fig. 8.23 that informs the user that the customer is entitled to a 7% discount. The message dialog should contain an `Information` icon and an **OK** `Button`. Calculate 7% of the total amount, and display the discount amount in the **Discount:** field. Subtract this amount from the total, and update the **Total:** field.

d) ***Running the application.*** Select **Debug > Start** to run your application. Confirm that your application calculates and displays the discount properly.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 8.13 Solution
' FuzzyDiceOrderFormModified.vb

Public Class FrmFuzzyDiceOrderFormModified
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click
```

```
11
12          ' display message if user does not check box
13          If (Val(txtWhiteBlackQuantity.Text) > 0 AndAlso _
14             chkWhiteBlack.Checked = False) OrElse _
15             (Val(txtRedBlackQuantity.Text) > 0 AndAlso _
16             chkRedBlack.Checked = False) OrElse _
17             (Val(txtBlueBlackQuantity.Text) > 0 AndAlso _
18             chkBlueBlack.Checked = False) Then
19
20             ' display message in dialog
21             MessageBox.Show( _
22                "Please check item you wish to purchase", _
23                "No Item Selected", MessageBoxButtons.OK, _
24                MessageBoxIcon.Exclamation)
25
26          ' display message if order number, name or address fields
27          ' are empty
28          ElseIf (txtOrderNumber.Text = "") _
29             OrElse (txtName.Text = "") _
30             OrElse (txtAddressLine1.Text = "") _
31             OrElse (txtCityStateZip.Text = "") Then
32
33             ' display message in dialog
34             MessageBox.Show( _
35                "Please fill out all information fields.", _
36                "Empty fields", MessageBoxButtons.OK, _
37                MessageBoxIcon.Exclamation)
38
39          Else ' calculate totals
40
41             ' individual totals
42
43             ' total of white/black dice ordered
44             Dim decWhiteBlackTotals As Decimal = _
45                Val(txtWhiteBlackQuantity.Text) * _
46                lblWhiteBlackPrice.Text
47
48             ' total of red/black dice ordered
49             Dim decRedBlackTotals As Decimal = _
50                Val(txtRedBlackQuantity.Text) * _
51                lblRedBlackPrice.Text
52
53             ' total of blue/black dice ordered
54             Dim decBlueBlackTotals As Decimal = _
55                Val(txtBlueBlackQuantity.Text) * _
56                lblBlueBlackPrice.Text
57
58             ' display totals for dice
59             lblWhiteBlackTotals.Text = _
60                String.Format("{0:C}", decWhiteBlackTotals)
61             lblRedBlackTotals.Text = _
62                String.Format("{0:C}", decRedBlackTotals)
63             lblBlueBlackTotals.Text = _
64                String.Format("{0:C}", decBlueBlackTotals)
65
66             ' calculate and display subtotal
67             Dim decSubtotal As Decimal = decWhiteBlackTotals + _
68                decRedBlackTotals + decBlueBlackTotals
69
70             lblSubtotalResult.Text = _
71                String.Format("{0:C}", decSubtotal)
```

```

         ' if decTotalCharge is greater than $500
         ' display message box and give 7% discount
         If decSubtotal > 500 Then

            MessageBox.Show( _
               "7% discount will be applied", "Discount Offer", _
               MessageBoxButtons.OK, MessageBoxIcon.Information)

            ' calculate and display new decTotalCharge with discount
            Dim decDiscount As Decimal = decSubtotal * 0.07

            decSubtotal -= decDiscount

            ' decDiscount is negative to reflect that it is
            ' being subtracted from the subtotal during display
            lblDiscount.Text = String.Format("{0:C}", -decDiscount)

         End If

         ' calculate and display tax
         Dim decTax As Decimal = decSubtotal * 0.05

         lblTaxResult.Text = String.Format("{0:C}", decTax)

         ' shipping
         ' $1.50 for up to 20 items
         ' free after 20 items
         Dim intNumberOfItems As Integer = _
            (Val(txtWhiteBlackQuantity.Text) + _
            Val(txtRedBlackQuantity.Text) + _
            Val(txtBlueBlackQuantity.Text))

         Dim decShippingCost As Decimal = 0.0

         ' shipping is $1.50 if less than 20 pairs ordered
         If intNumberOfItems <= 20 Then

            decShippingCost = 1.5

         End If

         ' display shipping charges
         lblShippingResult.Text = _
            String.Format("{0:C}", decShippingCost)

         ' calculate total charge
         Dim decTotalCharge As Decimal = _
            decSubtotal + decTax + decShippingCost

         ' display total charge
         lblTotalResult.Text = _
            String.Format("{0:C}", decTotalCharge)

      End If

   End Sub ' btnCalculate_Click

End Class ' FrmFuzzyDiceOrderFormModified
```

What does this code do?

8.14 Assume that `txtName` is a `TextBox` and that `chkOther` is a `CheckBox` next to which is a `TextBox` `txtOther`, in which the user should specify a value. What does this code segment do?

```
If (txtName.Text = "" OrElse _
   (chkOther.Checked = True AndAlso _
   txtOther.Text = "")) Then

   MessageBox.Show("Please enter a name or value", _
      "Input Error", MessageBoxButtons.OK, _
      MessageBoxIcon.Exclamation)

End If
```

Answer: This code displays a message dialog only if `txtName.Text` is empty or `CheckBox` `chkOther` is selected and its corresponding `TextBox` is left blank.

What's wrong with this code?

8.15 Assume that `txtName` is a `TextBox`. Find the error(s) in the following code:

```
If txtName.Text = "John Doe" Then

   MessageBox.Show("Welcome, John!", _
      MessageBoxIcon.Exclamation)

End If
```

Answer: The call to method `MessageBox.Show` is missing arguments. Also, the nature of the message indicates that `MessageBoxIcon.Information` should be used instead of `MessageBoxIcon.Exclamation`. The corrected code should read:

```
If txtName.Text = "John Doe" Then

   MessageBox.Show("Welcome, John!", _
      "Welcome", MessageBoxButtons.OK, _
      MessageBoxIcon.Information)

End If
```

Using the Debugger

8.16 (*Sibling Survey Application*) The **Sibling Survey** application displays the siblings selected by the user in a dialog. If the user checks either the **Brother(s)** or **Sister(s)** `CheckBox`, and the **No Siblings** `CheckBox`, the user is asked to verify the selection. Otherwise, the user's selection is displayed in a `MessageBox`. While testing this application, you noticed that it does not execute properly. Use the debugger to find and correct the logic error(s) in the code. This exercise is located in the `C:\Examples\Tutorial08\Debugger\SiblingSurvey` directory. Figure 8.24 shows the correct output for the application.

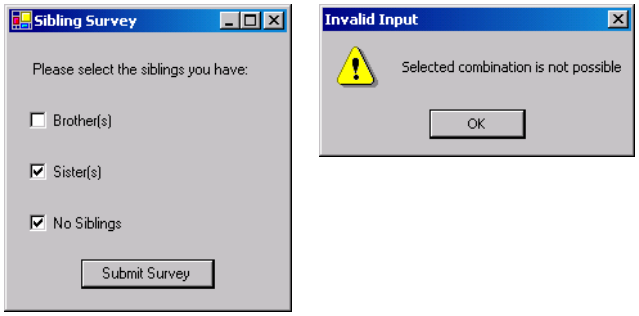


Figure 8.24 Correct output for the **Sibling Survey** application.

Answer:

Replaced `AndAlso` with `OrElse`

Replaced `OrElse` with `AndAlso`

```
' Exercise 8.16 Solution
' SiblingSurvey.vb

Public Class FrmSiblingSurvey
   Inherits System.Windows.Forms.Form

   ' Visual Studio .NET generated code

   ' display what siblings user selects
   Private Sub btnSubmit_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSubmit.Click

      ' check if user selects brothers or sisters
      ' and no siblings
      If (chkNone.Checked = True) AndAlso _
         (chkBrother.Checked = True OrElse _
         chkSister.Checked = True) Then

         MessageBox.Show("Selected combination is not possible", _
            "Invalid Input", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      ' check if user selects CheckBox
      ElseIf chkNone.Checked = False AndAlso _
         chkBrother.Checked = False AndAlso _
         chkSister.Checked = False Then

         MessageBox.Show("Please check at least one CheckBox", _
            "Invalid Input", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      ' check if user has brothers and sisters
      ElseIf chkBrother.Checked = True AndAlso _
         chkSister.Checked = True Then
         MessageBox.Show("You have brothers and sisters", _
            "Siblings", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

      ' check if user has brothers
      ElseIf chkBrother.Checked = True Then
         MessageBox.Show("You have at least one brother", _
            "Siblings", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

      ' check if user has sisters
```

```
      ElseIf chkSister.Checked = True Then
         MessageBox.Show("You have at least one sister", _
            "Siblings", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

      ' user has no siblings
      Else
         MessageBox.Show("You have no siblings", _
            "Siblings", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

      End If

   End Sub ' btnSubmit_Click

End Class ' FrmSiblingSurvey
```

Programming Challenge

8.17 (***Enhanced Fuzzy Dice Order Form Application***) Enhance the **Fuzzy Dice Order Form** application from Exercise 8.12 by replacing the **Calculate** `Button` with a **Clear** `Button`. The application should update the total cost, tax and shipping when the user changes any one of the three **Quantity** field's values (Fig. 8.25). The **Clear** `Button` should return all fields to their original values. [*Hint*: You will need to use the `CheckBox` **CheckedChanged** event for each `CheckBox`. This event is raised when the state of a `CheckBox` changes. Double click a `CheckBox` in design view to create an event handler for that `CheckBox`'s `CheckedChanged` event. You also will need to assign `Boolean` values to the `CheckBoxes'` `Checked` properties to control their states.]

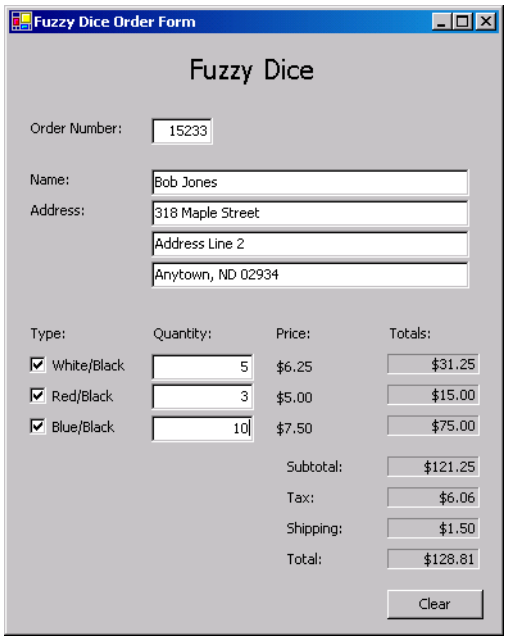


Figure 8.25 Enhanced **Fuzzy Dice Order Form** application.

Answer:

```
' Exercise 8.17 Solution
' FuzzyDiceOrderFormEnhanced.vb

Public Class FrmFuzzyDiceOrderFormEnhanced
   Inherits System.Windows.Forms.Form

```

```
    ' Windows Form Designer generated code

    Private Sub txtWhiteBlackQuantity_TextChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) Handles _
      txtWhiteBlackQuantity.TextChanged

       ' store quantity entered as Integer
       Dim intNumberOfWhiteBlack As Integer = _
          Val(txtWhiteBlackQuantity.Text)

       ' display message if user tries to enter a value
       ' without selecting CheckBox
       If (intNumberOfWhiteBlack <> 0 _
          AndAlso chkWhiteBlack.Checked = False) Then

          ' keep white/black quantity at 0
          txtWhiteBlackQuantity.Text = 0

          ' display message in dialog
          MessageBox.Show( _
             "Please check item you wish to purchase", _
             "No Item Selected", MessageBoxButtons.OK, _
             MessageBoxIcon.Exclamation)

       ' display message if shipping information is not supplied
       ElseIf _
          (txtOrderNumber.Text = "") _
          OrElse (txtName.Text = "") _
          OrElse (txtAddressLine1.Text = "") _
          OrElse (txtCityStateZip.Text = "") Then

          ' display message in dialog
          MessageBox.Show( _
             "Please fill out all information fields.", _
             "Empty Fields", MessageBoxButtons.OK, _
             MessageBoxIcon.Exclamation)

       ' display message if negative number entered
       ElseIf _
          (intNumberOfWhiteBlack < 0) Then
          txtWhiteBlackQuantity.Text = 0
          MessageBox.Show( _
             "Please enter a positive quantity", _
             "Bad Input", MessageBoxButtons.OK, _
             MessageBoxIcon.Exclamation)

       Else ' calculate totals

          ' individual totals
          ' total of white/black dice
          Dim decWhiteBlackTotals As Decimal = _
             Val(txtWhiteBlackQuantity.Text) * lblWhiteBlackPrice.Text

          ' total of red/black dice
          Dim decRedBlackTotals As Decimal = _
             Val(txtRedBlackQuantity.Text) * lblRedBlackPrice.Text

          ' total of blue/black dice
          Dim decBlueBlackTotals As Decimal = _
             Val(txtBlueBlackQuantity.Text) * lblBlueBlackPrice.Text

```

```
         ' display individual totals
         lblWhiteBlackTotals.Text = _
            String.Format("{0:C}", decWhiteBlackTotals)
         lblRedBlackTotals.Text = _
            String.Format("{0:C}", decRedBlackTotals)
         lblBlueBlackTotals.Text = _
            String.Format("{0:C}", decBlueBlackTotals)

         ' subtotal, before tax and shipping
         Dim decSubtotal As Decimal = decWhiteBlackTotals _
            + decRedBlackTotals + decBlueBlackTotals

         lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

         ' calculate and display tax
         Dim decTax As Decimal = decSubtotal * 0.05

         lblTaxResult.Text = String.Format("{0:C}", decTax)

         ' shipping
         ' $1.50 for up to 20 items
         ' free after 20 items
         Dim intNumberOfItems As Integer = _
            (Val(txtWhiteBlackQuantity.Text) + _
            Val(txtRedBlackQuantity.Text) + _
            Val(txtBlueBlackQuantity.Text))

         Dim decShippingCost As Decimal = 0.0

         ' shipping is $1.50 if under 20 items ordered
         If (intNumberOfItems <= 20) AndAlso _
            (intNumberOfItems > 0) Then

            decShippingCost = 1.5

         End If

         ' display shipping cost
         lblShippingResult.Text = _
            String.Format("{0:C}", decShippingCost)

         ' calculate and display total charge
         Dim decTotalCharge As Decimal = decSubtotal + decTax + _
            decShippingCost

         lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

      End If

   End Sub ' txtWhiteBlackQuantity_TextChanged

   Private Sub txtRedBlackQuantity_TextChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) Handles _
      txtRedBlackQuantity.TextChanged

      ' store quantity entered as Integer
      Dim intNumberOfRedBlack As Integer = _
         Val(txtRedBlackQuantity.Text)

      ' check validity of order before calculating totals
      ' and display message for invalid orders
```

```
129
130         ' display message if user tries to enter a value
131         ' without selecting CheckBox
132         If intNumberOfRedBlack <> 0 AndAlso _
133            chkRedBlack.Checked = False Then
134
135            ' keep red/black quantity at 0
136            txtRedBlackQuantity.Text = 0
137
138            ' display message in dialog
139            MessageBox.Show( _
140               "Please check item you wish to purchase", _
141               "No Item Selected", MessageBoxButtons.OK, _
142               MessageBoxIcon.Exclamation)
143
144         ' display message if shipping information is not supplied
145         ElseIf _
146            (txtOrderNumber.Text = "") _
147            OrElse (txtName.Text = "") _
148            OrElse (txtAddressLine1.Text = "") _
149            OrElse (txtCityStateZip.Text = "") Then
150
151            ' display message in dialog
152            MessageBox.Show( _
153               "Please fill out all information fields", _
154               "Empty Fields", MessageBoxButtons.OK, _
155               MessageBoxIcon.Exclamation)
156
157         ' display message if negative number entered
158         ElseIf _
159            (intNumberOfRedBlack < 0) Then
160            txtRedBlackQuantity.Text = 0
161            MessageBox.Show( _
162               "Please enter a positive quantity", _
163               "Bad Input", MessageBoxButtons.OK, _
164               MessageBoxIcon.Exclamation)
165
166         Else ' calculate totals
167
168            ' individual totals
169            ' total of white/black dice
170            Dim decWhiteBlackTotals As Decimal = _
171               Val(txtWhiteBlackQuantity.Text) * lblWhiteBlackPrice.Text
172
173            ' total of red/black dice
174            Dim decRedBlackTotals As Decimal = _
175               Val(txtRedBlackQuantity.Text) * lblRedBlackPrice.Text
176
177            ' total of blue/black dice
178            Dim decBlueBlackTotals As Decimal = _
179               Val(txtBlueBlackQuantity.Text) * lblBlueBlackPrice.Text
180
181            ' display individual totals
182            lblWhiteBlackTotals.Text = _
183               String.Format("{0:C}", decWhiteBlackTotals)
184            lblRedBlackTotals.Text = _
185               String.Format("{0:C}", decRedBlackTotals)
186            lblBlueBlackTotals.Text = _
187               String.Format("{0:C}", decBlueBlackTotals)
188
189            ' subtotal, before tax and shipping
```

```
         Dim decSubtotal As Decimal = decWhiteBlackTotals _
            + decRedBlackTotals + decBlueBlackTotals

         lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

         ' calculate and display tax
         Dim decTax As Decimal = decSubtotal * 0.05

         lblTaxResult.Text = String.Format("{0:C}", decTax)

         ' shipping
         ' $1.50 for up to 20 items
         ' free after 20 items
         Dim intNumberOfItems As Integer = _
            (Val(txtWhiteBlackQuantity.Text) + _
            Val(txtRedBlackQuantity.Text) + _
            Val(txtBlueBlackQuantity.Text))

         Dim decShippingCost As Decimal = 0.0

         ' shipping if $1.50 if under 20 items ordered
         If intNumberOfItems <= 20 AndAlso _
            intNumberOfItems > 0 Then

            decShippingCost = 1.5

         End If

         ' display shipping cost
         lblShippingResult.Text = _
            String.Format("{0:C}", decShippingCost)

         ' calculate and display total charge
         Dim decTotalCharge As Decimal = decSubtotal + decTax + _
            decShippingCost

         lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

      End If

   End Sub ' txtRedBlackQuantity_TextChanged

   Private Sub txtBlueBlackQuantity_TextChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) Handles _
      txtBlueBlackQuantity.TextChanged

      ' store quantity entered as Integer
      Dim intNumberOfBlueBlack As Integer = _
         Val(txtBlueBlackQuantity.Text)

      ' check validity of order before calculating totals
      ' and display message for invalid orders

      ' display message if user tries to enter a value
      ' without selecting CheckBox
      If intNumberOfBlueBlack <> 0 AndAlso _
         chkBlueBlack.Checked = False Then

         ' keep blue/black quantity at 0
         txtBlueBlackQuantity.Text = 0

```

```
         ' display message in dialog
         MessageBox.Show( _
            "Please check item you wish to purchase", _
            "No Item Selected", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      ' display message if shipping information is not supplied
      ElseIf _
         (txtOrderNumber.Text = "") _
         OrElse (txtName.Text = "") _
         OrElse (txtAddressLine1.Text = "") _
         OrElse (txtCityStateZip.Text = "") Then

         ' display message in dialog
         MessageBox.Show( _
            "Please fill out all information fields", _
            "Empty Fields", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      ' display message if negative number is entered
      ElseIf _
         (intNumberOfBlueBlack < 0) Then

         txtBlueBlackQuantity.Text = 0
         MessageBox.Show( _
            "Please enter a positive quantity", _
            "Bad Input", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

      Else ' calculate totals

         ' individual totals
         ' total of white/black dice
         Dim decWhiteBlackTotals As Decimal = _
            Val(txtWhiteBlackQuantity.Text) * lblWhiteBlackPrice.Text

         ' total of red/black dice
         Dim decRedBlackTotals As Decimal = _
            Val(txtRedBlackQuantity.Text) * lblRedBlackPrice.Text

         ' total of blue/black dice
         Dim decBlueBlackTotals As Decimal = _
            Val(txtBlueBlackQuantity.Text) * lblBlueBlackPrice.Text

         ' display individual totals
         lblWhiteBlackTotals.Text = _
            String.Format("{0:C}", decWhiteBlackTotals)
         lblRedBlackTotals.Text = _
            String.Format("{0:C}", decRedBlackTotals)
         lblBlueBlackTotals.Text = _
            String.Format("{0:C}", decBlueBlackTotals)

         ' subtotal, before tax and shipping
         Dim decSubtotal As Decimal = decWhiteBlackTotals _
            + decRedBlackTotals + decBlueBlackTotals

         lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

         ' calculate and display tax
         Dim decTax As Decimal = decSubtotal * 0.05

```

```
         lblTaxResult.Text = String.Format("{0:C}", decTax)

         ' shipping
         ' $1.50 for up to 20 items
         ' free after 20 items
         Dim intNumberOfItems As Integer = _
            Val(txtWhiteBlackQuantity.Text) + _
            Val(txtRedBlackQuantity.Text) + _
            Val(txtBlueBlackQuantity.Text)

         Dim decShippingCost As Decimal = 0.0

         ' shipping is $1.50 if under 20 items ordered
         If intNumberOfItems <= 20 AndAlso _
            intNumberOfItems > 0 Then

            decShippingCost = 1.5

         End If

         ' display shipping cost
         lblShippingResult.Text = _
            String.Format("{0:C}", decShippingCost)

         ' calculate and display total charge
         Dim decTotalCharge As Decimal = decSubtotal + decTax + _
            decShippingCost

         lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

      End If

   End Sub ' txtBlueBlackQuantity_TextChanged

   ' clear all fields
   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      ' set all fields to their original values
      txtOrderNumber.Text = "0"
      txtName.Text = "Enter name here"
      txtAddressLine1.Text = "Address Line 1"
      txtAddressLine2.Text = "Address Line 2"
      txtCityStateZip.Text = "City, State, zip"
      txtWhiteBlackQuantity.Text = "0"
      txtRedBlackQuantity.Text = "0"
      txtBlueBlackQuantity.Text = "0"
      lblWhiteBlackTotals.Text = "$0.00"
      lblRedBlackTotals.Text = "$0.00"
      lblBlueBlackTotals.Text = "$0.00"
      lblSubtotalResult.Text = "$0.00"
      lblTaxResult.Text = "$0.00"
      lblShippingResult.Text = "$0.00"
      lblTotalResult.Text = "$0.00"
      chkWhiteBlack.Checked = False
      chkRedBlack.Checked = False
      chkBlueBlack.Checked = False

   End Sub ' btnClear_Click

   Private Sub chkWhiteBlack_CheckedChanged(ByVal sender As _
```

```
        System.Object, ByVal e As System.EventArgs) Handles _
        chkWhiteBlack.CheckedChanged

        txtWhiteBlackQuantity.Text = "0"
        lblWhiteBlackTotals.Text = "0"

        ' individual totals
        ' total of white/black dice
        Dim decWhiteBlackTotals As Decimal = _
           Val(txtWhiteBlackQuantity.Text) * _
           lblWhiteBlackPrice.Text

        ' total of red/black dice
        Dim decRedBlackTotals As Decimal = _
           Val(txtRedBlackQuantity.Text) * _
           lblRedBlackPrice.Text

        ' total of blue/black dice
        Dim decBlueBlackTotals As Decimal = _
           Val(txtBlueBlackQuantity.Text) * _
           lblBlueBlackPrice.Text

        ' display individual totals
        lblWhiteBlackTotals.Text = _
           String.Format("{0:C}", decWhiteBlackTotals)
        lblRedBlackTotals.Text = _
           String.Format("{0:C}", decRedBlackTotals)
        lblBlueBlackTotals.Text = _
           String.Format("{0:C}", decBlueBlackTotals)

        ' subtotal, before tax and shipping
        Dim decSubtotal As Decimal = decWhiteBlackTotals + _
           decRedBlackTotals + decBlueBlackTotals

        lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

        ' calculate and display tax
        Dim decTax As Decimal = decSubtotal * 0.05

        lblTaxResult.Text = String.Format("{0:C}", decTax)

        ' shipping
        ' $1.50 for up to 20 items
        ' free after 20 items
        Dim intNumberOfItems As Integer = _
           Val(txtWhiteBlackQuantity.Text) + _
           Val(txtRedBlackQuantity.Text) + _
           Val(txtBlueBlackQuantity.Text)

        Dim decShippingCost As Decimal = 0.0

        ' shipping is $1.50 if under 20 items ordered
        If intNumberOfItems <= 20 AndAlso _
           intNumberOfItems > 0 Then

           decShippingCost = 1.5

        End If

        ' display shipping cost
        lblShippingResult.Text = _
```

```
         String.Format("{0:C}", decShippingCost)

      ' calculate and display total charge
      Dim decTotalCharge As Decimal = decSubtotal + decTax + _
         decShippingCost

      lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

   End Sub ' chkWhiteBlack_CheckedChanged

   Private Sub chkRedBlack_CheckedChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) Handles _
      chkRedBlack.CheckedChanged

      txtRedBlackQuantity.Text = "0"
      lblRedBlackTotals.Text = "0"

      ' individual totals
      ' total of white/black dice
      Dim decWhiteBlackTotals As Decimal = _
         Val(txtWhiteBlackQuantity.Text) * _
         lblWhiteBlackPrice.Text

      ' total of red/black dice
      Dim decRedBlackTotals As Decimal = _
         Val(txtRedBlackQuantity.Text) * _
         lblRedBlackPrice.Text

      ' total of blue/black dice
      Dim decBlueBlackTotals As Decimal = _
         Val(txtBlueBlackQuantity.Text) * _
         lblBlueBlackPrice.Text

      ' display individual totals
      lblWhiteBlackTotals.Text = _
         String.Format("{0:C}", decWhiteBlackTotals)
      lblRedBlackTotals.Text = _
         String.Format("{0:C}", decRedBlackTotals)
      lblBlueBlackTotals.Text = _
         String.Format("{0:C}", decBlueBlackTotals)

      ' subtotal, before tax and shipping
      Dim decSubtotal As Decimal = decWhiteBlackTotals + _
         decRedBlackTotals + decBlueBlackTotals

      lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

      ' calculate and display tax
      Dim decTax As Decimal = decSubtotal * 0.05

      lblTaxResult.Text = String.Format("{0:C}", decTax)

      ' shipping
      ' $1.50 for up to 20 items
      ' free after 20 items
      Dim intNumberOfItems As Integer = _
         Val(txtWhiteBlackQuantity.Text) + _
         Val(txtRedBlackQuantity.Text) + _
         Val(txtBlueBlackQuantity.Text)

      Dim decShippingCost As Decimal = 0.0
```

```
        If intNumberOfItems <= 20 AndAlso intNumberOfItems > 0 Then
           decShippingCost = 1.5

        End If

        ' display shipping cost
        lblShippingResult.Text = _
           String.Format("{0:C}", decShippingCost)

        ' calculate and display total charge
        Dim decTotalCharge As Decimal = decSubtotal + decTax + _
           decShippingCost

        lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

     End Sub ' chkRedBlack_CheckedChanged

     Private Sub chkBlueBlack_CheckedChanged(ByVal sender As _
        System.Object, ByVal e As System.EventArgs) Handles _
        chkBlueBlack.CheckedChanged

        txtBlueBlackQuantity.Text = "0"
        lblBlueBlackTotals.Text = "0"

        ' individual totals
        ' total of white/black dice
        Dim decWhiteBlackTotals As Decimal = _
           Val(txtWhiteBlackQuantity.Text) * _
           lblWhiteBlackPrice.Text

        ' total of red/black dice
        Dim decRedBlackTotals As Decimal = _
           Val(txtRedBlackQuantity.Text) * _
           lblRedBlackPrice.Text

        ' total of blue/black dice
        Dim decBlueBlackTotals As Decimal = _
           Val(txtBlueBlackQuantity.Text) * _
           lblBlueBlackPrice.Text

        ' display individual totals
        lblWhiteBlackTotals.Text = _
           String.Format("{0:C}", decWhiteBlackTotals)
        lblRedBlackTotals.Text = _
           String.Format("{0:C}", decRedBlackTotals)
        lblBlueBlackTotals.Text = _
           String.Format("{0:C}", decBlueBlackTotals)

        ' subtotal, before tax and shipping
        Dim decSubtotal As Decimal = decWhiteBlackTotals + _
           decRedBlackTotals + decBlueBlackTotals

        lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

        ' calculate and display tax
        Dim decTax As Decimal = decSubtotal * 0.05

        lblTaxResult.Text = String.Format("{0:C}", decTax)

        ' shipping
```

```
        ' $1.50 for up to 20 items
        ' free after 20 items
        Dim intNumberOfItems As Integer = _
           Val(txtWhiteBlackQuantity.Text) + _
           Val(txtRedBlackQuantity.Text) + _
           Val(txtBlueBlackQuantity.Text)
        Dim decShippingCost As Decimal = 0.0

        ' shipping is $1.50 if under 20 items ordered
        If intNumberOfItems <= 20 AndAlso _
           intNumberOfItems > 0 Then

           decShippingCost = 1.5

        End If

        ' display shipping cost
        lblShippingResult.Text = _
           String.Format("{0:C}", decShippingCost)

        ' calculate and display total charge
        Dim decTotalCharge As Decimal = decSubtotal + decTax + _
           decShippingCost

        lblTotalResult.Text = String.Format("{0:C}", decTotalCharge)

    End Sub ' chkBlueBlack_CheckedChanged

End Class ' FrmFuzzyDiceOrderFormEnhanced
```

TUTORIAL 9

Car Payment Calculator Application

Introducing the `Do While...Loop` and `Do Until...Loop` Repetition Statements

Solutions

Instructor's Manual Exercise Solutions Tutorial 9

MULTIPLE-CHOICE QUESTIONS

9.1 The _____ statement executes until its loop-continuation condition becomes `True`.

a) `Do While...Loop` b) `Do Until...Loop`
c) `Do` d) `Loop`

9.2 The _____ statement executes until its loop-continuation condition becomes `False`.

a) `Do While...Loop` b) `Do Until...Loop`
c) `Do` d) `Do While`

9.3 A(n) _____ loop occurs when a condition in a `Do While...Loop` never becomes `False`.

a) infinite b) undefined
c) nested d) indefinite

9.4 A _____ is a variable that helps control the number of times that a set of statements will execute.

a) repeater b) counter
c) loop d) repetition control statement

9.5 The _____ control allows users to add and view items in a list.

a) `ListItems` b) `SelectBox`
c) `ListBox` d) `ViewBox`

9.6 In a UML activity diagram, a(n) _____ symbol joins two flows of activity into one flow of activity.

a) merge b) combine
c) action state d) decision

9.7 Property _____ returns an object containing all the values in a `ListBox`.

a) `All` b) `List`
c) `ListItemValues` d) `Items`

9.8 Method _____ deletes all the values in a `ListBox`.

a) `Remove` b) `Delete`
c) `Clear` d) `Del`

9.9 `Items`'s method _____ adds an item to a `ListBox`.

a) `Include` b) `Append`
c) `Add` d) `Insert`

9.10 Method _____ calculates monthly payments on a loan based on a fixed interest rate.

a) `MonPmt` b) `Payment`
c) `MonthlyPayment` d) `Pmt`

Answers: 9.1) b. 9.2) a. 9.3) a. 9.4) b. 9.5) c. 9.6) a. 9.7) d. 9.8) c. 9.9) c. 9.10) d.

EXERCISES

9.11 ***(Table of Powers Application)*** Write an application that displays a table of numbers from 1 to an upper limit, along with each number's squared value (for example, the number *n* to the power 2, or *n* ^ 2) and cubed value (the number *n* to the power 3, or *n* ^ 3). The users should specify the upper limit, and the results should be displayed in a `ListBox`, as in Fig. 9.20.

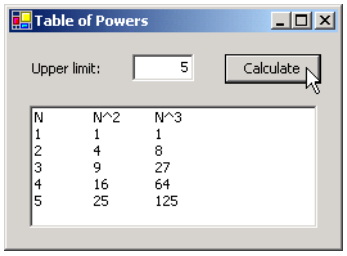


Figure 9.20 Table of Powers application's `Form`.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial09\Exercises\TableOfPowers` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `TableOfPowers.sln` in the `TableOfPowers` directory to open the application.

c) ***Adding a `ListBox`.*** Add a `ListBox` to the application, as shown in Fig. 9.20. Name the `ListBox` `lstResults`.

d) ***Adding the Upper limit: `TextBox` event handler.*** Double click the **Upper limit:** `TextBox` to generate an event handler for this `TextBox`'s `TextChanged` event. In this event handler, clear the `ListBox`.

e) ***Adding the Calculate `Button` event handler.*** Double click the **Calculate** `Button` to generate the empty event handler `btnCalculate_Click`. Add the code specified by the remaining steps to this event handler.

f) ***Clearing the `ListBox`.*** Use method `Clear` on the `Items` property to clear the `ListBox` from any previous data.

g) ***Obtaining the upper limit supplied by the user.*** Assign the value entered by the user in the **Upper limit:** `TextBox` to a variable. Note that the `TextBox`'s `Name` property is set to `txtInput`.

h) ***Adding a header.*** Use method `Add` on the `Items` property to insert a header in the `ListBox`. The header should label three columns—`N`, `N^2` and `N^3`. Column headings should be separated by tab characters.

i) ***Calculating the powers from 1 to the specified upper limit.*** Use a `Do Until...Loop` to calculate the squared value and the cubed value of each number from `1` to the upper limit, inclusive. Add an item to the `ListBox` containing the current number being analyzed, its squared value and its cubed value.

j) ***Incrementing the counter.*** Remember to increment the counter appropriately each time through the loop.

k) ***Running the application.*** Select **Debug > Start** to run your application. Enter an upper limit and click the **Calculate** `Button`. Verify that the table of powers displayed contains the correct values.

l) ***Closing the application.*** Close your running application by clicking its close box.

m) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 9.11 Solution
' TableOfPowers.vb

Public Class FrmTableOfPowers
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

```

```
      Dim intLimit As Integer = 0   ' upper limit set by user
      Dim intCounter As Integer = 1 ' counter begins at 1

      ' clear ListBox
      lstResults.Items.Clear()

      ' retrieve user input
      intLimit = Val(txtInput.Text)

      ' add header
      lstResults.Items.Add("N" & ControlChars.Tab & "N^2" & _
         ControlChars.Tab & "N^3")

      ' calculate and display square and cube of 1 to intLimit
      Do Until intCounter > intLimit

         lstResults.Items.Add(intCounter & ControlChars.Tab & _
            intCounter ^ 2 & ControlChars.Tab & intCounter ^ 3)

         ' increment counter
         intCounter += 1
      Loop

   End Sub ' btnCalculate_Click

   ' handles TextChanged event
   Private Sub txtInput_TextChanged(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles txtInput.TextChanged

      lstResults.Items.Clear()
   End Sub ' txtInput_TextChanged

End Class ' FrmTableOfPowers
```

9.12 ***(Mortgage Calculator Application)*** A bank offers mortgages that can be repaid in 5, 10, 15, 20, 25 or 30 years. Write an application that allows a user to enter the price of a house (the amount of the mortgage) and the annual interest rate. When the user clicks a `Button`, the application displays a table of the mortgage length in years together with the monthly payment, as shown in Fig. 9.21.

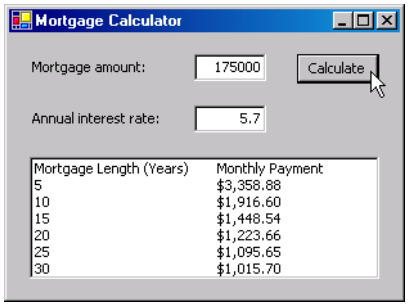


Figure 9.21 Mortgage Calculator application's `Form`.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial09\Exercises\MortgageCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MortgageCalculator.sln` in the `MortgageCalculator` directory to open the application.

c) ***Adding a `ListBox` to display the results.*** Add a `ListBox` as shown in Fig. 9.21. Name the `ListBox` `lstResults`.

d) ***Adding a Calculate `Button` event handler.*** Double click the **Calculate** `Button` to generate the empty event handler `btnCalculate_Click`. Add the code specified in the remaining steps to your event handler.

e) ***Converting the annual interest rate to the monthly interest rate.*** To convert the annual interest rate from a percent value into its `Double` equivalent, divide the annual rate by `100`. Then divide the `Double` annual rate by `12` to obtain the monthly rate.

f) ***Clearing the `ListBox`.*** Use method `Clear` on the `Items` property to clear the `ListBox` from any previous data.

g) ***Displaying a header.*** Use method `Add` to display a header in the `ListBox`. The header should be the column headers "`Mortgage Length (Years)`" and "`Monthly Payment`", separated by a tab character.

h) ***Using a repetition statement.*** Add a `Do While…Loop` repetition statement to calculate six monthly payment options for the user's mortgage. Each option has a different number of years that the mortgage can last. For this exercise, use the following number of years: `5`, `10`, `15`, `20`, `25` and `30`.

i) ***Converting the length of the mortgage from years to months.*** Convert the number of years to months.

j) ***Calculating the monthly payments for six different mortgages.*** Use the `Pmt` function to compute the monthly payments. Pass to the function the monthly interest rate, the number of months in the mortgage and the mortgage amount. Remember that the mortgage amount must be negative, as it represents an amount of money being paid out by the lender.

k) ***Displaying the results.*** Use method `Add` on the `Items` property to display the length of the mortgage in years and the monthly payment in the `ListBox`. You will need to use three tab characters to ensure that the monthly payment appears in the second column.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter a mortgage amount and annual interest rate, then click the **Calculate** `Button`. Verify that the monthly payments displayed contain the correct values.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 9.12 Solution
' MortgageCalculator.vb

Public Class FrmMortgageCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      Dim intMortgageAmount As Integer = 0  ' mortgage amount
      Dim dblAnnualRate As Double = 0       ' annual interest rate
      Dim dblMonthlyRate As Double = 0      ' monthly interest rate
      Dim decPayment As Decimal = 0         ' monthly payment amount
      Dim intYears As Integer = 5           ' years in mortgage
      Dim intMonths As Integer = 0          ' months in mortgage

      ' obtain user input
      intMortgageAmount = Val(txtMortgageAmount.Text)
      dblAnnualRate = Val(txtRate.Text) / 100

      ' calculate monthly interest rate
```

```
        dblMonthlyRate = dblAnnualRate / 12

        ' clear previous results from ListBox
        lstResults.Items.Clear()

        ' add header to ListBox
        lstResults.Items.Add("Mortgage Length (Years)" & _
           ControlChars.Tab & "Monthly Payment")

        ' perform Pmt calculation and display result for
        ' 5, 10, 15, 20, 25 and 30 years
        Do While intYears <= 30

           ' convert years to months for the calculation
           intMonths = intYears * 12

           ' perform calculation
           decPayment = Convert.ToDecimal( _
              Pmt(dblMonthlyRate, intMonths, -intMortgageAmount))

           ' display result
           lstResults.Items.Add(intYears & ControlChars.Tab & _
              ControlChars.Tab & ControlChars.Tab & _
              String.Format("{0:C}", decPayment))

           ' increment counter
           intYears += 5
        Loop

     End Sub ' btnCalculate_Click

  End Class ' FrmMortgageCalculator
```

9.13 ***(Office Supplies Application)*** Create an application that allows a user to make a list of office supplies to buy, as shown in Fig. 9.22. The user should enter the supply in a `TextBox` and click the **Buy** `Button` to add it to the `ListBox`. The **Clear** `Button` removes all the items from the `ListBox`.

Figure 9.22 Office Supplies application's `Form`.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial09\Exercises\OfficeSupplies` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `OfficeSupplies.sln` in `OfficeSupplies` directory to open the application.

c) ***Adding a `ListBox`.*** Add a `ListBox` to the `Form`. Name the `ListBox` `lstSupplies`. Place and size it as shown in Fig. 9.22.

d) ***Adding an event handler for the Buy `Button`.*** Double click the **Buy** `Button` to generate the event handler `btnBuy_Click`. The event handler should obtain the user input

from the `TextBox`. The user input is then added as an item into the `ListBox`. After the input is added to the `ListBox`, clear the **Supply:** `TextBox`.

e) ***Adding an event handler for the Clear `Button`.*** Double click the **Clear** `Button` to generate the event handler `btnClear_Click`. The event handler should use the `Clear` method on the `Items` property to clear the `ListBox`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter several items into the **Supply:** `TextBox` and click the **Buy** `Button` after entering each item. Verify that each item is added to the `ListBox`. Click the **Clear** `Button` and verify that all items are removed from the `ListBox`.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 9.13 Solution
' OfficeSupplies.vb

Public Class FrmOfficeSupplies
   Inherits System.Windows.Forms.Form

   ' handles Buy Button's Click event
   Private Sub btnBuy_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnBuy.Click

      ' add supply item to ListBox, clear input TextBox
      lstSupplies.Items.Add(txtOfficeSupply.Text)
       txtOfficeSupply.Text = ""
    End Sub ' btnBuy_Click

   ' handles Clear Button's Click event
   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      lstSupplies.Items.Clear() ' clear supply items
    End Sub ' btnClear_Click

End Class ' FrmOfficeSupplies
```

What does this code do?

9.14 What is the result of the following code?

```
Dim intX As Integer = 1
Dim intMysteryValue As Integer = 1

Do While intX < 6

   intMysteryValue *= intX
   intX += 1
Loop

lblDisplay.Text = intMysteryValue
```

Answer: `intX` = 6, `intMysteryValue` = 120.

What's wrong with this code?

9.15 Find the error(s) in the following code:

a) Assume that the variable `intX` is declared and initialized to 1. The loop should total the numbers from 1 to 10.

```
Dim intTotal As Integer = 0

Do Until intX <= 10

   intTotal += intX
   intX += 1
Loop
```

Answer: This loop will never execute, as `intX` is already less than or equal to 10. The code should use > instead of <=. An alternative solution would be to convert the loop to a `Do While...Loop`.

```
Dim intTotal As Integer = 0

Do Until intX > 10

   intTotal += intX
   intX += 1
Loop
```

b) Assume that the variable `intCounter` is declared and initialized to 1. The loop should sum the numbers from 1 to 100.

```
Do While intCounter <= 100

   intTotal += intCounter
Loop

intCounter += 1
```

Answer: This is an infinite loop, as `intCounter` will never be greater than 100. The statement that increments `intCounter` must be placed within the `Do While...Loop` statement.

```
Do While intCounter <= 100

   intTotal += intCounter
   intCounter += 1
Loop
```

c) Assume that the variable `intCounter` is declared and initialized to 1000. The loop should iterate from 1000 to 1.

```
Do While intCounter > 0

   lblDisplay.Text = intCounter
   intCounter += 1
Loop
```

Answer: The values must decrease. The value 1 should be subtracted from, rather than added to, `intCounter`.

```
Do While intCounter > 0

   lblDisplay.Text = intCounter
   intCounter -= 1
Loop
```

d) Assume that the variable `intCounter` is declared and initialized to `1`. The loop should execute five times, adding the numbers `1`–`5` to a `ListBox`.

```
Do While intCounter < 5

   lstNumbers.Items.Add(intCounter)
   intCounter += 1
Loop
```

Answer: This loop will execute only four times. To fix the application, the loop-continuation condition should use the <= operator, rather than the < operator.

```
Do While intCounter <= 5

   lstNumbers.Items.Add(intCounter)
   intCounter += 1
Loop
```

Using the Debugger

9.16 (*Odd Numbers Application*) The **Odd Numbers** application should display all of the odd integers between one and the number input by the user. Copy the **Odd Numbers** application from `C:/Examples/Tutorial09/Debugger` to your working directory. Run the application. Notice that, after you enter a value into the **Upper limit:** `TextBox` and click the **View** `Button`, an infinite loop occurs. Use the debugger to find and fix the error(s) in the application. Figure 9.23 displays the correct output for the application.

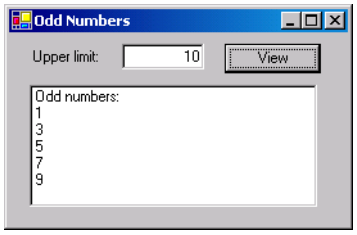


Figure 9.23 Correct output for the **Odd Numbers** application.

```
' Exercise 9.16 Solution
' OddNumbers.vb

Public Class FrmOddNumbers
   Inherits System.Windows.Forms.Form

   ' Windows Forms Designer generated code

   ' display odd numbers from one to number input by user
   Private Sub btnView_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnView.Click

      Dim intLimit As Integer = 0 ' upper limit set by user
      Dim intCounter As Integer = 1 ' counter begins at 1

      lstResults.Items.Clear() ' clear ListBox
      intLimit = Val(txtLimit.Text) ' retrieve upper limit
      lstResults.Items.Add("Odd numbers:") ' display header

      Do While intCounter < intLimit

         ' determine and display odd numbers
         If intCounter Mod 2 <> 0 Then
```

Incorrect code given to students incremented `intLimit` instead of `intCounter`

```
            lstResults.Items.Add(intCounter)
         End If

         intCounter += 1 ' increment counter
      Loop

   End Sub ' btnView_Click

End Class ' FrmOddNumbers
```

Programming Challenge

9.17 ***(To Do List Application)*** Use a `ListBox` as a to do list. Enter each item in a `TextBox`, and add it to the `ListBox` by clicking a `Button`. The item should be displayed in a numbered list as in Fig. 9.24. To do this, we introduce property `Count`, which returns the number of items in a `ListBox`'s `Items` property. The following is a sample call to assign the number of items displayed in the `lstSample` `ListBox` to an `Integer` variable:

```
intCount = lstSample.Items.Count
```

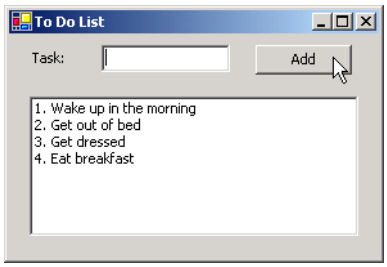


Figure 9.24 **To Do List** application's `Form`.

```
' Exercise 9.17 Solution
' ToDoList.vb

Public Class FrmToDoList
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      Dim intItemNumber As Integer

      ' set number of item
      intItemNumber = lstOutput.Items.Count + 1

      ' add input with number to ListBox
      lstOutput.Items.Add(intItemNumber & ". " & _
         txtInput.Text)

      ' clear TextBox
      txtInput.Text = ""
   End Sub ' btnAdd_Click

End Class ' FrmToDoList
```

TUTORIAL 10

Class Average Application

Introducing the `Do...Loop While` and `Do...Loop Until` Repetition Statements

Solutions

Instructor's Manual Exercise Solutions Tutorial 10

MULTIPLE-CHOICE QUESTIONS

10.1 A(n) _____ occurs when a loop-continuation condition in a `Do...Loop While` never becomes `False`.

a) infinite loop
b) counter-controlled loop
c) control statement
d) nested control statement

10.2 Set property _____ to `True` to enable a `Button`.

a) `Disabled`
b) `Focus`
c) `Enabled`
d) `ButtonEnabled`

10.3 The _____ statement executes at least once and continues executing until its loop-termination condition becomes `True`.

a) `Do While...Loop`
b) `Do...Loop Until`
c) `Do...Loop While`
d) `Do Until...Loop`

10.4 The _____ statement executes at least once and continues executing until its loop-continuation condition becomes `False`.

a) `Do...Loop Until`
b) `Do Until...Loop`
c) `Do While...Loop`
d) `Do...Loop While`

10.5 Method _____ transfers the focus to a control.

a) `GetFocus`
b) `Focus`
c) `Transfer`
d) `Activate`

10.6 A _____ contains the sum of a series of values.

a) total
b) counter
c) condition
d) loop

10.7 Property _____ of _____ contains the number of items in a `ListBox`.

a) `Count`, `ListBox`
b) `ListCount`, `Items`
c) `ListCount`, `ListBox`
d) `Count`, `Items`

10.8 A(n) _____ occurs when a loop executes for one more or one less iteration than is necessary.

a) infinite loop
b) counter-controlled loop
c) off-by-one error
d) nested control statement

10.9 A `Do...Loop Until` repetition statement's loop-termination condition is evaluated _____.

a) only the first time the body executes
b) before the body executes
c) after the body executes
d) None of the above

10.10 If its continuation condition is initially `False`, a `Do...Loop While` repetition statement _____.

a) never executes
b) executes until the condition becomes `True`
c) executes until the condition becomes `True`
d) executes only once

Answers: 10.1) a. 10.2) c. 10.3) b. 10.4) d. 10.5) b. 10.6) a. 10.7) d. 10.8) c. 10.9) c. 10.10) d.

EXERCISES

10.11 ***(Modified Class Average Application)*** Modify the **Class Average** application, as in Fig. 10.18, so that the **Average** `Button` is disabled until 10 grades have been entered.

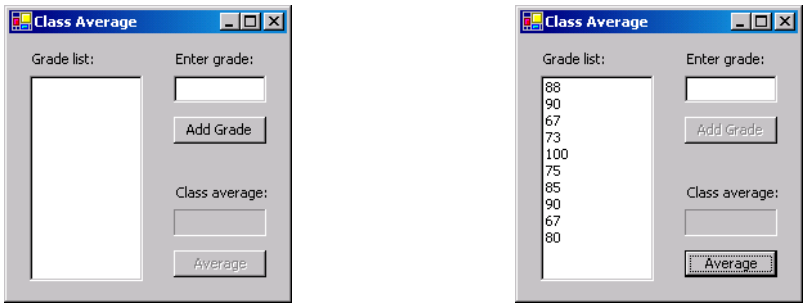


Figure 10.18 Modified **Class Average** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial10\Exercises\ModifiedClassAverage` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ClassAverage.sln` in the `ModifiedClassAverage` directory to open the application.

c) ***Initially disabling the Average `Button`.*** Use the **Properties** window to modify the **Average** `Button` in the `Form` so that it is disabled when the application first executes by initially setting its `Enabled` property to `False`.

d) ***Enabling the Average `Button` after 10 grades have been entered.*** Add code to the `btnAdd_Click` event handler so that the **Average** `Button` becomes enabled when 10 grades have been entered.

e) ***Disabling the Average `Button` after the calculation has been performed.*** Add code to the `btnAverage_Click` event handler so that the **Average** `Button` is disabled once the calculation result has been displayed.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 grades and ensure that the **Average** `Button` is disabled until all 10 grades are entered. Verify that the **Add Grade** `Button` is disabled after 10 grades are entered. Once the **Average** `Button` is enabled, click it and verify that the average displayed is correct. The **Average** `Button` should then become disabled again, and the **Add Grade** `Button` should be enabled.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 10.11 Solution
' ClassAverage.vb

Public Class FrmClassAverage
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Add Grade Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' clear previous grades and calculation result
      If lblOutput.Text <> "" Then
         lblOutput.Text = ""
         lstGrades.Items.Clear()
      End If

```

```
        ' display grade in ListBox
        lstGrades.Items.Add(Val(txtInput.Text))
        txtInput.Clear()  ' clear grade from TextBox
        txtInput.Focus()  ' transfer focus to TextBox

        ' prohibit users from entering more than 10 grades
        If lstGrades.Items.Count >= 10 Then
           btnAdd.Enabled = False  ' disable Add Grade Button
           btnAverage.Enabled = True  ' enable Average Button
           btnAverage.Focus()      ' transfer focus to Average Button
        End If

     End Sub ' btnAdd_Click

     ' handles Average Button's Click event
     Private Sub btnAverage_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnAverage.Click

        ' initialization phase
        Dim intTotal As Integer = 0
        Dim intGradeCounter As Integer = 0
        Dim intGrade As Integer = 0
        Dim dblAverage As Double = 0

        ' sum grades in ListBox
        Do

           ' read grade from ListBox
           intGrade = lstGrades.Items.Item(intGradeCounter)
           intTotal += intGrade    ' add grade to total
           intGradeCounter += 1    ' increment counter
        Loop Until intGradeCounter >= 10

        dblAverage = intTotal / 10   ' calculate average
        lblOutput.Text = String.Format("{0:F}", dblAverage)
        btnAdd.Enabled = True        ' enable Add Grade Button
        btnAverage.Enabled = False   ' disable Average Button
        txtInput.Focus()  ' reset focus to Enter grade: TextBox
     End Sub ' btnAverage_Click

  End Class ' FrmClassAverage
```

10.12 ***(Class Average Application That Handles Any Number of Grades)*** Rewrite the **Class Average** application to handle any number of grades, as in Fig. 10.19. Note that, because the application does not know how many grades the user will enter, the `Buttons` must be enabled at all times.

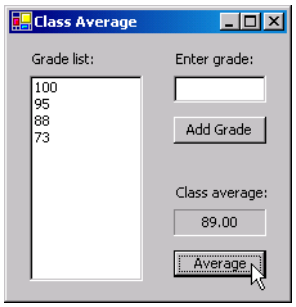


Figure 10.19 Modified **Class Average** application handling an unspecified number of grades.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial10\Exercises\UndeterminedClassAverage` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ClassAverage.sln` in the `UndeterminedClassAverage` directory to open the application.

c) ***Never disabling the Add Grade `Button`.*** Remove code from the `btnAdd_Click` event handler so that the **Add Grade** `Button` is not disabled after entering 10 grades.

d) ***Summing the grades in the `ListBox`.*** Modify code in the `btnAverage_Click` event handler so that `intGradeCounter` is incremented until it is equal to the number of grades entered. Use `lstGrades.Items.Count` to determine the number of items in the `ListBox`. The number returned by the `Count` property will be zero if there are no grades entered. Use an `If…Then` selection statement to avoid division by zero and display a message dialog to the user if there are no grades entered when the user clicks the **Average** `Button`.

e) ***Calculating the class average.*** Modify the code in the `btnAverage_Click` event handler so that `dblAverage` is computed by using `intGradeCounter` rather than the value 10.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 grades and click the Average Button. Verify that the average displayed is correct. Follow the same actions but this time for 15 grades, then for 5 grades. Each time, verify that the appropriate average is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 10.12 Solution
' ClassAverage.vb

Public Class FrmClassAverage
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Add Grade Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' clear previous grades and calculation result
      If lblOutput.Text <> "" Then
         lblOutput.Text = ""
         lstGrades.Items.Clear()
      End If

      ' display grade in ListBox
      lstGrades.Items.Add(Val(txtInput.Text))
      txtInput.Clear()  ' clear grade from TextBox
      txtInput.Focus()  ' transfer focus to TextBox
   End Sub ' btnAdd_Click

   ' handles Average Button's Click event
   Private Sub btnAverage_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAverage.Click

      ' initialization phase
      Dim intTotal As Integer = 0
      Dim intGradeCounter As Integer = 0
      Dim intGrade As Integer = 0
      Dim dblAverage As Double = 0

```

```
      ' no grades entered
      If lstGrades.Items.Count = 0 Then
         MessageBox.Show("Please enter at least one grade", _
            "Enter Grade", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      Else

         ' sum grades in ListBox
         Do

            ' read grade from ListBox
            intGrade = lstGrades.Items.Item(intGradeCounter)
            intTotal += intGrade     ' add grade to total
            intGradeCounter += 1     ' increment counter
         Loop Until intGradeCounter >= lstGrades.Items.Count

         dblAverage = intTotal / intGradeCounter  ' calculate average
         lblOutput.Text = String.Format("{0:F}", dblAverage)
         txtInput.Focus()  ' reset focus to Enter grade: TextBox
      End If
   End Sub ' btnAverage_Click

End Class ' FrmClassAverage
```

10.13 ***(Arithmetic Calculator Application)*** Write an application that allows users to enter a series of numbers and manipulate them. The application should provide users with the option of adding or multiplying the numbers. Users should enter each number in a `TextBox`. After entering each number, users should click a `Button` and the number should be inserted in a `ListBox`. The GUI should behave as in Fig. 10.20.

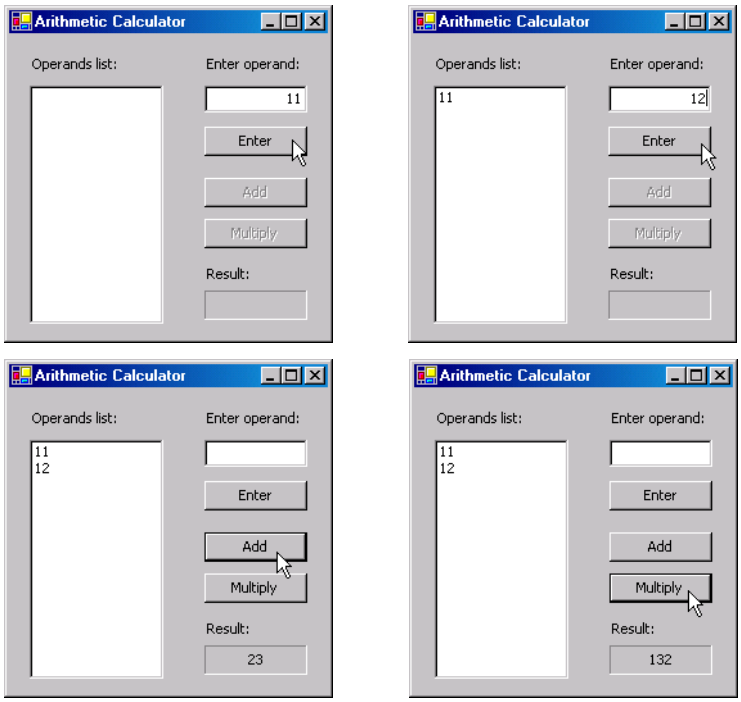


Figure 10.20 Arithmetic Calculator application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial10\Exercises\ArithmeticCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ArithmeticCalculator.sln` in the `ArithmeticCalculator` directory to open the application.

c) ***Add a `ListBox` to display the entered numbers.*** Add a `ListBox`. Place and size it as in Fig. 10.22.

d) ***Creating an event handler for the Enter `Button`.*** Create the `Click` event handler for the **Enter** `Button`. If the result of a previous calculation is displayed, this event handler should clear the result and disable the addition and multiplication `Buttons`. It should then insert the current number in the **Operands list:** `ListBox`. When the `ListBox` contains at least two numbers, the event handler should then enable the addition and multiplication `Buttons`.

e) ***Summing the grades in the `ListBox`.*** Define the `Click` event handler for the **Add** `Button`. This event handler should compute the sum of all of the values in the **Operands list:** `ListBox` and display the result in a `Label` `lblResult`.

f) ***Define the `Click` event handler for the Multiply `Button`.*** This event handler should compute the product of all of the values in the **Operands list:** `ListBox` and display the result in the `Label` `lblResult`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter two values, then click the **Add** and **Multiply** `Buttons`. Verify that the results displayed are correct. Also, make sure that the **Add** and **Multiply** `Buttons` are not enabled until two values have been entered.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 10.13 Solution
' ArithmeticCalculator.vb

Public Class FrmArithmeticCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Enter Button's Click event
   Private Sub btnEnter_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEnter.Click

      ' clear ListBox and lblResult if necessary
      If lblResult.Text <> "" Then
         lblResult.Text = ""
         lstNumbers.Items.Clear()
         btnAdd.Enabled = False ' disable operation Buttons
         btnMultiply.Enabled = False
      End If

      lstNumbers.Items.Add(txtInput.Text) ' add number to ListBox

      ' enable binary operation Buttons when
      ' user has entered two numbers
      If lstNumbers.Items.Count = 2 Then
         btnAdd.Enabled = True
         btnMultiply.Enabled = True
      End If

      txtInput.Clear() ' clear TextBox
      txtInput.Focus() ' set focus to TextBox
   End Sub ' btnEnter_Click

   ' handles addition Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
```

```
        ByVal e As System.EventArgs) Handles btnAdd.Click

        ' initialize total and counter
        Dim dblTotal As Double = 0
        Dim intCounter As Integer = 0

        ' sum numbers in ListBox
        Do
           dblTotal += Val(lstNumbers.Items.Item(intCounter))
           intCounter += 1
        Loop While intCounter < lstNumbers.Items.Count

        lblResult.Text = dblTotal
     End Sub ' btnAdd_Click

     ' handles multiplication Button's Click event
     Private Sub btnMultiply_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnMultiply.Click

        ' initialize dblTotal and intCounter
        Dim dblTotal As Double = 1
        Dim intCounter As Integer = 0

        ' multiply numbers in ListBox
        Do
           dblTotal *= Val(lstNumbers.Items.Item(intCounter))
           intCounter += 1
        Loop While intCounter < lstNumbers.Items.Count

        lblResult.Text = dblTotal
     End Sub ' btnMultiply_Click

  End Class ' FrmArithmeticCalculator
```

What does this code do?

10.14 What is the result of the following code?

```
Dim intY As Integer
Dim intX As Integer
Dim intMysteryValue As Integer

intX = 1
intMysteryValue = 0

Do
   intY = intX ^ 2
   lstDisplay.Items.Add(intY)
   intMysteryValue += 1
   intX += 1
Loop While intX <= 10

lblResult.Text = intMysteryValue
```

Answer: The value displayed in `lblResult` is 11.

What's wrong with this code?

10.15 Find the error(s) in the following code. This code should add 10 to the value in `intY` and store it in `intZ`. It then should reduce the value of `intY` by one and repeat until `intY` is less than 10. The output `Label` `lblResult` should display the final value of `intZ`.

```
1   Dim intY As Integer = 10
2   Dim intZ As Integer = 2
3
4   Do
5      intZ = intY + 10
6   Loop Until intY < 10
7
8   intY -= 1
9
10  lblResult.Text = intZ
```

Answer: This code will loop infinitely because the statement that decrements `intY` (line 8) is not inside the repetition statement. Correct the code as follows:

```
1   Dim intY As Integer = 10
2   Dim intZ As Integer = 2
3
4   Do
5      intZ = intY + 10
6      intY -= 1
7   Loop Until intY < 10
8
9   lblResult.Text = intZ
```

Using the Debugger

10.16 (*Factorial Application*) The **Factorial** application calculates the factorial of an integer input by the user. The factorial of an integer is the product of the integers from one to that number. For example, the factorial of 3 is 6 ($1 \times 2 \times 3$). While testing the application you noticed that it does not execute correctly. Use the debugger to find and fix the logic error(s) in the application. Figure 10.21 displays the correct output for the **Factorial** application.

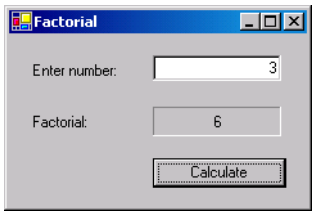


Figure 10.21 Correct output for the **Factorial** application.

Answer:

```
1   ' Exercise 10.16 Solution
2   ' Factorial.vb
3
4   Public Class FrmFactorial
5      Inherits System.Windows.Forms.Form
6
7      ' Windows Form Designer generated code
8
9      ' calculate factorial of user input number
10     Private Sub btnCalculate_Click(ByVal sender As System.Object, _
11        ByVal e As System.EventArgs) Handles btnCalculate.Click
12
13        Dim intInput As Integer ' user input
14        Dim intFactorial As Integer = 1 ' holds factorial
15
```

```
      intInput = Val(txtFactorial.Text) ' get user input

      ' loop until intInput equals zero
      Do
         intFactorial *= intInput ' calculate factorial
         intInput -= 1 ' decrement counter
      Loop While intInput > 1 ' test condition

      lblResult.Text = intFactorial ' display factorial
   End Sub ' btnCalculate_Click

End Class ' FrmFactorial
```

Replaced `Until` with `While`

Programming Challenge

10.17 (***Restaurant Bill Application***) Develop an application that calculates a restaurant bill. The user should be able to enter the item ordered, the quantity of the item ordered and the price per item. When the user clicks the **Add Item** `Button`, your application should display the number ordered, the item ordered and the price per unit in three `ListBoxes` as shown in Fig. 10.22. When the user clicks the **Total Bill** `Button`, the application should calculate the total cost. For each entry in the `ListBox`, multiply the cost of each item by the number of items ordered.

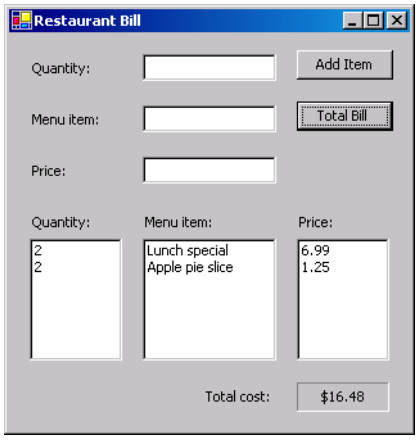


Figure 10.22 **Restaurant Bill** application's `Form`.

Answer:

```
' Exercise 10.17 Solution
' RestaurantBill.vb

Public Class FrmRestaurantBill
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Add Item Button's Click event
   Private Sub btnAddItem_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAddItem.Click

      ' display user input in ListBoxes
      lstQuantity.Items.Add(Val(txtQuantity.Text))
      lstItem.Items.Add(txtItem.Text)
      lstPrice.Items.Add(Val(txtPrice.Text))

      ' clear TextBoxes
      txtItem.Clear()
```

```
20         txtQuantity.Clear()
21         txtPrice.Clear()
22
23         txtQuantity.Focus() ' set the focus to Quantity: TextBox
24      End Sub ' btnAddItem_Click
25
26      ' handles Total Bill Button's Click event
27      Private Sub btnTotal_Click(ByVal sender As System.Object, _
28         ByVal e As System.EventArgs) Handles btnTotal.Click
29
30         Dim intCounter As Integer = 0
31         Dim decCost As Decimal = 0
32
33         ' calculate bill
34         Do
35            decCost += lstPrice.Items.Item(intCounter) * _
36               lstQuantity.Items.Item(intCounter)
37            intCounter += 1
38         Loop While intCounter < lstPrice.Items.Count()
39
40         ' display result
41         lblTotalCost.Text = String.Format("{0:C}", decCost)
42      End Sub ' btnTotal_Click
43
44   End Class ' FrmRestaurantBill
```

TUTORIAL 11

Interest Calculator Application

Introducing the `For...Next` Repetition Statement

Solutions

Instructor's Manual Exercises Solutions Tutorial 11

MULTIPLE-CHOICE QUESTIONS

11.1 "Hello" has data type _____.

a) `String` b) `StringLiteral`
c) `Character` d) `StringText`

11.2 A _____ provides the ability to enter or display multiple lines of text in the same control.

a) `TextBox` b) `NumericUpDown`
c) `MultilineTextBox` d) multiline `NumericUpDown`

11.3 The `NumericUpDown` control allows you to specify _____.

a) a maximum value the user can select b) a minimum value the user can select
c) an increment for the values presented to the user d) All of the above.

11.4 _____ is optional in a `For…Next` header when the control variable's increment is 1.

a) Keyword `To` b) The initial value of the control variable
c) Keyword `Step` d) The final value of the control variable

11.5 Setting `TextBox` property `ScrollBars` to _____ creates a vertical scrollbar.

a) `True` b) `Vertical`
c) `Up` d) `Both`

11.6 _____ is used to determine whether a `For…Next` loop continues to iterate.

a) The initial value of the control variable b) Keyword `For`
c) Keyword `Step` d) The control variable

11.7 In a `For…Next` loop, the control variable is incremented (or decremented) _____.

a) after the body of the loop executes b) when keyword `To` is reached
c) while the loop-continuation condition is `False` d) while the body of the loop executes

11.8 Setting a `NumericUpDown` control's _____ property to `True` ensures that the user cannot enter invalid values in the control.

a) `Increment` b) `ScrollBars`
c) `ReadOnly` d) `InValid`

11.9 The _____ and _____ properties limit the values users can select in the `NumericUpDown` control.

a) `Maximum`, `Minimum` b) `Top`, `Bottom`
c) `High`, `Low` d) `Max`, `Min`

11.10 `For…Next` header _____ can be used to vary the control variable over the odd numbers between 1 and 10.

a) `For intI = 1 To 10 Step 1` b) `For intI = 1 To 10 Step 2`
c) `For intI = 1 To 10 Step -1` d) `For intI = 1 To 10 Step -2`

Answers: 11.1) a. 11.2) a. 11.3) d. 11.4) c. 11.5) b or d. 11.6) d. 11.7) a. 11.8) c. 11.9) a. 11.10) b.

EXERCISES

11.11 ***(Present Value Calculator Application)*** A bank wants to show its customers how much they would need to invest to achieve a specified financial goal (future value) in 5, 10,

15, 20, 25 or 30 years. Users must provide their financial goal (the amount of money desired after the specified number of years has elapsed), an interest rate and the length of the investment in years. Create an application that calculates and displays the principal (initial amount to invest) needed to achieve the user's financial goal. Your application should allow the user to invest money for 5, 10, 15, 20, 25 or 30 years. For example, if a customer wants to reach the financial goal of $15,000 over a period of 5 years when the interest rate is 6.6%, the customer would need to invest $10,896.96 as shown in Fig. 11.16.

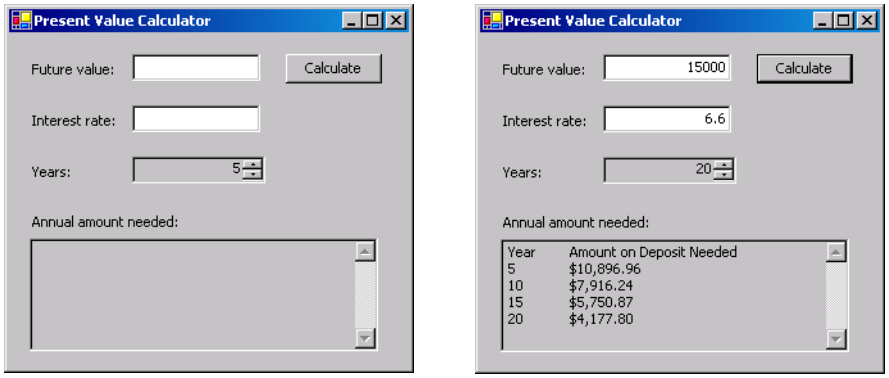


Figure 11.16 Present Value Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial11\Exercises\PresentValue` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `PresentValue.sln` in the `PresentValue` directory to open the application.

c) ***Adding the `NumericUpDown` control.*** Place and size the `NumericUpDown` so that it follows the GUI Design Guidelines. Set the `NumericUpDown` control's `Name` property to `updYear`. Set the `NumericUpDown` control to allow only multiples of five for the number of years. Also, allow the user to select only a duration that is in the specified range of values.

d) ***Adding a multiline `TextBox`.*** Add a `TextBox` to the `Form` below the `NumericUpDown` control. Change the size to `272, 88`, and position the `TextBox` on the `Form` so that it follows the GUI Design Guidelines. Then set that `TextBox` to display multiple lines and a vertical scrollbar. Also ensure that the user cannot modify the text in the `TextBox`.

e) ***Adding a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to the application such that, when the **Calculate** `Button` is clicked, the multiline `TextBox` displays the necessary principal for each five-year interval. Use the following version of the present-value calculation formula:

$$p = a / (1 + r)^{n}$$

where
p is the amount needed to achieve the future value
r is the annual interest rate (for example, .05 is equivalent to 5%)
n is the number of years
a is the future-value amount.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter amounts for the future value, interest rate and number of years. Click the **Calculate** `Button` and verify that the year intervals and the amount on deposit needed for each is correct. Test the application again, this time entering `30` for the number of years. Verify that the vertical scrollbar appears to display all of the output.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 11.11 Solution
' PresentValue.vb

Public Class FrmPresentValue
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' clear txtResult from previous results
      txtResult.Clear()

      ' declare variables
      Dim decFutureValue As Decimal
      Dim dblRate As Double
      Dim intYears As Integer
      Dim decPresentValue As Decimal

      ' retrieve values from user input
      decFutureValue = Val(txtFutureValue.Text)
      dblRate = Val(txtRate.Text)
      intYears = updYear.Text

      ' set initial output line
      txtResult.Text = "Year" & ControlChars.Tab & _
         "Amount on Deposit Needed" & ControlChars.CrLf

      ' calculate principal and display result
      Dim intCounter As Integer

      For intCounter = 5 To intYears Step 5
         decPresentValue = _
            decFutureValue / ((1 + (dblRate / 100)) ^ intCounter)

         ' append result to txtResult's text property
         txtResult.Text &= intCounter & ControlChars.Tab & _
            String.Format("{0:C}", decPresentValue) & _
            ControlChars.CrLf

      Next

   End Sub ' btnCalculate_Click

End Class ' FrmPresentValue
```

11.12 ***(Compound Interest: Comparing Rates Application)*** Write an application that calculates the amount of money in an account after 10 years for interest rate amounts of 5%–10%. For this application, users must provide the initial principal.

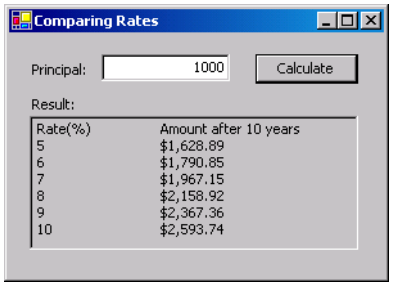


Figure 11.17 Comparing Rates GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial11\Exercises\ComparingRates` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ComparingRates.sln` in the `ComparingRates` directory to open the application.

c) ***Adding a multiline `TextBox`.*** Add a `TextBox` to the `Form` below the **Result:** `Label` control. Change the size to `256, 104`, and position the `TextBox` on the `Form` so that it follows the GUI Design Guidelines (Fig. 11.17). Then set that `TextBox` to display multiple lines. Also ensure that the user cannot modify the text in the `TextBox`.

d) ***Adding a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to the application such that, when the **Calculate** `Button` is clicked, the multiline `TextBox` displays the amount in the account after 10 years for interest rates of 5, 6, 7, 8, 9 and 10 percent. Use the following version of the interest-calculation formula:

$$a = p\ (1 + r)^{n}$$

where
p is the original amount invested (the principal)
r is the annual interest rate (for example, .05 is equivalent to 5%)
n is the number of years
a is the investment's value at the end of the nth year.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the principal amount for an account and click the **Calculate** `Button`. Verify that the correct amounts after 10 years are then displayed, based on interest rate amounts of 5%–10%.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 11.12 Solution
' ComparingRates.vb

Public Class FrmComparingRates
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoke when Calculate Button is pressed
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' declare local variables
      Dim strOutput As String
      Dim intRateCounter As Integer
      Dim decPrincipal As Decimal = Val(txtPrincipal.Text)
      Dim decAmount As Decimal = 0
```

```
18
19         ' set output header
20         strOutput = "Rate(%)" & ControlChars.Tab & ControlChars.Tab _
21            & "Amount after 10 years" & ControlChars.CrLf
22
23         ' calculate amount for each rate and append to string
24         For intRateCounter = 5 To 10
25            decAmount = _
26               decPrincipal * ((1 + intRateCounter / 100) ^ 10)
27
28            strOutput &= (intRateCounter & ControlChars.Tab & _
29               ControlChars.Tab & String.Format("{0:C}", decAmount) _
30               & ControlChars.CrLf)
31         Next
32
33         txtResult.Text = strOutput ' display result
34      End Sub ' btnCalculate_Click
35
36   End Class ' FrmComparingRates
```

11.13 ***(Validating Input to the Interest Calculator Application)*** Enhance the **Interest Calculator** application with error checking. Test for whether the user has entered valid values for the principal and interest rate. If the user enters an invalid value, display a message in the multiline `TextBox`. Figure 11.18 demonstrates the application handling an invalid input.

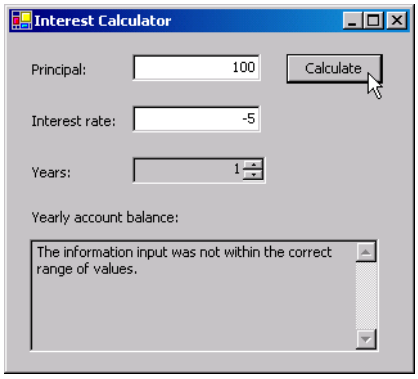


Figure 11.18 **Interest Calculator** application with error checking.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial11\Exercises\InterestCalculatorEnhancement` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `InterestCalculator.sln` in the `InterestCalculatorEnhancement` directory to open the application.

c) ***Adding a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, modify the code to validate the input. The principal should be a positive amount greater than `0`. Also, the interest rate should be greater than `0`, but less than `100`.

d) ***Displaying the error message.*** Display the text "`The information was not within the correct range of values.`" in `txtResult` if the values are not valid.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data for the principal and interest rate. The invalid data can include negative numbers and letters. Verify that entering invalid data and clicking the **Calculate** `Button` results in the error message displayed in Fig. 11.18.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 11.13 Solution
' InterestCalculator.vb

Public Class FrmInterestCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' declare variables to store user input
      Dim decPrincipal As Decimal
      Dim dblRate As Double
      Dim intYear As Integer
      Dim decAmount As Decimal ' store each calculation
      Dim strOutput As String ' store output

      ' retrieve user input
      decPrincipal = Val(txtPrincipal.Text)
      dblRate = Val(txtRate.Text)
      intYear = updYear.Text

      If decPrincipal > 0 AndAlso dblRate > 0 _
         AndAlso dblRate < 100 Then

         ' set output header
         strOutput = "Year" & ControlChars.Tab _
            & "Amount on Deposit" & ControlChars.CrLf

         ' calculate amount after each year and append to string
         For intYear = 1 To intYear

            decAmount = decPrincipal * _
               (1 + dblRate / 100) ^ intYear
            strOutput &= (intYear & ControlChars.Tab & _
               String.Format("{0:C}", decAmount) & _
               ControlChars.CrLf)

         Next

      Else
         strOutput = "The information input was not within the" _
             & " correct range of values."
      End If

      txtResult.Text = strOutput ' display result
   End Sub ' btnCalculate_Click

End Class ' FrmInterestCalculator
```

What does this code do?

11.14 What is the value of `intResult` after the following code executes? Assume that `intPower`, `intI`, `intResult` and `intNumber` are all declared as `Integer`s.

```
intPower = 5
intNumber = 10
intResult = intNumber
```

```

For intI = 1 To (intPower - 1)
   intResult *= intNumber
Next
```

Answer: This code segment raises `intNumber` to the `intPower` power. In this case, `intResult` gets 10^5 (100000).

What's wrong with this code?

11.15 Assume that the variable `intCounter` is declared as an `Integer` for both a and b. Identify and correct the error(s) in each of the following:

a) This statement should display in a `ListBox` all numbers from 100 to 1 in decreasing order.

```
For intCounter = 100 To 1
   lstDisplay.Items.Add(intCounter)
Next
```

Answer: The code needs `Step -1` at the end of the `For...Next` header.

```
For intCounter = 100 To 1 Step -1
   lstDisplay.Items.Add(intCounter)
Next
```

b) The following code should display in a `ListBox` the odd `Integers` from `19` to `1` in decreasing order.

```
For intCounter = 19 To 1 By -1
   lstDisplay.Add(intCounter)
Next
```

Answer: `By -1` should be `Step -2` and `lstDisplay.Add(intCounter)` should be `lstDisplay.Items.Add(intCounter)`.

```
For intCounter = 19 To 1 Step -2
   lstDisplay.Items.Add(intCounter)
Next
```

Using the Debugger

11.16 (*Savings Calculator Application*) The **Savings Calculator** application calculates the amount that the user will have on deposit after one year. The application gets the initial amount on deposit from the user, and assumes that the user will add $100 dollars to the account every month for the entire year. No interest is added to the account. While testing the application, you noticed that the amount calculated by the application was incorrect. Use the debugger to locate and correct any logic error(s). Figure 11.19 displays the correct output for this application.

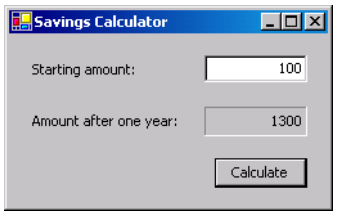


Figure 11.19 Correct output for the **Savings Calculator** application.

Answer:

```
' Exercise 11.16 Solution
' Savings.vb

Public Class FrmSavings
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' calculate amount in account after one year
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      Dim intTotal As Integer = 0 ' amount on deposit
      Dim intCounter As Integer = 1 ' counter starts at 1

      intTotal = Val(txtStartAmount.Text) ' get amount on deposit

      ' add $100 a month for one year
      For intCounter = 1 To 12
         intTotal += 100 ' add money
      Next

      lblTotal.Text = intTotal ' display total after 12 months
   End Sub ' btnCalculate_Click

End Class ' FrmSavings
```

Incorrect code given to students looped 13 times (0-12), instead of 12 (1-12)

Programming Challenge

11.17 ***(Pay Raise Calculator Application)*** Develop an application that computes the amount of money an employee makes each year over a user-specified number of years. The employee receives an hourly wage and a pay raise once every year. The user specifies the hourly wage and the amount of the raise (in percentages per year) in the application.

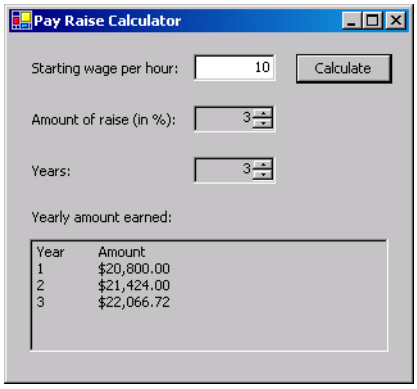


Figure 11.20 Pay Raise application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial11\Exercises\PayRaise` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `PayRaise.sln` in the `PayRaise` directory to open the application.

c) ***Adding controls to the Form.*** Add two `NumericUpDown` controls to the `Form`. The first `NumericUpDown` control should be provided to allow the user to specify the pay raise percentage. The user should only be able to specify percentages in the range of 3–8 percent. Create the second `NumericUpDown` control for users to select the number of years in the range 1–50. Then add a multiline `TextBox` control to the application. Ensure that the user cannot modify the text in the `NumericUpDown` and `TextBox` con-

trols. Resize and move the controls you created so that they follow the GUI Design Guidelines as in Fig. 11.20.

d) ***Adding a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to use the `For`…`Next` statement to compute the yearly salary amounts, based on the yearly pay raise.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a starting wage per hour, the size of the yearly raise and the number of years worked. Click the **Calculate** `Button` and verify that the correct amount after each year is displayed in the **Yearly amount earned:** `TextBox`.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 11.17 Solution
' PayRaise.vb

Public Class FrmPayRaise
   Inherits System.Windows.Forms.Form

   ' Window Form Designer generated code

   ' invoke when btnCalculate Button is clicked
   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      Dim intYears As Integer = updYears.Text
      Dim intCounter As Integer
      Dim decWage As Decimal = Val(txtWage.Text)
      Dim intCurrentYear As Integer = 0
      Dim decTotal As Decimal = 0

      ' create headers to display in txtResult
      txtResult.Text = "Year" & ControlChars.Tab & _
         "Amount" & ControlChars.CrLf

      ' calculate first year's total
      decTotal += (decWage * 40 * 52)

      ' display wages per year with raise
      For intCounter = 1 To intYears Step 1

         ' determine if raise should be applied
         If intCounter <> 1 Then

            ' calculate total with raise amount
            decTotal *= 1 + ((updRaise.Text)/100)
         End If

         intCurrentYear += 1 ' increment intYear count

         ' append amounts to string displayed in
         ' txtResult TextBox
         txtResult.Text &= (intCurrentYear & _
            ControlChars.Tab & _
            String.Format("{0:C}", decTotal) & _
            ControlChars.CrLf)
      Next

```

```
    End Sub ' btnCalculate_Click

End Class ' FrmPayRaise
```

TUTORIAL 12

Security Panel Application

Introducing the `Select Case` Multiple-Selection Statement

Solutions

Instructor's Manual Exercise Solutions Tutorial 12

MULTIPLE-CHOICE QUESTIONS

12.1 The _____ keywords signify the end of a `Select Case` statement.

a) `End Case` b) `End Select`
c) `End Select Case` d) `Case End`

12.2 The expression _____ returns the current system time and date.

a) `Date.DateTime` b) `Date.SystemDateTime`
c) `Date.Now` d) `Date.SystemTimeDate`

12.3 You can hide information entered into a `TextBox` by setting that `TextBox`'s _____ property to a character; that character will be displayed for every character entered by the user.

a) `PrivateChar` b) `Mask`
c) `MaskingChar` d) `PasswordChar`

12.4 Which of the following is a syntax error?

a) Having duplicate `Case` statements in the same `Select Case` statement
b) Having a `Case` statement in which the value to the left of a `To` keyword is larger than the value to the right
c) Preceding a `Case` statement with the `Case Else` statement in a `Select Case` statement
d) Using keyword `Is` in a `Select Case` statement

12.5 Keyword _____ is used to specify a range in a `Case` statement.

a) `Also` b) `Between`
c) `To` d) `From`

12.6 _____ separates multiple values tested in a `Case` statement.

a) A comma b) An underscore
c) Keyword `Also` d) A semicolon

12.7 The _____ method inserts a value in a `ListBox`.

a) `Append` b) `Items.Insert`
c) `Insert` d) `Items.Add`

12.8 If the value on the left of the `To` keyword in a `Case` statement is larger than the value on the right, _____.

a) a syntax error occurs b) the body of the `Case` statement executes
c) the body of the `Case` statement never executes
d) the statement causes a runtime error

12.9 The expression following the keywords `Select Case` is called a(n) _____.

a) guard condition b) controlling expression
c) selection expression d) case expression

12.10 To prevent a user from modifying text in a `TextBox`, set its _____ property to `False`.

a) `Enabled` b) `Text`
c) `TextChange` d) `Editable`

Answers: 12.1) b. 12.2) c. 12.3) d. 12.4) c. 12.5) c. 12.6) a. 12.7) d. 12.8) c. 12.9) b. 12.10) a.

EXERCISES

12.11 ***(Sales Commission Calculator Application)*** Develop an application that calculates a salesperson's commission from the number of items sold (Fig. 12.17). Assume that all items have a fixed price of 10 dollars per unit. Use a `Select Case` statement to implement the following sales commission schedule:

Fewer than 10 items sold = 1% commission
Between 10 and 40 items sold = 2% commission
Between 41 and 100 items sold = 4% commission
More than 100 items sold = 8% commission

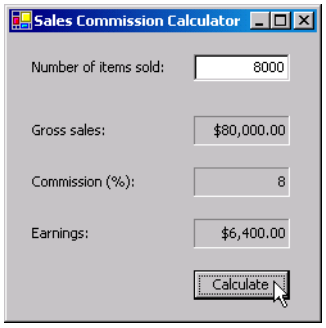


Figure 12.17 Sales Commission Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial12\Exercises\SalesCommissionCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SalesCommissionCalculator.sln` in the `SalesCommissionCalculator` directory to open the application.

c) ***Defining an event handler for the `Button`'s `Click` event.*** Create an event handler for the **Calculate** `Button`'s `Click` event.

d) ***Display the salesperson's gross sales.*** In your new event handler, multiply the number of items that the salesperson has sold by `10`, and display the resulting gross sales as a monetary amount.

e) ***Calculate the salesperson's commission percentage.*** Use a `Select Case` statement to compute the salesperson's commission percentage, from the number of items sold. The rate that is selected is applied to all the items the salesperson sold.

f) ***Display the salesperson's earnings.*** Multiply the salesperson's gross sales by the commission percentage determined in the previous step to calculate the salesperson's earnings. Remember to divide by `100` to obtain the percentage.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value for the number of items sold and click the **Calculate** `Button`. Verify that the gross sales displayed is correct, that the percentage of commission is correct and that the earnings displayed is correct based on the commission assigned.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 12.11 Solution
' SalesCommissionCalculator.vb

Public Class FrmSalesCommission
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoked when Calculate Button is clicked
   Private Sub btnCalculate_Click(ByVal sender As _
```

```
        System.Object, ByVal e As System.EventArgs) _
        Handles btnCalculate.Click

        lblGrossSalesResult.Text = String.Format("{0:C}", _
           Val(txtItemsSold.Text) * 10)

        Dim intItemsSold As Integer = Val(txtItemsSold.Text)

        ' Select Case used to determine commission percentage
        Select Case intItemsSold

           Case Is < 10
              lblCommissionPercentageResult.Text = 1

           Case 10 To 40
              lblCommissionPercentageResult.Text = 2

           Case 41 To 100
              lblCommissionPercentageResult.Text = 4

           Case Is > 100
              lblCommissionPercentageResult.Text = 8

        End Select

        ' calculate the earnings
        lblEarningsResult.Text = String.Format("{0:C}", _
           lblGrossSalesResult.Text * _
           (lblCommissionPercentageResult.Text / 100))

     End Sub ' btnCalculate_Click

  End Class ' FrmSalesCommission
```

12.12 ***(Cash Register Application)*** Use the numeric keypad from the **Security Panel** application to build a **Cash Register** application (Fig. 12.18). In addition to numbers, the cash register should include a decimal point `Button`. Apart from this numeric operation, there should be **Enter**, **Delete**, **Clear** and **Total** `Buttons`. Sales tax should be calculated on the amount purchased. Use a `Select Case` statement to compute sales tax. Add the tax amount to the subtotal to calculate the total. Display the tax and total for the user. Use the following sales-tax percentages, which are based on the amount of money spent:

Amounts under $100 = 5% (.05) sales tax
Amounts between $100 and $500 = 7.5% (.075) sales tax
Amounts above $500 = 10% (.10) sales tax

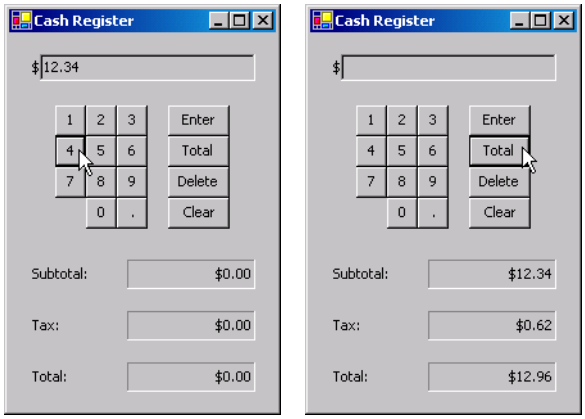


Figure 12.18 Cash Register GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial12\Exercises\CashRegister` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CashRegister.sln` in the `CashRegister` directory to open the application.

c) ***Define event handlers for the numeric `Button`s and decimal point in the keypad.*** Create event handlers for each of these `Button`'s `Click` events. Have each event handler concatenate the proper value to the `TextBox` at the top of the `Form`.

d) ***Define an event handler for the Enter `Button`'s `Click` event.*** Create an event handler for this `Button`'s `Click` event. Have this event handler add the current amount to the subtotal and display the new subtotal.

e) ***Define an event handler for the Total `Button`'s `Click` event.*** Create an event handler for this `Button`'s `Click` event. Have this event handler use the subtotal to compute the tax amount.

f) ***Define an event handler for the Clear `Button`'s `Click` event.*** Create an event handler for this `Button`'s `Click` event. Have this event handler clear the user input and display the value `$0.00` for the subtotal, sales tax and total.

g) ***Define an event handler for the Delete `Button`'s `Click` event.*** Create an event handler for this `Button`'s `Click` event. Have this event handler clear only the data in the `TextBox`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Use the keypad to enter various dollar amounts, clicking the **Enter** `Button` after each. After several amounts have been entered, click the **Total** `Button` and verify that the appropriate sales tax and total are displayed. Enter several values again and click the **Delete** `Button` to clear the current input. Click the **Clear** `Button` to clear all the output values.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 12.12 Solution
' CashRegister.vb

Public Class FrmCashRegister
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoked when btnOne is clicked
   Private Sub btnOne_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
```

```
      Handles btnOne.Click

      txtCurrentPrice.Text &= "1"
   End Sub ' btnOne_Click

   ' invoked when btnTwo is clicked
   Private Sub btnTwo_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnTwo.Click

      txtCurrentPrice.Text &= "2"
   End Sub ' btnTwo_Click

   ' invoked when btnThree is clicked
   Private Sub btnThree_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnThree.Click

      txtCurrentPrice.Text &= "3"
   End Sub ' btnThree_Click

   ' invoked when btnFour is clicked
   Private Sub btnFour_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnFour.Click

      txtCurrentPrice.Text &= "4"
   End Sub ' btnFour_Click

   ' invoked when btnFive is clicked
   Private Sub btnFive_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnFive.Click

      txtCurrentPrice.Text &= "5"
   End Sub ' btnFive_Click

   ' invoked when btnSix is clicked
   Private Sub btnSix_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSix.Click

      txtCurrentPrice.Text &= "6"
   End Sub ' btnSix_Click

   ' invoked when btnSeven is clicked
   Private Sub btnSeven_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSeven.Click

      txtCurrentPrice.Text &= "7"
   End Sub ' btnSeven_Click

   ' invoked when btnEight is clicked
   Private Sub btnEight_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnEight.Click

      txtCurrentPrice.Text &= "8"
   End Sub ' btnEight_Click

```

```
    ' invoked when btnNine is clicked
    Private Sub btnNine_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnNine.Click

       txtCurrentPrice.Text &= "9"
    End Sub ' btnNine_Click

    ' invoked when btnZero is clicked
    Private Sub btnZero_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnZero.Click

       txtCurrentPrice.Text &= "0"
    End Sub ' btnZero_Click

    ' invoked when btnPoint is clicked
    Private Sub btnPoint_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnPoint.Click

       txtCurrentPrice.Text &= "."
    End Sub ' btnPoint_Click

    ' invoked when btnEnter is clicked
    Private Sub btnEnter_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnEnter.Click

       ' variable to store new value
       Dim decAmount As Decimal

       ' store value in txtCurrentPrice to decAmount
       decAmount = Val(txtCurrentPrice.Text)

       ' add input amount to dblTotal and clear TextBox
       lblSubTotalValue.Text = String.Format("{0:C}", _
          lblSubTotalValue.Text + decAmount)

       txtCurrentPrice.Clear() ' clear the TextBox
    End Sub ' btnEnter_Click

    ' invoked when btnTotal is clicked
    Private Sub btnTotal_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnTotal.Click

       Dim dblTaxRate As Double

       ' determines tax rate based on subtotal
       Select Case lblSubTotalValue.Text

          Case Is < 100
             dblTaxRate = 0.05

          Case 100 To 500
             dblTaxRate = 0.075

          Case Is > 500
             dblTaxRate = 0.1

```

```
      End Select

      ' display subtotal, tax and total
      lblSubTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text)

      lblTaxValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text * dblTaxRate)

      lblTotalValue.Text = String.Format("{0:C}", _
         (lblSubTotalValue.Text + _
         lblSubTotalValue.Text * dblTaxRate))
   End Sub ' btnTotal_Click

   ' invoked when btnClear is clicked
   Private Sub btnClear_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnClear.Click

      ' clear txtCurrentPrice and set remaining Labels to $0.00
      txtCurrentPrice.Clear()
      lblSubTotalValue.Text = "$0.00"
      lblTaxValue.Text = "$0.00"
      lblTotalValue.Text = "$0.00"
   End Sub ' btnClear_Click

   ' invoked when btnDelete is clicked
   Private Sub btnDelete_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnDelete.Click

      txtCurrentPrice.Clear() ' clear the TextBox
   End Sub ' btnDelete_Click

End Class ' FrmCashRegister
```

12.13 ***(Income Tax Calculator Application)*** Create an application that computes the amount of income tax that a person must pay, depending upon that person's salary. Your application should perform as shown in Fig. 12.19. Use the following income ranges and corresponding tax rates:

Under $20,000 = 2% income tax
$20,000 –$50, 000 = 5% income tax
$50,001 – $75,000 = 10% income tax
$75,001 – $100,000= 15% income tax
Over $100,000 = 20% income tax

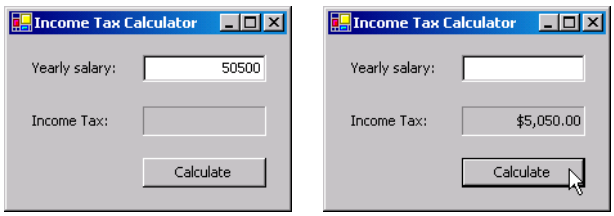


Figure 12.19 Income Tax Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial12\Exercises\IncomeTaxCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `IncomeTaxCalculator.sln` in the `IncomeTaxCalculator` directory to open the application.

c) ***Define an event handler for the Calculate `Button`'s `Click` event.*** Use the designer to create an event handler for this `Button`'s `Click` event. Have this event handler use a `Select Case` statement to determine the user's income-tax percentage. For simplicity, this value should then be multiplied by the user's salary and displayed in the output `Label`.

d) ***Running the application.*** Select **Debug > Start** to run your application. Enter a yearly salary and click the **Calculate** `Button`. Verify that the appropriate income tax is displayed, based on the ranges listed in the exercise description.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 12.13 Solution
' IncomeTax.vb

Public Class FrmIncomeTax
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoked when Calculate Button is clicked
   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      Dim dblPercent As Double
      Dim intSalary As Integer = Val(txtSalary.Text)

      ' determine income percentage
      Select Case intSalary

         Case Is < 20000
            dblPercent = 0.02

         Case 20000 To 50000
            dblPercent = 0.05

         Case 50001 To 75000
            dblPercent = 0.1

         Case 75001 To 100000
            dblPercent = 0.15

         Case Is > 100000
            dblPercent = 0.2

      End Select

      ' display result in currency format
      lblResult.Text = String.Format("{0:C}", _
         intSalary * dblPercent)

      txtSalary.Clear() ' clear the TextBox
   End Sub ' btnCalculate_Click

End Class ' FrmIncomeTax
```

What does this code do?

12.14 What is output by the following code? Assume that `btnDonation` is a `Button`, `txtDonation` is a `TextBox` and `lblMessage` is an output `Label`.

```
Private Sub btnDonation_Click(ByVal sender As _
   System.Object, ByVal e As System.EventArgs) _
   Handles btnDonation.Click

   Select Case Val(txtDonationAmount.Text)

      Case 0
         lblMessage.Text = "Please consider donating to our cause."

      Case 1 To 100
         lblMessage.Text = "Thank you for your donation."

      Case Is > 100
         lblMessage.Text = "Thank you very much for your donation!"

      Case Else
         lblMessage.Text = "Please enter a valid amount."

   End Select

End Sub
```

Answer: The output `Label` `lblMessage` displays "`Please consider donating to our cause`" if the user inputs 0 for a donation amount, "`Thank you for your donation`" if the user enters a dollar amount between 1 and 100, "`Thank you very much for your donation`" if the user enters a value greater than 100 dollars and "`Please enter a valid amount`" if the user enters invalid data.

What's wrong with this code?

12.15 This `Select Case` statement should determine whether an `Integer` is even or odd. Find the error(s) in the following code:

```
1  Select Case intValue Mod 2
2
3     Case 0
4        lblOutput.Text = "Odd Integer"
5
6     Case 1
7        lblOutput.Text = "Even Integer"
8
9  End Select
```

Answer: Line 4 and line 7 should be swapped.

```
1  Select Case intValue Mod 2
2
3     Case 0
4        lblOutput.Text = "Even Integer"
5
6     Case 1
7        lblOutput.Text = "Odd Integer"
8
9  End Select
```

Using the Debugger

12.16 (*Discount Calculator Application*) The **Discount Calculator** application determines the discount the user will receive, based on how much money the user spends. A 15% discount is received for purchases over $200, a 10% discount is received for purchases between $150–$199, a 5% discount is received for purchases between $100–$149 and a 2% discount is received for purchases between $50–$99. While testing your application, you notice that the application is not calculating the discount properly for some values. Use the debugger to find and fix the logic error(s) in the application. Figure 12.20 displays the correct output for the application.

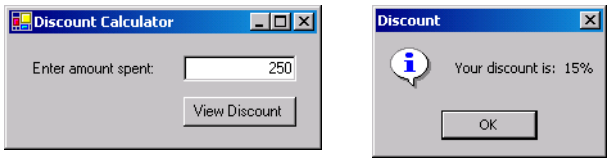


Figure 12.20 Correct output for the **Discount Calculator** application.

Answer:

Removed superfluous `Case Is > 200` statement that prevented correct `Case Is > 200` statement from executing

Incorrect code given to students displayed a 5% discount for values between 100 and 150 inclusive

```
1  ' Exercise 12.16 Solution
2  ' Discount.vb
3
4  Public Class FrmDiscountCalculator
5     Inherits System.Windows.Forms.Form
6
7     ' Windows Form Designer generated code
8
9     ' display user's discount
10    Private Sub btnView_Click(ByVal sender As System.Object, _
11       ByVal e As System.EventArgs) Handles btnView.Click
12
13       Dim intTotal As Integer ' amount spent
14       Dim strOutput As String ' displays discount
15
16       intTotal = Val(txtAmount.Text) ' get user's total
17
18       Select Case intTotal
19
20          Case 50 To 99 ' user spent between $50-99
21             strOutput = "Your discount is:  2%"
22
23          Case 100 To 149 ' user spent between $100-149
24             strOutput = "Your discount is:  5%"
25
26          Case 150 To 199 ' user spent between $150-199
27             strOutput = "Your discount is:  10%"
28
29          Case Is >= 200 ' user spent over $200
30             strOutput = "Your discount is:  15%"
31
32          Case Else ' user spent less than $50
33             strOutput = "No discount"
34
35       End Select
36
37       ' display discount to user
38       MessageBox.Show(strOutput, "Discount", _
39          MessageBoxButtons.OK, MessageBoxIcon.Information)
40
41    End Sub ' btnView_Click
```

```
End Class ' FrmDiscountCalculator
```

Programming Challenge

12.17 ***(Enhanced Cash Register Application)*** Modify the **Cash Register** application (Exercise 12.12) to include the operations addition, subtraction and multiplication. Remove the **Enter** `Button`, and replace it with the addition (**+**), subtraction (**–**) and multiplication (*****) `Buttons`. These `Buttons` should take the value displayed in the **Subtotal:** field and the value displayed in the upper `Label` and perform the operation of the clicked `Button`. The result should be displayed in the **Subtotal:** field. Figure 12.21 displays the enhanced **Cash Register** application GUI.

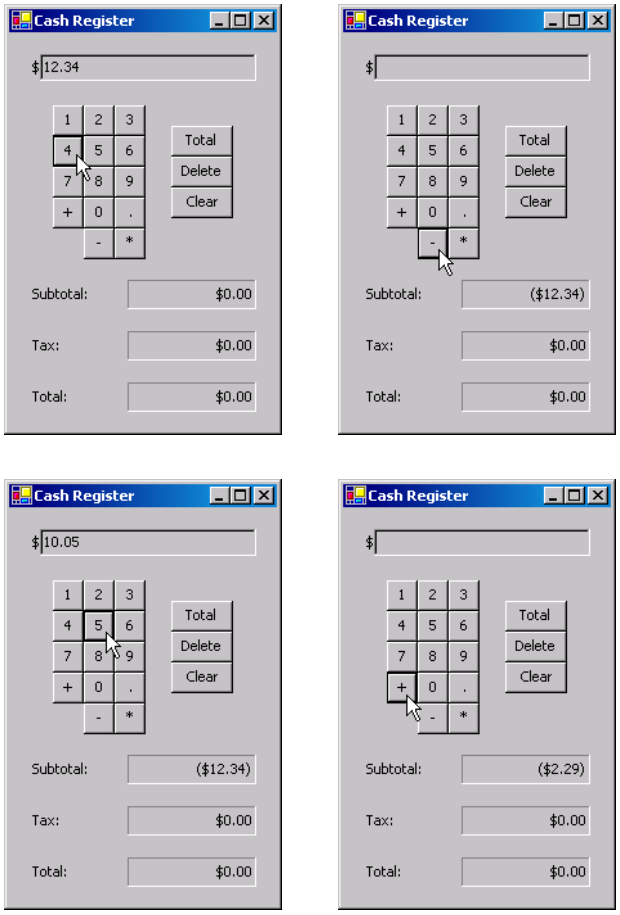


Figure 12.21 Enhanced Cash Register GUI.

Answer:

```
' Exercise 12.17 Solution
' CashRegister.vb

Public Class FrmCashRegister
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoked when btnOne is clicked
   Private Sub btnOne_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnOne.Click

```

```
      txtCurrentPrice.Text &= "1"
   End Sub ' btnOne_Click

   ' invoked when btnTwo is clicked
   Private Sub btnTwo_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnTwo.Click

      txtCurrentPrice.Text &= "2"
   End Sub ' btnTwo_Click

   ' invoked when btnThree is clicked
   Private Sub btnThree_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnThree.Click

      txtCurrentPrice.Text &= "3"
   End Sub ' btnThree_Click

   ' invoked when btnFour is clicked
   Private Sub btnFour_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnFour.Click

      txtCurrentPrice.Text &= "4"
   End Sub ' btnFour_Click

   ' invoked when btnFive is clicked
   Private Sub btnFive_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnFive.Click

      txtCurrentPrice.Text &= "5"
   End Sub ' btnFive_Click

   ' invoked when btnSix is clicked
   Private Sub btnSix_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSix.Click

      txtCurrentPrice.Text &= "6"
   End Sub ' btnSix_Click

   ' invoked when btnSeven is clicked
   Private Sub btnSeven_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSeven.Click

      txtCurrentPrice.Text &= "7"
   End Sub ' btnSeven_Click

   ' invoked when btnEight is clicked
   Private Sub btnEight_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnEight.Click

      txtCurrentPrice.Text &= "8"
   End Sub ' btnEight_Click

   ' invoked when btnNine is clicked
   Private Sub btnNine_Click(ByVal sender As _
```

```
      System.Object, ByVal e As System.EventArgs) _
      Handles btnNine.Click

      txtCurrentPrice.Text &= "9"
   End Sub ' btnNine_Click

   ' invoked when btnZero is clicked
   Private Sub btnZero_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnZero.Click

      txtCurrentPrice.Text &= "0"
   End Sub ' btnZero_Click

   ' invoked when btnPoint is clicked
   Private Sub btnPoint_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPoint.Click

      txtCurrentPrice.Text &= "."
   End Sub ' btnPoint_Click

   ' invoked when btnTotal is clicked
   Private Sub btnTotal_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnTotal.Click

      Dim dblTaxRate As Double

      ' determines tax rate based on subtotal
      Select Case lblSubTotalValue.Text

         Case Is < 100
            dblTaxRate = 0.05

         Case 100 To 500
            dblTaxRate = 0.075

         Case Is > 500
            dblTaxRate = 0.1

      End Select

      ' display subtotal, tax and total in Labels
      lblSubTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text)

      lblTaxValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text * dblTaxRate)

      lblTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text + (lblSubTotalValue.Text * _
         dblTaxRate))
   End Sub ' btnTotal_Click

   ' invoked when btnClear is clicked
   Private Sub btnClear_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnClear.Click

      ' clear all Labels and set total and tax to $0.00
```

```
      txtCurrentPrice.Clear()
      lblSubTotalValue.Text = "$0.00"
      lblTaxValue.Text = "$0.00"
      lblTotalValue.Text = "$0.00"
   End Sub ' btnClear_Click

   ' invoked when btnDelete is clicked
   Private Sub btnDelete_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnDelete.Click

      txtCurrentPrice.Clear() ' clear the TextBox
   End Sub ' btnDelete_Click

   ' invoked when plus Button is pressed
   Private Sub btnPlus_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPlus.Click

      Dim decAmount As Decimal

      ' store txtCurrentPrice value for adding
      decAmount = Val(txtCurrentPrice.Text)

      ' add input amount to dblTotal and clear Label
      lblSubTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text + decAmount)

      txtCurrentPrice.Clear() ' clear the TextBox
   End Sub ' btnPlus_Click

   ' invoked when minus Button clicked
   Private Sub btnMinus_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnMinus.Click

      Dim decAmount As Decimal

      ' store txtCurrentPrice to subtract from lblSubTotalValue
      decAmount = Val(txtCurrentPrice.Text)

      ' subtract decAmount from lblSubTotalValue
      lblSubTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text - decAmount)

      txtCurrentPrice.Clear() ' clear txtCurrentPrice
   End Sub ' btnMinus_Click

   ' invoked when multiply Button is clicked
   Private Sub btnMultiply_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnMultiply.Click

      Dim decAmount As Decimal ' store price entered

      ' store amount to multiply
      decAmount = Val(txtCurrentPrice.Text)

      ' multiply subtotal amount with decAmount
      lblSubTotalValue.Text = String.Format("{0:C}", _
         lblSubTotalValue.Text * decAmount)
```

```
197
198         txtCurrentPrice.Clear() ' clear the TextBox
199      End Sub ' btnMultiply_Click
200
201   End Class ' FrmCashRegister
```

TUTORIAL 13

Enhancing the Wage Calculator Application

Introducing `Function` Procedures and Sub Procedures

Solutions

Instructor's Manual Exercise Solutions Tutorial 13

MULTIPLE-CHOICE QUESTIONS

13.1 A procedure defined with keyword `Sub` _____.

a) must specify a return type
b) does not accept parameters
c) returns a value
d) does not return a value

13.2 The technique of developing large applications from small, manageable pieces is known as _____.

a) divide and conquer
b) returning a value
c) click and mortar
d) a building-block algorithm

13.3 What is the difference between `Sub` and `Function` procedures?

a) `Sub` procedures return values, `Function` procedures do not.
b) `Function` procedures return values, `Sub` procedures do not.
c) `Sub` procedures accept parameters, `Function` procedures do not.
d) `Function` procedures accept parameters, `Sub` procedures do not.

13.4 What occurs after a procedure call is made?

a) Control is given to the called procedure. After the procedure is run, the application continues execution at the point where the procedure call was made.
b) Control is given to the called procedure. After the procedure is run, the application continues execution with the statement after the called procedure's definition.
c) The statement before the procedure call is executed.
d) The application terminates.

13.5 `Functions` can return _____ value(s).

a) zero
b) exactly one
c) one or more
d) any number of

13.6 Which of the following must be true when making a procedure call?

a) The number of arguments in the procedure call must match the number of parameters in the procedure header.
b) The argument types must be compatible with their corresponding parameter types.
c) Both a and b.
d) None of the above.

13.7 Which of the following statements correctly returns the variable `intValue` from a `Function` procedure?

a) `Return Dim intValue`
b) `Return intValue As Integer`
c) `intValue Return`
d) `Return intValue`

13.8 The _____ `Button` executes the next statement in the application. If the next statement to execute contains a procedure call, the called procedure executes in its entirety.

a) Step Into
b) Step Out
c) Step Over
d) Steps

13.9 The first line of a procedure (including the keyword `Sub` or `Function`, the procedure name, the parameter list and the `Function` procedure return type) is known as the procedure _____.

a) body
b) title
c) call
d) header

13.10 Method _____ of class `Math` calculates the square root of the value passed as an argument.

a) `SquareRoot` b) `Root`
c) `Sqrt` d) `Square`

Answers: 13.1) d. 13.2) a. 13.3) b. 13.4) a. 13.5) b. 13.6) c. 13.7) d. 13.8) c. 13.9) d. 13.10) c.

EXERCISES

13.11 (*Temperature Converter Application*) Write an application that performs various temperature conversions (Fig. 13.28). The application should be capable of performing two types of conversions: degrees Fahrenheit to degrees Celsius and degrees Celsius to degrees Fahrenheit.

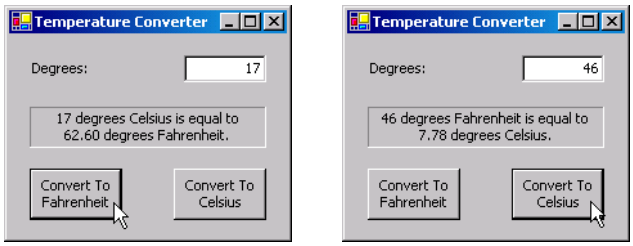


Figure 13.28 Temperature Converter GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial13\Exercises\TemperatureConversion` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `TemperatureConversion.sln` in the `TemperatureConversion` directory to open the application.

c) ***Convert Fahrenheit to Celsius.*** To convert degrees Fahrenheit to degrees Celsius, use this formula:

```
dblCelsius = (5 / 9) * (dblFahrenheit - 32)
```

d) ***Convert Celsius to Fahrenheit.*** To convert degrees Celsius to degrees Fahrenheit, use this formula:

```
dblFahrenheit = (9 / 5) * dblCelsius + 32
```

e) ***Adding event handlers to your application.*** Double click each `Button` to add the proper event handlers to your application. These event handlers will call procedures (that you will define in the next step) to convert the degrees entered to either Fahrenheit or Celsius. Each event handler will display the result in the application's output `Label`.

f) ***Adding `Function` procedures to your application.*** Create `Function` procedures to perform each conversion, using the formulas above. The user should provide the temperature to convert.

g) ***Formatting the temperature output.*** To format the temperature information, use the `String.Format` method. Use `F` as the formatting code to limit the temperature to two decimal places.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a temperature value. Click the **Convert to Fahrenheit** `Button` and verify that correct output is displayed based on the formula given. Click the **Convert to Celsius** `Button` and again verify that the output is correct.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 13.11 Solution
' TemperatureConversion.vb

```

```
Public Class FrmTemperatureConverter
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' converts degrees to Fahrenheit
   Private Sub btnConvertFahrenheit_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnConvertFahrenheit.Click

      Dim dblDegree As Double = Val(txtDegrees.Text)

      lblOutput.Text = dblDegree & _
         " degrees Celsius is equal to " & _
         ControlChars.CrLf & String.Format("{0:F}", _
         ConvertToFahrenheit(dblDegree)) & _
         " degrees Fahrenheit."

   End Sub ' btnConvertFahrenheit_Click

   ' converts degrees to Celsius
   Private Sub btnConvertCelsius_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnConvertCelsius.Click

      Dim dblDegree As Double = Val(txtDegrees.Text)

      lblOutput.Text = dblDegree & _
         " degrees Fahrenheit is equal to " & _
         ControlChars.CrLf & String.Format("{0:F}", _
         ConvertToCelsius(dblDegree)) & _
         " degrees Celsius."

   End Sub ' btnConvertCelsius_Click

   ' convert degree to Fahrenheit
   Function ConvertToFahrenheit(ByVal dblDegree As Double) As Double

      Return (9 / 5) * dblDegree + 32
   End Function ' ConvertToFahrenheit

   ' convert degree to Celsius
   Function ConvertToCelsius(ByVal dblDegree As Double) As Double

      Return (5 / 9) * (dblDegree - 32)
   End Function ' ConvertToCelsius

End Class ' FrmTemperatureConverter
```

13.12 *(**Display Square Application**)* Write an application that displays a solid square composed of a character input by the user (Fig. 13.29). The user also should input the size.

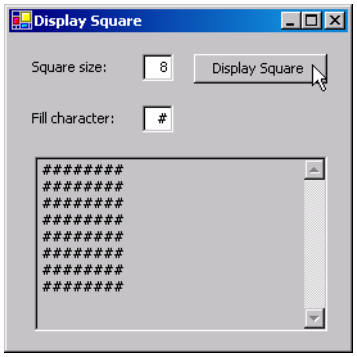


Figure 13.29 **Display Square** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial13\Exercises\DisplaySquare` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `DisplaySquare.sln` in the `DisplaySquare` directory to open the application.

c) ***Adding a Sub procedure.*** Write a `Sub` procedure `DisplaySquare` to display the solid square. The size should be specified by the `Integer` parameter `intSize`. The character that fills the square should be specified by the `String` parameter `strFillCharacter`. You should use a `For…Next` statement nested within another `For…Next` statement to create the square. The outer `For…Next` specifies what row is currently being displayed. The inner `For…Next` appends all the characters that form the row to a display `String`.

d) ***Adding an event handler for your `Button`'s `Click` event.*** Double click the **Display Square** `Button` to create the event handler. Program the event handler to call procedure `DisplaySquare`.

e) ***Displaying the output.*** Use the multiline `TextBox` provided to display the square. For example, if `intSize` is 8 and `strFillCharacter` is #, the application should look similar to Fig. 13.29.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter a size for the square (the length of each side) and a fill character. Click the **Display Square** `Button`. A square should be displayed of the size you specified, using the character you specified.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 13.12 Solution
' DisplaySquare.vb

Public Class FrmDisplaySquare
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' display square in TextBox
   Sub DisplaySquare(ByVal intSize As Integer, _
      ByVal strFillCharacter As String)

      ' declare loop variables
      Dim intRow As Integer     ' number of rows counter
      Dim intColumn As Integer ' number of columns counter
      Dim strOutput As String  ' output String

      ' loop until intRow reaches value of first argument (intSize)
```

```
      For intRow = 1 To intSize

         ' loop until intColumn reaches value of intSize
         For intColumn = 1 To intSize
            strOutput &= strFillCharacter
         Next

         strOutput &= ControlChars.CrLf ' add line to output
      Next

      txtOutput.Text = strOutput ' display square in output area
   End Sub ' DisplaySquare

   ' handles Display Square Button's Click event
   Private Sub btnDisplaySquare_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnDisplaySquare.Click

      ' if valid input is entered
      If txtSideSize.Text <> "" AndAlso _
         txtFillCharacter.Text <> "" Then

         DisplaySquare(Val(txtSideSize.Text), _
            txtFillCharacter.Text)
      Else
         MessageBox.Show("Square size and fill character needed", _
            "Incorrect Input", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      End If

   End Sub ' btnDisplaySquare_Click

End Class ' FrmDisplaySquare
```

13.13 (*Miles Per Gallon Application*) Drivers often want to know the miles per gallon their cars get so they can estimate gasoline costs. Develop an application that allows the user to input the numbers of miles driven and the number of gallons used for a tank of gas.

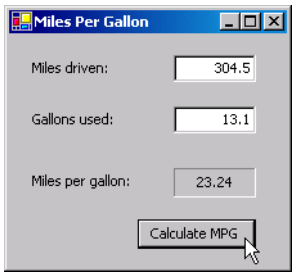


Figure 13.30 **Miles Per Gallon** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial13\Exercises\MilesPerGallon` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MilesPerGallon.sln` in the `MilesPerGallon` directory to open the application.

c) ***Calculating the miles per gallon.*** Write a `Function` procedure `MilesPerGallon` that takes the number of miles driven and gallons used (entered by the user), calculates the amount of miles per gallon and returns the miles per gallon for a tankful of gas.

d) ***Displaying the result***. Create a `Click` event handler for the **Calculate MPG** `Button` that invokes the `Function` procedure `MilesPerGallon` and displays the result returned from the procedure as in Fig. 13.30.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value for the number of miles driven and the amount of gallons used. Click the **Calculate MPG** `Button` and verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 13.13 Solution
' MilesPerGallon.vb

Public Class FrmMilesPerGallon
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' calculate and return miles per gallon
   Function MilesPerGallon( _
      ByVal dblMilesDriven As Double, _
      ByVal dblGallonsUsed As Double) As Double

      Return dblMilesDriven / dblGallonsUsed
   End Function ' MilesPerGallon

   ' handles CalculateMPG Button's Click event
   Private Sub btnCalculateMPG_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculateMPG.Click

      ' display miles per gallon
      lblOutputValue.Text = String.Format("{0:F}", _
         MilesPerGallon(Val(txtMilesDriven.Text), _
         Val(txtGallonsUsed.Text)))

   End Sub ' btnCalculateMPG_Click

End Class ' FrmMilesPerGallon
```

What does this code do?

13.14 What does the following code do? Assume this procedure is invoked by using `Mystery(70, 80)`.

```
Sub Mystery(ByVal intNumber1 As Integer, ByVal _
   intNumber2 As Integer)

   Dim intX As Integer
   Dim dblY As Double

   intX = intNumber1 + intNumber2
   dblY = intX / 2

   If dblY <= 60 Then
      lblResult.Text = "<= 60 "
   Else
      lblResult.Text = "Result is " & dblY
   End If
```

```
End Sub ' Mystery
```

Answer: This code calculates the average of two numbers. The user is informed of the average, but only if the average is above 60. In this case the two numbers entered are 70 and 80, which results in an average of 75, which will be displayed.

What's wrong with this code?

13.15 Find the error(s) in the following code, which should take an `Integer` value as an argument and return the value of that argument multiplied by two.

```
Function TimesTwo(ByVal intNumber As Integer) As Integer

   Dim intResult As Integer

   intResult = intNumber * 2
End Function ' TimesTwo
```

Answer: A `Function` procedure should `Return` a value. The default return value for a `Function` procedure with an `Integer` return type is 0. This `Function` procedure will always return 0. Corrected code:

```
Function TimesTwo(ByVal intNumber As Integer) As Integer

   Dim intResult As Integer

   intResult = intNumber * 2

   Return intResult
End Function ' TimesTwo
```

Using the Debugger

13.16 (*Gas Pump Application*) The **Gas Pump** application calculates the cost of gas at a local gas station. This gas station charges `$1.41` per gallon for **Regular** grade gas, `$1.47` per gallon for **Special** grade gas and `$1.57` per gallon for **Super+** grade gas. The user enters the number of gallons to purchase and clicks the desired grade. The application calls a `Sub` procedure to compute the total cost from the number of gallons entered and the selected grade. While testing your application, you noticed that one of your totals was incorrect, given the input.

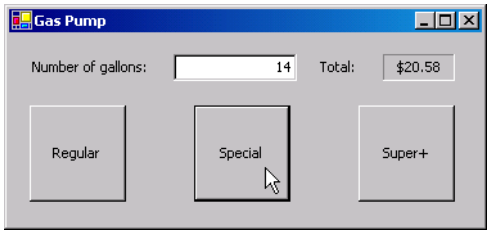


Figure 13.31 **Gas Pump** application executing correctly.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial13\Debugger\GasPumpIncorrect` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `GasPump.sln` in the `GasPumpIncorrect` directory to open the application.

c) ***Running the application.*** Select **Debug > Start** to run your application. Determine which total is incorrect.

d) ***Setting a breakpoint***. Set a breakpoint at the beginning of the event handler that is providing incorrect output. For instance, if the **Regular** `Button` is providing incorrect output when clicked, add a breakpoint at the beginning of that `Button`'s `Click` event handler. Use the debugger to help find any logic error(s) in the application.

e) ***Modifying the application***. Once you have located the error(s), modify the application so that it behaves correctly.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter a number of gallons and click the **Regular**, **Special** and **Super+** `Button`s. After each `Button` is clicked, verify that the total displayed is correct based on the prices given in this exercise's description.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 13.16 Solution
' GasPump.vb

Public Class FrmGasPump
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Dim intGallons As Integer = 0 ' number of gallons

   ' handles Regular Button's Click event
   Private Sub btnRegular_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnRegular.Click

      intGallons = Val(txtNumberGallons.Text)

      ' call procedure to determine total
      ' first argument is Button's Text
      Total(btnRegular.Text, intGallons)
   End Sub ' btnRegular_Click

   ' handles Special Button's Click event
   Private Sub btnSpecial_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSpecial.Click

      intGallons = Val(txtNumberGallons.Text)

      ' call procedure to determine total
      ' first argument is Button's Text
      Total(btnSpecial.Text, intGallons)
   End Sub ' btnSpecial_Click

   ' handles Super Button's Click event
   Private Sub btnSuper_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSuper.Click

      intGallons = Val(txtNumberGallons.Text)

      ' call procedure to determine total
      ' first argument is Button's Text
      Total(btnSuper.Text, intGallons)
   End Sub ' btnSuper_Click

   ' calculate total cost of gas
   Sub Total(ByVal strGrade As String, ByVal intGallons As Integer)
```

```
46
47         ' determine grade selected and output total
48         Select Case strGrade
49
50            Case "Regular"
51               lblTotalResult.Text = _
52                  String.Format("{0:C}", 1.41 * intGallons)
53
54            Case "Special"
55               lblTotalResult.Text = _
56                  String.Format("{0:C}", 1.47 * intGallons)
57
58            Case "Super+"
59               lblTotalResult.Text = _
60                  String.Format("{0:C}", 1.57 * intGallons)
61
62         End Select
63
64      End Sub ' Total
65
66   End Class ' FrmGasPump
```

Code provided to student had value `1.91` in place of `1.47`

Programming Challenge

13.17 ***(Prime Numbers Application)*** An `Integer` greater than 1 is said to be prime if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime numbers, but 4, 6, 8 and 9 are not. Write an application that takes two numbers (representing a lower bound and an upper bound) and determines all of the prime numbers within the specified bounds, inclusive.

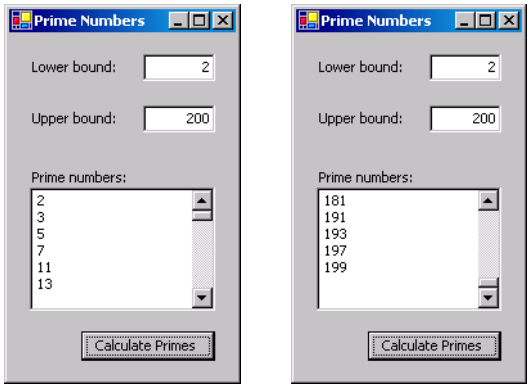


Figure 13.32 **Prime Numbers** application.

a) ***Creating the application***. Create an application named `PrimeNumbers` and have its GUI appear as shown in Fig. 13.32. Add an event handler for the **Calculate Primes** `Button`'s `Click` event.

b) ***Checking for prime numbers***. Write a `Function` procedure `Prime` that returns `True` if a number is prime, `False` otherwise.

c) ***Limiting user input***. Allow users to enter a lower bound (`intLower`) and an upper bound (`intUpper`). Prevent the user from entering bounds less than or equal to 1, or an upper bound that is smaller than the lower bound.

d) ***Displaying the prime numbers***. Call `Function` procedure `Prime` from your event handler to determine which numbers between the lower and upper bounds are prime. Then have the event handler display the prime numbers in a multiline, scrollable `TextBox`, as in Fig. 13.32.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a lower bound and an upper bound that is smaller than the lower bound. Click the **Calculate Primes** `Button`. You should receive an error message. Enter negative bounds

and click the **Calculate Primes** `Button`. Again, you should receive an error message. Enter valid bounds and click the **Calculate Primes** Button. This time, the primes within that range should be displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 13.17 Solution
' PrimeNumbers.vb

Public Class FrmPrimeNumbers
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' determine if number is prime
   Function Prime(ByVal intNumber As Integer) As Boolean

      Dim intCount As Integer ' declare counter

      ' set square root of intNumber as limit
      Dim intLimit As Integer = Math.Sqrt(intNumber)

      ' loop until intCount reaches square root of intNumber
      For intCount = 2 To intLimit

         If intNumber Mod intCount = 0 Then
            Return False ' number is not prime
         End If

      Next

      Return True ' number is prime
   End Function ' Prime

   ' handles Calculate Primes Button's Click event
   Private Sub btnCalculatePrimes_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculatePrimes.Click

      ' declare variables
      Dim intLowerBound As Integer = Val(txtLowerBound.Text)
      Dim intUpperBound As Integer = Val(txtUpperBound.Text)
      Dim intCounter As Integer
      Dim strOutput As String

      If intLowerBound <= 1 OrElse intUpperBound <= 1 Then
         MessageBox.Show("Bounds must be greater than 1", _
            "Invalid Bounds", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      ElseIf intUpperBound < intLowerBound Then
         MessageBox.Show("Upper bound cannot be less than " & _
            "lower bound", "Invalid Bounds", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      Else

         ' loop from lower bound to upper bound
         For intCounter = intLowerBound To intUpperBound

            ' if prime number, display in TextBox
```

```
            If Prime(intCounter) = True Then
               strOutput &= (intCounter & ControlChars.CrLf)
            End If

         Next

      End If

      txtPrimeNumbers.Text = strOutput
   End Sub ' btnCalculatePrimes_Click

End Class ' FrmPrimeNumbers
```

TUTORIAL 14

Shipping Time Application

Using Dates and Timers
Solutions

Instructor's Manual Exercises Solutions Tutorial 14

MULTIPLE-CHOICE QUESTIONS

14.1 The _____ allows you to store and manipulate date information easily.

a) `Date` structure
b) `DatePicker` control
c) `GroupBox` control
d) `Now` property

14.2 You can _____ to a `Date` variable.

a) add hours
b) add days
c) subtract hours
d) All of the above.

14.3 To subtract one day from `Date` variable `dtmDay`'s value, assign the value returned by _____ to `dtmDay`.

a) `dtmDay.AddHours(-24)`
b) `dtmDay.SubtractDays(1)`
c) `dtmDay.AddDays(-1)`
d) Both a and c.

14.4 The time 3:45 and 35 seconds in the afternoon would be formatted as 03:45:35 PM according to the format string _____.

a) `"hh:mm:ss"`
b) `"hh:mm:ss tt"`
c) `"hh:mm:ss am:pm"`
d) `"h:m:s tt"`

14.5 A(n) _____ event occurs before the `Form` is displayed.

a) `LoadForm`
b) `InitializeForm`
c) `Load`
d) `FormLoad`

14.6 `Timer` property `Interval` sets the rate at which `Tick` events occur in _____.

a) nanoseconds
b) microseconds
c) milliseconds
d) seconds

14.7 To set `Date` `dtmNow`'s time five hours earlier, use _____.

a) `dtmNow = dtmNow.SubtractHours(5)`
b) `dtmNow = dtmNow.AddHours(-5)`
c) `dtmNow = dtmNow.AddHours(5)`
d) `dtmNow.AddHours(-5)`

14.8 A(n) _____ is a container.

a) `GroupBox`
b) `Form`
c) `Timer`
d) Both a and b.

14.9 A `Date` variable stores hour values in the range _____.

a) 1 to 12
b) 0 to 12
c) 0 to 24
d) 0 to 23

14.10 A `DateTimePicker`'s _____ property specifies the format string with which to display the date.

a) `CustomFormat`
b) `FormatString`
c) `Format`
d) `Text`

Answers: 14.1) a. 14.2) d. 14.3) d. 14.4) b. 14.5) c. 14.6) c. 14.7) b. 14.8) d. 14.9) d. 14.10) a.

EXERCISES

14.11 (***World Clock Application***) Create an application that displays the current time in Los Angeles, Atlanta, London and Tokyo. Use a `Timer` to update the clock every second. Assume that your local time is the time in Atlanta. Atlanta is three hours later than Los Angeles. London is five hours later than Atlanta. Tokyo is eight hours later than London. The application should look similar to Fig. 14.20.

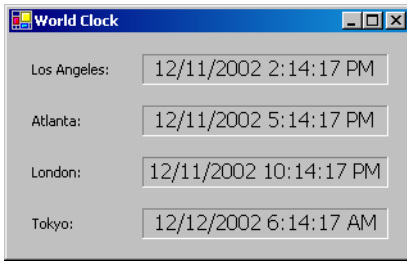


Figure 14.20 World Clock GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial14\Exercises\WorldClock` directory to your `C:\SimplyVB` directroy.

b) ***Opening the application's template file.*** Double click `WorldClock.sln` in the `WorldClock` directory to open the application.

c) ***Adding a `Timer` to the Form.*** Add a `Timer` control to the **World Clock** application. Set the `Timer` control's name property to `tmrClock`.

d) ***Adding a `Tick` event handler for `tmrClock`.*** Add a `Tick` event handler for `Timer` `tmrClock`. The event handler should calculate and display the current times for Los Angeles, Atlanta, London and Tokyo. Use the `Date` variable's `ToShortDateString` and `ToLongTimeString` methods to create the display text.

e) ***Running the application.*** Select **Debug > Start** to run your application. Look at the clock on your machine to verify that the time for Los Angeles is three hours earlier, the time in Atlanta is the same as what your clock says, the time in London is five hours later, and the time in Tokyo is 13 hours later (eight hours later than London).

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 14.11 Solution
' WorldClock.vb

Public Class FrmWorldClock
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' update times
   Private Sub tmrClock_Tick(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles tmrClock.Tick

      ' retrieve current time
      Dim dtmNow As Date = Date.Now

      ' display Los Angeles time
      lblLATime.Text = dtmNow.AddHours(-3).ToShortDateString & _
         " " & dtmNow.AddHours(-3).ToLongTimeString

      ' display Atlanta time
      lblAtlantaTime.Text = dtmNow.ToShortDateString & _
         " " & dtmNow.ToLongTimeString

      ' display London time
      lblLondonTime.Text = dtmNow.AddHours(5).ToShortDateString & _
         " " & dtmNow.AddHours(5).ToLongTimeString

      ' display Tokyo time
      lblTokyoTime.Text = dtmNow.AddHours(13).ToShortDateString & _
```

```
            " " & dtmNow.AddHours(13).ToLongTimeString

      End Sub ' tmrClock_Tick

   End Class ' FrmWorldClock
```

14.12 (*Shipping Time Application Enhancement*) During the winter, a distribution center in Denver, Colorado needs to receive seafood shipments to supply the local ski resorts. Enhance the **Shipping Time** application by adding Denver, Colorado as another shipping destination. Denver is two time zones west of Portland, meaning time is two hours earlier than Portland, Maine. Because there are no direct flights to Denver, shipments from Portland will take 8 hours.

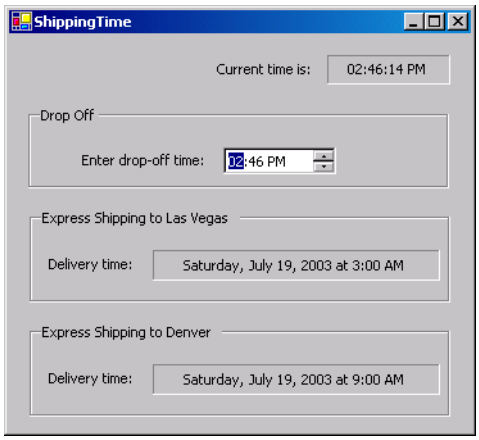


Figure 14.21 Enhanced Shipping Time GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial14\Exercises\ShippingTimeEnhanced` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ShippingTime.sln` in the `ShippingTimeEnhanced` directory to open the application.

c) ***Inserting a `GroupBox`.*** Resize the `Form` to fit the **Express Shipping to Denver** `GroupBox` as shown in Fig. 14.21. Add a `GroupBox` to the `Form`. Change the `Text` property of the `GroupBox` to indicate that it will contain the delivery time in Denver. Resize and move the `GroupBox` so that it resembles the GUI shown in Fig. 14.21.

d) ***Inserting `Labels`.*** In the `GroupBox` you just created, add an output `Label` to display the delivery time for a seafood shipment to Denver and a corresponding descriptive `Label`.

e) ***Inserting code to the `DisplayDeliveryTime` procedure.*** Add code to `DisplayDeliveryTime` procedure to compute and display the delivery time in Denver.

f) ***Running the application.*** Select **Debug > Start** to run your application. Select various drop off times and ensure the delivery times are correct for both Las Vegas and Denver.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 14.12 Solution
' ShippingTime.vb

Public Class FrmShippingTime
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

```

```
    ' update current time every second
    Private Sub tmrClock_Tick(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles tmrClock.Tick

       ' print current time
       lblCurrentTime.Text = String.Format("{0:hh:mm:ss tt}", _
          Date.Now)

    End Sub ' tmrClock_Tick

    ' initialize DateTimePicker status when Form loads
    Private Sub FrmShippingTime_Load(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) Handles _
       MyBase.Load

       Dim dtmCurrentTime As Date = Date.Now ' store current time

       ' set range of possible drop-off times
       dtpDropOff.MinDate = New Date(dtmCurrentTime.Year, _
          dtmCurrentTime.Month, dtmCurrentTime.Day, 0, 0, 0)

       dtpDropOff.MaxDate = dtpDropOff.MinDate.AddDays(1)

       ' display the delivery time
       DisplayDeliveryTime()

    End Sub ' FrmShippingTime_Load

    ' update ship time on change of drop-off time
    Private Sub dtpDropOff_ValueChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) Handles _
       dtpDropOff.ValueChanged

       ' display the delivery time
       DisplayDeliveryTime()

    End Sub ' dtpDropOff_ValueChanged

    ' calculates and displays the delivery time
    Sub DisplayDeliveryTime() As Date

       ' print initial delivery time
       Dim dtmDelivery As Date = DepartureTime()

       ' add 3 hours to departure and display result
       dtmDelivery = dtmDelivery.AddHours(3)
       lblLasVegasTime.Text = dtmDelivery.ToLongDateString _
          & " at " & dtmDelivery.ToShortTimeString

       ' add 6 hours to departure and display result
       dtmDelivery = dtmDelivery.AddHours(6)
       lblDenverTime.Text = dtmDelivery.ToLongDateString _
          & " at " & dtmDelivery.ToShortTimeString

    End Sub ' DisplayDeliveryTime

    ' returns flight departure time for selected drop-off time
    Function DepartureTime() As Date

       Dim dtmCurrentDate As Date = Date.Now ' store current date
       Dim dtmDepartureTime As Date          ' store departure time
```

```
70
71          ' determine which flight the shipment takes
72          Select Case dtpDropOff.Value.Hour
73
74             ' seafood will be on the noon flight
75             Case 0 To 10
76                dtmDepartureTime = New Date(dtmCurrentDate.Year, _
77                   dtmCurrentDate.Month, dtmCurrentDate.Day, 12, 0, 0)
78
79             ' seafood will be on tomorrow's noon flight
80             Case 23
81                dtmCurrentDate = dtmCurrentDate.AddDays(1)
82                dtmDepartureTime = New Date(dtmCurrentDate.Year, _
83                   dtmCurrentDate.Month, dtmCurrentDate.Day, 12, 0, 0)
84
85             ' seafood will be on midnight flight
86             Case Else
87                dtmCurrentDate = dtmCurrentDate.AddDays(1)
88                dtmDepartureTime = New Date(dtmCurrentDate.Year, _
89                   dtmCurrentDate.Month, dtmCurrentDate.Day, 0, 0, 0)
90
91          End Select
92
93          Return dtmDepartureTime ' return the flight's departure time
94       End Function ' DepartureTime
95
96    End Class ' FrmShippingTime
```

14.13 (*Alarm Application*) Create an application that allows the user to set an alarm clock. The application should allow the user to set the exact time of the alarm by using a `DateTimePicker`. While the alarm is set, the user should not be able to modify the `DateTimePicker`. If the alarm is set and the current time matches or exceeds the time in the `DateTimePicker`, play the computer's "beep" sound. (Your computer must have the necessary hardware for sound.) The user should be able to cancel an alarm by using a **Reset** `Button`. This `Button` is disabled when the application starts.

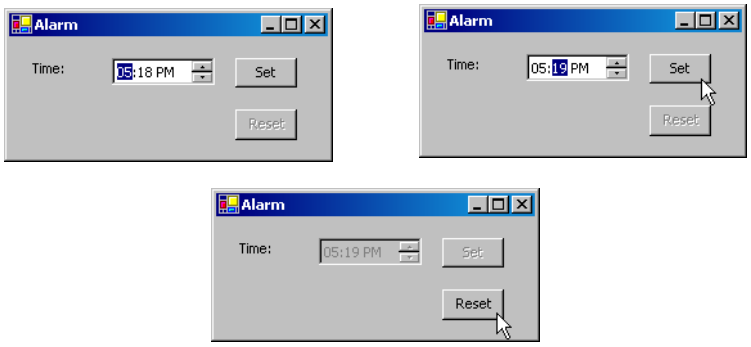


Figure 14.22 Alarm GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial14\Exercises\AlarmClock` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AlarmClock.sln` in the `AlarmClock` directory to open the application.

c) ***Inserting a `DateTimePicker`.*** Add a `DateTimePicker` control to the `Form`. Set the `DateTimePicker` to display only the time, as is shown in Fig. 14.22. Set the `DateTimePicker` control's `Size` property to `80, 20`, and move the control so that it appears as it does in Fig. 14.22.

d) ***Coding the Set Button's Click event handler.*** Add a `Click` event handler for the **Set** `Button`. This event handler should disable the **Set** `Button` and the `DateTimePicker` and enable the **Reset** `Button`.

e) ***Coding the Timer's Tick event handler.*** Define the `Tick` event handler for the `Timer`. A `Tick` event should occur every 1000 milliseconds (one second). If the alarm is set and the current time matches or exceeds the time in the `DateTimePicker`, play the computer's "beep" sound by calling the `Beep` function. To call the `Beep` function, type `Beep()` on its own line in your code.

f) ***Coding the Reset Button's Click event handler.*** Define the `Click` event handler for the **Reset** `Button`. When the **Reset** `Button` is clicked, the GUI should be set back to its original state.

g) ***Running the application.*** Select **Debug > Start** to run your application. Use the `DateTimePicker` and the **Set** `Button` to set a time for the alarm to go off. Wait for that time to verify that the alarm will make beeping sounds. Click the **Reset** `Button` to set a new time for the alarm to go off.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 14.13 Solution
' AlarmClock.vb

Public Class FrmAlarmClock
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' set time for alarm to go off
   Private Sub btnSetTime_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSetTime.Click

      ' disable user input
      btnSetTime.Enabled = False
      dtpAlarmTime.Enabled = False

      btnReset.Enabled = True ' enable reset
   End Sub ' btnSetTime_Click

   ' timer ticks once every minute
   Private Sub tmrTimerAlarm_Tick(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles tmrTimerAlarm.Tick

      ' sound the alarm
      If dtpAlarmTime.Value.Hour = Date.Now.Hour AndAlso _
         dtpAlarmTime.Value.Minute = Date.Now.Minute AndAlso _
         btnReset.Enabled = True Then

         ' call the Beep function
         Beep()
      End If

   End Sub ' tmrTimerAlarm_Tick

   ' return to initial state
   Private Sub btnReset_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnReset.Click

      ' return all GUI controls to initial state
      btnSetTime.Enabled = True
      btnReset.Enabled = False
```

```
        dtpAlarmTime.Enabled = True
    End Sub ' btnReset_Click

End Class ' FrmAlarmClock
```

What does this code do?

14.14 This code creates a `Date` variable. What date does this variable contain?

```
Dim dtmTime As Date = New Date(2003, 1, 2, 3, 4, 5)
```

Answer: This variable contains the date January 2, 2003 at 3:04:05 A.M.

What's wrong with this code?

14.15 The following lines of code are supposed to create a `Date` variable and increment its hour value by two. Find the error(s) in the code.

```
Dim dtmNow As Date = Date.Now
dtmNow.AddHours(2)
```

Answer: Method `AddHours` does not actually increment the `Date` variable, but instead returns a new `Date` variable with the updated value. Thus, the preceding code will not successfully add two hours to `dtmNow`. Correct the code as follows:

```
Dim dtmNow As Date = Date.Now
dtmNow = dtmNow.AddHours(2)
```

Programming Challenge

14.16 (***Parking Garage Fee Calculator***) Create an application that computes the fee for parking a car in a parking garage (Fig. 14.23). The user should provide the **Time In:** and **Time Out:** values by using `DateTimePickers`. The application should calculate the cost of parking in the garage for the specified amount of time. Assume that parking costs three dollars an hour. When calculating the total time spent in the garage, you can ignore the seconds value, but treat the minutes value as a fraction of an hour (1 minute is 1/60 of an hour). For simplicity, assume that no overnight parking is allowed, so each car leaves the garage on the same day in which it arrives.

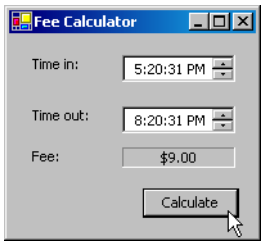


Figure 14.23 Parking Garage **Fee Calculator** GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial14\Exercises\ParkingGarageFeeCalculator` directory to your `C:\SimplyVB` directory.
b) ***Opening the application's template file.*** Double click `ParkingGarageFeeCalculator.sln` in the `ParkingGarageFeeCalculator` directory to open the application
c) ***Inserting the `DateTimePicker` controls.*** Add two `DateTimePicker` controls to the `Form`. Set the `DateTimePickers` so that they show the time only. Set the `Size` property of each `DateTimePicker` control to `80, 20`, and move the `DateTimePickers` so that they are positioned as in Fig. 14.23.
d) ***Writing the `Function` procedure `Fee`.*** Define a `Function` procedure `Fee` that accepts four `Integers` as parameters—the hour value of the **Time In:**, the hour value of the **Time Out:**, the minute value of the **Time In:** and the minute value of the **Time Out:**. Using this information, procedure `Fee` should calculate the fee for parking in the garage. The `Function` procedure should then return this value as a `Decimal`.

e) ***Coding the Calculate `Button`'s `Click` event handler.*** Add the `Click` event handler for the **Calculate** `Button`. This event handler should call `Fee` to obtain the amount due. It should then display the amount (formatted as currency) in a `Label`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Use the `DateTimePickers`' up and down arrows to select a time the car was placed in the garage and the time the car was taken out of the garage. Click the **Calculate** `Button` and verify that the correct fee is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  ' Exercise 14.16 Solution
2  ' ParkingGarageFeeCalculator.vb
3
4  Public Class FrmParkingGarageFeeCalculator
5     Inherits System.Windows.Forms.Form
6
7     ' Windows Form Designer generated code
8
9     ' calculates cost of parking in garage
10    Function Fee(ByVal intTimeOutHour As Integer, _
11        ByVal intTimeInHour As Integer, ByVal intTimeOutMinute _
12        As Integer, ByVal intTimeInMinute As Integer) As Decimal
13
14       ' determines number of elapsed hours
15       Dim intHours As Integer = intTimeOutHour - intTimeInHour
16
17       ' determines number of elapsed minutes
18       Dim intMinutes As Integer = intTimeOutMinute - intTimeInMinute
19
20       If intMinutes < 0 Then
21          intHours -= 1
22          intMinutes += 60
23       End If
24
25       Return (intHours + (intMinutes / 60)) * 3
26    End Function ' Fee
27
28    ' called when Calculate Button is clicked
29    Private Sub btnCalculate_Click(ByVal sender As System.Object, _
30       ByVal e As System.EventArgs) Handles btnCalculate.Click
31
32       Dim intFee As Decimal = 0
33
34       ' calls procedure Fee
35       intFee = Fee(dtpTimeOut.Value.Hour, _
36          dtpTimeIn.Value.Hour, dtpTimeOut.Value.Minute, _
37          dtpTimeIn.Value.Minute)
38
39       ' output fee as currency
40       lblFeeResult.Text = String.Format("{0:C}", intFee)
41    End Sub ' btnCalculate_Click
42
43 End Class ' FrmParkingGarageFeeCalculator
```

TUTORIAL 15

Fund Raiser Application

Introducing Scope, Pass-by-Reference and `Option Strict`

Solutions

Instructor's Manual Exercise Solutions Tutorial 15

MULTIPLE-CHOICE QUESTIONS

15.1 In the **Property Pages** dialog, _____ must be selected to access `Option Strict`.

a) **Build** b) **Designer Defaults**
c) **General** d) **Imports**

15.2 When `Option Strict` is set to `On`, variables _____.

a) are passed by value b) are passed by reference
c) might need to be converted explicitly to a different type to avoid errors
d) are used only within the block in which the variables are declared

15.3 A variable declared inside a class, but outside a procedure, is called a(n) _____.

a) local variable b) hidden variable
c) instance variable d) constant variable

15.4 Visual Basic .NET provides methods in class _____ to convert from one data type to another.

a) `ChangeTo` b) `Convert`
c) `ConvertTo` d) `ChangeType`

15.5 When `Option Strict` is _____, the conversion attempt `intX = dblPercent` results in an error.

a) `On` b) `True`
c) `Off` d) `False`

15.6 Keyword _____ indicates pass-by-reference.

a) `ByReference` b) `ByRef`
c) `Ref` d) `Reference`

15.7 With _____, changes made to a parameter variable's value do not affect the value of the variable in the calling procedure.

a) `Option Strict` b) pass-by-value
c) pass-by-reference d) None of the above.

15.8 Instance variables _____.

a) are members of class b) are prefixed by `m_`
c) can be accessed by a procedure in the same class
d) All of the above.

15.9 Assigning a "smaller" type to a "larger" type is a _____ conversion.

a) narrowing b) shortening
c) widening d) lengthening

15.10 A value of type `Boolean` can be implicitly converted to _____.

a) `Integer` b) `String`
c) `Object` d) `Double`

Answers: 15.1) a. 15.2) c. 15.3) c. 15.4) b. 15.5) a. 15.6) b. 15.7) b. 15.8) d. 15.9) c. 15.10) c.

EXERCISES

15.11 (***Task List Application***) Create an application that allows users to add items to a daily task list. The application should also display the number of tasks to be performed. Use method `Convert.ToString` to display the number of tasks in a `Label`. The application should look like the GUI in Fig. 15.29.

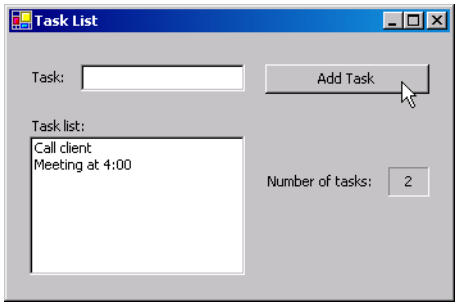


Figure 15.29 Task List application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial15\Exercises\TaskList` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `TaskList.sln` in the `TaskList` directory to open the application.

c) ***Setting `Option Strict` to `On`.*** Use the directions provided in the box, *Enabling Option Strict*, to set `Option Strict` to `On`.

d) ***Adding the Add Task `Button`'s `Click` event handler.*** Double click the **Add Task** `Button` to generate the empty event handler `btnAdd_Click`. This event handler should display the user input in the `ListBox` and clear the user input from the `TextBox`. The event handler should also update the `Label` that displays the number of tasks. Use method `Convert.ToString` to display the number of tasks in the `Label`. Finally, the event handler should transfer the focus to the `TextBox`.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter several tasks, click the **Add Task** `Button` after each. Verify that each task is added to the **Task list:** `ListBox`, and that the number of tasks is incremented with each new task.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 15.11 Solution
' TaskList.vb

Public Class FrmTaskList
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles Add Task Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      lstTasks.Items.Add(txtTask.Text)  ' insert task into ListBox

      txtTask.Clear()  ' clear TextBox of user input

      ' convert Integer to String to display number of tasks
      lblOutput.Text = Convert.ToString(lstTasks.Items.Count)

      txtTask.Focus() ' transfer the focus to the TextBox
   End Sub ' btnAdd_Click

End Class ' FrmTaskList
```

15.12 (*Quiz Average Application*) Develop an application that computes a student's average quiz score for all of the quiz scores entered. The application should look like the GUI in Fig. 15.30. Use method `Convert.ToInt32` to convert the user input to an `Integer`. Use

instance variables with module scope to keep track of the sum of all the quiz scores entered and the number of quiz scores entered.

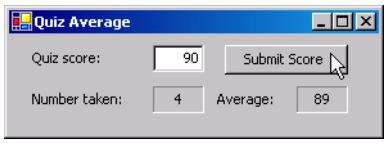


Figure 15.30 Quiz Average application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial15\Exercises\QuizAverage` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `QuizAverage.sln` in the `QuizAverage` directory to open the application.

c) ***Setting `Option Strict` to `On`.*** Use the directions provided in the box, *Enabling Option Strict*, to set `Option Strict` to `On`.

d) ***Adding instance variables.*** Add two instance variables—`m_intTotalScore`, which keeps track of the sum of all the quiz scores entered, and `m_intTaken`, which keeps track of the number of quiz scores entered.

e) ***Adding the Grade Quiz `Button`'s event handler.*** Double click the **Submit Score** `Button` to generate the empty event handler `btnCalculate_Click`. The code required in *Steps f–k* should be placed in this event handler.

f) ***Obtaining user input.*** Use method `Convert.ToInt32` to convert the user input from the `TextBox` to an `Integer`.

g) ***Updating the number of quiz scores entered.*** Increment the number of quiz scores entered.

h) ***Updating the sum of all the quiz scores entered.*** Add the current quiz score to the current total to update the sum of all the quiz scores entered.

i) ***Calculating the average score.*** Divide the sum of all the quiz scores entered by the number of quiz scores entered to calculate the average score.

j) ***Displaying the average score.*** Use method `Convert.ToString` to display the average quiz grade in the **Average:** field.

k) ***Displaying the number of quizzes taken.*** Use method `Convert.ToString` to display the number of quiz scores entered in the **Number taken:** field.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter several quiz scores, clicking the **Submit Score** `Button` after each. With each new score, verify that the **Number taken:** field is incremented and that the average is updated correctly.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 15.12 Solution
' QuizAverage.vb

Public Class FrmQuizAverage
   Inherits System.Windows.Forms.Form

   ' instance variables store total score and number quizzes taken
   Dim m_intTotalScore As Integer = 0
   Dim m_intTaken As Integer = 0

   ' Windows Form Designer generated code

   ' handles Submit Score Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
```

```
        ByVal e As System.EventArgs) Handles btnCalculate.Click

        Dim intScore As Integer
        Dim intAverage As Integer

        ' obtain and convert user input
        intScore = Convert.ToInt32(Val(txtScore.Text))

        ' update number of quizzes taken
        m_intTaken += 1

        ' update total score
        m_intTotalScore += intScore

        ' calculate average score
        intAverage = m_intTotalScore \ m_intTaken

        ' display average score
        lblAverage.Text = Convert.ToString(intAverage)

        ' display number of quizzes taken
        lblTaken.Text = Convert.ToString(m_intTaken)

    End Sub ' btnCalculate_Click

End Class ' FrmQuizAverage
```

15.13 (*Maximum Application*) Modify the **Maximum** application from Chapter 13 (Fig. 15.31) to use keyword `ByRef` to pass a fourth argument to procedure `Maximum` by reference. Also, use methods from class `Convert` to perform any necessary type conversions.

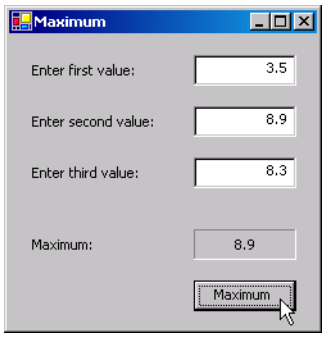


Figure 15.31 **Maximum** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial15\Exercises\Maximum` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Maximum.sln` in the `Maximum` directory to open the application.

c) ***Setting `Option Strict` to `On`.*** Use the directions provided in the box, *Enabling Option Strict*, to set `Option Strict` to `On`.

d) ***Adding a local variable.*** Add local variable `dblMaximum` of type `Double` to event handler `btnMaximum_Click`. The code required in *Steps d–f* should be placed in this event handler. Variable `dblMaximum` will store the result of procedure `Maximum`.

e) ***Passing four arguments to procedure `Maximum`.*** Use method `Convert.ToDouble` to convert the user input from the `TextBoxes` to `Doubles`. Pass these three values as the first three arguments to procedure `Maximum`. Pass local variable `dblMaximum` as the fourth argument to procedure `Maximum`.

f) ***Displaying the maximum value.*** Use method `Convert.ToString` to display local variable `dblMaximum` in the **Maximum:** field.

g) ***Changing procedure `Maximum` to a `Sub` procedure.*** Change procedure `Maximum` to a `Sub` procedure. Make sure that `Sub` procedure `Maximum` no longer returns a value and does not specify a return type. The modifications required in *Steps g–h* should be performed on this `Sub` procedure.

h) ***Adding a fourth parameter to procedure `Maximum`.*** Add a fourth parameter `dblFinalMaximum` of type `Double` to `Maximum`'s procedure header. Use keyword `ByRef` to specify that this argument will be passed by reference. Remove the declaration of variable `dblFinalMaximum` from the body of procedure `Maximum`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter three different values into the input fields and click the **Maximum** `Button`. Verify that the largest value is displayed in the **Maximum:** field.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 15.13 Solution
' Maximum.vb

Public Class FrmMaximum
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' obtain values in each TextBox, call procedure Maximum
   Private Sub btnMaximum_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnMaximum.Click

      Dim dblMaximum As Double

      Maximum(Convert.ToDouble(Val(txtFirst.Text)), _
         Convert.ToDouble(Val(txtSecond.Text)), _
         Convert.ToDouble(Val(txtThird.Text)), dblMaximum)

      lblOutput.Text = Convert.ToString(dblMaximum)
   End Sub ' btnMaximum_Click

   ' find maximum of three parameter values
   Sub Maximum(ByVal dblOne As Double, ByVal dblTwo _
      As Double, ByVal dblThree As Double, _
      ByRef dblFinalMaximum As Double)

      Dim dblTemporaryMaximum As Double

      dblTemporaryMaximum = Math.Max(dblOne, dblTwo)
      dblFinalMaximum = Math.Max(dblTemporaryMaximum, dblThree)
   End Sub ' Maximum

End Class ' FrmMaximum
```

What does this code do? ▶

15.14 What is displayed in `Label` `lblDisplay` when the following code is executed?

```
Public Class FrmScopeTest
   Inherits System.Windows.Forms.Form

   Dim intValue2 As Integer = 5

   Private Sub btnEnter_Click(ByVal sender As System.Object, _
```

```
        ByVal e As System.EventArgs) Handles btnEnter.Click

        Dim intValue1 As Integer = 10
        Dim intValue2 As Integer = 3

        Test(intValue1)
        lblDisplay.Text = Convert.ToString(intValue1)
    End Sub ' btnEnter_Click

    Sub Test(ByRef intValue1 As Integer)
        intValue1 *= intValue2
    End Sub ' Test

End Class ' FrmScopeTest
```

Answer: `Label lblDisplay` displays the value of variable `intValue1` (50). When the code invokes `Sub` procedure `Test`, it passes `intValue1` pass-by-reference. Any changes made to `intValue1` in `Sub` procedure `Test` are reflected in `btnEnter_Click`'s local variable `intValue1`. When `Sub` procedure `Test` multiplies `intValue1` by `intValue2`, `intValue2` is the class instance variable, whose value is 5. Procedure `Test` does not have access to `btnEnter_Click`'s local variable `intValue2`.

What's wrong with this code?

15.15 Find the error(s) in the following code (the procedure should assign the value 14 to variable `intResult`).

```
Sub Sum()
  Dim strNumber As String = "4"
  Dim intNumber As Integer = 10
  Dim intResult As Integer

  intResult = strNumber + intNumber
End Sub ' Sum
```

Answer: The code must explicitly convert `strNumber` to `Integer`:

```
Sub Sum()
  Dim strNumber As String = "4"
  Dim intNumber As Integer = 10
  Dim intResult As Integer

  intResult = Convert.ToInt32(strNumber) + intNumber
End Sub ' Sum
```

Programming Challenge

15.16 (***Schedule Book Application***) Develop an application that allows a user to enter a schedule of appointments and their respective times. Create the `Form` in Fig. 15.32 and name the application **Schedule Book**. Add a `Function` procedure called `TimeTaken` that returns a `Boolean` value. Each time a user enters a new appointment, `Function` procedure `TimeTaken` determines if the user has scheduled more than one appointment at the same time. If `TimeTaken` returns `True`, the user will be notified via a message dialog. Otherwise, the appointment should be added to the `ListBoxes`. Set `Option Strict` to `On` and use methods from class `Convert` as necessary.

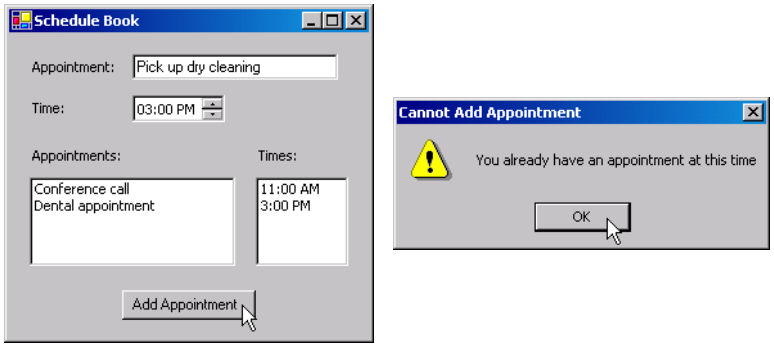


Figure 15.32 **Schedule Book** application's GUI.

```
' Exercise 15.16 Solution
' ScheduleBook.vb

Public Class FrmScheduleBook
   Inherits System.Windows.Forms.Form

   ' Windows Form designer generated code

   ' handles Add Appointment Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' appointment scheduled for given time
      Dim blnTimeTaken As Boolean = TimeTaken()

      ' display message if appointment conflict
      If blnTimeTaken = True Then

         MessageBox.Show("You already have an appointment " & _
             "at this time", "Cannot Add Appointment", _
             MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

      ' otherwise add appointment and time to ListBoxes
      Else
         lstAppointments.Items.Add(txtAppointment.Text)
         lstTimes.Items.Add(dtpTime.Value.ToShortTimeString())
      End If

      ' clear user input from TextBoxes
      txtAppointment.Clear()
   End Sub ' btnAdd_Click

   ' determines if an appointment already exists at specified time
   Function TimeTaken() As Boolean

      ' determine number of appointments
      Dim intItems As Integer = lstTimes.Items.Count()

      ' determines if items are in time ListBox
      If intItems <> 0 Then

         Dim intCounter As Integer

         ' search ListBox to determine if an appointment
         ' has been made for that time
         For intCounter = 0 To intItems - 1
```

```
47
48              ' compare times in ListBox with user entry
49              If Convert.ToString(lstTimes.Items.Item( _
50                 intCounter)) = dtpTime.Value.ToShortTimeString() Then
51
52                 Return True
53              End If
54
55           Next
56
57        End If
58
59        Return False
60
61     End Function ' TimeTaken
62
63  End Class ' FrmScheduleBook
```

TUTORIAL 16

Craps Game Application

Introducing Random-Number Generation

Solutions

Instructor's Manual Exercise Solutions Tutorial 16

MULTIPLE-CHOICE QUESTIONS

16.1 A `Random` object can generate pseudorandom numbers of type _____.

a) `Integer` b) `Single`
c) `Double` d) Both a and c.

16.2 A _____ is a group of related classes in the Framework Class Library.

a) classpace b) directory
c) namespace d) library

16.3 Object variable names should be prefixed with _____.

a) `var` b) `obj`
c) `ran` d) `ojt`

16.4 The `Next` method of class `Random` can be called using _____.

a) one argument b) no arguments
c) two arguments d) All of the above.

16.5 The statement _____ assigns `intValue` a random number in the range from 5 to 20.

a) `intValue = objRandom.Next(5, 21)` b) `intValue = objRandom.Next(4, 20)`
c) `intValue = objRandom.Next(5, 20)` d) `intValue = objRandom.Next(4, 21)`

16.6 The _____ method specifies the file from which an image is loaded.

a) `Next` in class `Random` b) `FromFile` in class `Image`
c) `GetCurrentDirectory` in class `Directory` d) None of the above.

16.7 The `System.IO` namespace contains classes and methods to _____.

a) access files and directories b) display graphics in an application
c) insert multimedia into an application d) All of the above.

16.8 The values returned by the _____ method of class `Random` are actually pseudo-random numbers.

a) `NextRandom` b) `PseudoRandom`
c) `Next` d) `Pseudo`

16.9 When creating random numbers, the second argument passed to the `Next` method is _____.

a) equal to the maximum value you wish to be generated
b) equal to one more than the maximum value you wish to be generated
c) equal to one less than the maximum value you wish to be generated
d) equal to the minimum value you wish to be generated

16.10 A(n) _____ is a group of related, named constants.

a) namespace b) variable
c) enumeration d) None of the above.

Answers: 16.1) d. 16.2) c. 16.3) b. 16.4) d. 16.5) a. 16.6) b. 16.7) a. 16.8) c. 16.9) b. 16.10) c.

EXERCISES

16.11 (*Guess the Number Application*) Develop an application that generates a random number and prompts the user to guess the number (Fig. 16.21). When the user clicks the **New Game** `Button`, the application chooses a number in the range 1 to 100 at random. The user

enters guesses into the **Guess:** `TextBox` and clicks the **Enter** `Button`. If the guess is correct, the game ends, and the user can start a new game. If the guess is not correct, the application should indicate if the guess is higher or lower than the correct number.

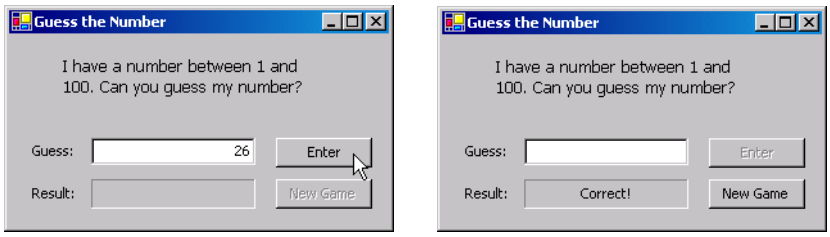


Figure 16.21 Guess the Number application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial16\Exercises\GuessNumber` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `GuessNumber.sln` in the `GuessNumber` directory to open the application (Fig. 16.21).

c) ***Creating a `Random` object.*** Create two instance variables. The first variable should store a `Random` object and the second variable should store a random-generated number.

d) ***Adding a `Click` event handler for the Enter `Button`.*** Add a `Click` event handler for the **Enter** `Button` that retrieves the value entered by the user and compares that value to the random-generated number. If the guess is correct, display **Correct!** in the output `Label`. Then disable the **Enter** `Button`, and enable the **New Game** `Button`. If the user's guess is higher than the correct answer, display **Too high...** in the output `Label`. If the user's guess is lower than the correct answer, display **Too low...** in the output `Label`.

e) ***Adding a `Click` event handler for the New Game `Button`.*** Add a `Click` event handler for the **New Game** `Button` that generates a new random number for the instance variable. The event handler should then disable the **New Game** `Button`, enable the **Enter** `Button` and clear the **Result:** `TextBox`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter guesses (clicking the **Enter** `Button` after each) until you have successfully determined the answer. Click the **New Game** `Button` and test the application again.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 16.11 Solution
' GuessNumber.vb

Public Class FrmGuessNumber
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Dim m_objRandom As Random = New Random
   Dim m_intNumber As Integer = m_objRandom.Next(1, 101)

   ' handles Enter button click event
   Private Sub btnEnter_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEnter.Click

      ' check answer
      If Val(txtGuessNumber.Text) = m_intNumber Then
         lblOutput.Text = "Correct!"
         btnEnter.Enabled = False
```

```
            btnNewGame.Enabled = True
         ElseIf Val(txtGuessNumber.Text) > m_intNumber Then
            lblOutput.Text = "Too high..."
         Else
            lblOutput.Text = "Too low..."
         End If

         txtGuessNumber.Clear()
         txtGuessNumber.Focus()
      End Sub ' btnEnter_Click

      ' restart game with new number
      Private Sub btnNewGame_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnNewGame.Click

         ' generate new random number
         m_intNumber = m_objRandom.Next(1, 101)
         btnEnter.Enabled = True
         btnNewGame.Enabled = False
         lblOutput.Text = "" ' clear result
      End Sub ' btnNewGame_Click

   End Class ' FrmGuessNumber
```

16.12 (*Dice Simulator Application*) Develop an application that simulates rolling two six-sided dice. Your application should have a **Roll** `Button` that, when clicked, displays two dice images corresponding to random numbers. It should also display the number of times each face has appeared. Your application should appear similar to Fig. 16.22.

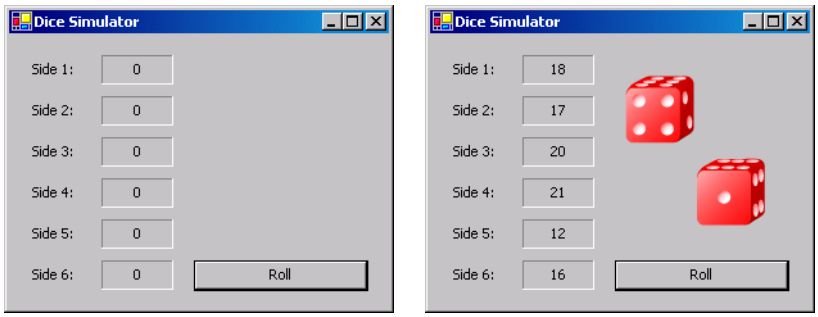


Figure 16.22 **Dice Simulator** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial16\Exercises\DiceSimulator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `DiceSimulator.sln` in the `DiceSimulator` directory to open the application.

c) ***Displaying the die image.*** Create a `Sub` procedure named `DisplayDie` that takes a `PictureBox` control as an argument. This method should generate a random number to simulate a die roll. Then display the die image in the corresponding `PictureBox` control on the `Form`. The die image should correspond to the random number that was generated. To set the image, refer to the code presented in Fig. 16.20.

d) ***Adding a `Click` event handler for the Roll `Button`.*** Add a `Click` event handler for the **Roll** `Button`. Call method `DisplayDie` in this event handler to display the images for both dice.

e) ***Displaying the frequency.*** Add a `Sub` procedure called `DisplayFrequency` that uses a `Select Case` statement to update the number of times each face has appeared. Create an enumeration for the dice faces which will be used in the `Select Case` statement.

f) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Roll** `Button` several times. Each time, two die faces should be displayed at random. Verify after each roll that the appropriate face values on the left are incremented.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 16.12 Solution
' DiceSimulator.vb

Imports System.IO

Public Class FrmDiceSimulator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' die face constants
   Enum FaceNames
      ONE = 1
      TWO = 2
      THREE = 3
      FOUR = 4
      FIVE = 5
      SIX = 6
   End Enum

   ' declare Random object reference
   Dim m_objRandomNumber As Random = New Random

   ' display results of roll
   Private Sub btnRoll_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnRoll.Click

      ' method randomly assigns a face to each die
      DisplayDie(picDie1)
      DisplayDie(picDie2)
   End Sub ' btnRoll_Click

   ' get a random die image
   Sub DisplayDie(ByVal picDie As PictureBox)

      ' generate random integer in range 1 to 6
      Dim intFace As Integer = m_objRandomNumber.Next(1, 7)

      ' load corresponding image
      picDie.Image = Image.FromFile( _
         Directory.GetCurrentDirectory & "/Images/die" & _
         intFace & ".png")

      ' method displays rolls frequency
      DisplayFrequency(intFace)
   End Sub ' DisplayDie

   ' display the rolls frequency
   Sub DisplayFrequency(ByVal intFace As Integer)

      ' increment and display frequency values
      Select Case intFace

```

```
            Case FaceNames.ONE
               lblOutput1.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput1.Text) + 1)

            Case FaceNames.TWO
               lblOutput2.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput2.Text) + 1)

            Case FaceNames.THREE
               lblOutput3.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput3.Text) + 1)

            Case FaceNames.FOUR
               lblOutput4.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput4.Text) + 1)

            Case FaceNames.FIVE
               lblOutput5.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput5.Text) + 1)

            Case FaceNames.SIX
               lblOutput6.Text = _
                  Convert.ToString(Convert.ToInt32(lblOutput6.Text) + 1)

         End Select

      End Sub ' DisplayFrequency

   End Class ' FrmDiceSimulator
```

16.13 (***Lottery Numbers Picker Application***) A lottery commission offers four different lottery games to play: Three number, Four number, Five number and Five number + 1 lotteries. Each game has independent numbers. Develop an application that randomly picks numbers for all four games and displays the generated numbers in a GUI (Fig. 16.23). The games are played as follows

- Three-number lotteries require players to choose three numbers in the range of 0–9.
- Four-number lotteries require players to choose four numbers, in the range of 0–9.
- Five-number lotteries require players to choose five numbers in the range of 1–39.
- Five-number + 1 lotteries require players to choose five numbers in the range of 1–49 and an additional number in the range of 1–42.

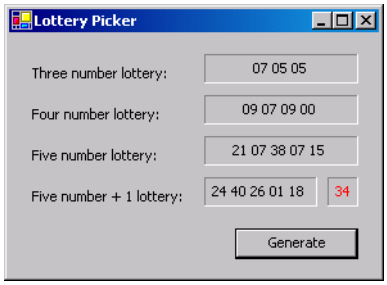


Figure 16.23 Lottery Picker application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial16\Exercises\LotteryPicker` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `LotteryPicker.sln` in the `LotteryPicker` directory to open the application.

c) ***Generating random numbers.*** Create a `Function` procedure that will generate the random numbers for all four games.

d) ***Drawing numbers for the games.*** Add code into your application to generate numbers for all four games. To make the applications simple, allow repetition of numbers.

e) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Generate** `Button` multiple times. Make sure the values displayed are within the ranges described in the exercise description.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 16.13 Solution
' LotteryPicker.vb

Public Class FrmLotteryPicker
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Public m_objRandom As Random = New Random

   ' display random lottery numbers
   Private Sub btnGenerate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnGenerate.Click

      ' generate three numbers
      lblOutput3.Text = Generate(0, 10) & " " & _
         Generate(0, 10) & " " & Generate(0, 10)

      ' generate four numbers
      lblOutput4.Text = Generate(0, 10) & " " & _
         Generate(0, 10) & " " & Generate(0, 10) & " " _
         & Generate(0, 10)

      ' generate five numbers
      lblOutput5.Text = Generate(1, 40) & " " & _
         Generate(1, 40) & " " & Generate(1, 40) & _
         " " & Generate(1, 40) & " " & Generate(1, 40)

      ' generate five plus one numbers
      lblOutput5Plus1.Text = Generate(1, 50) & " " & _
         Generate(1, 50) & " " & Generate(1, 50) & " " & _
         Generate(1, 50) & " " & Generate(1, 50)

      lblOutputExtra1.Text = Generate(1, 43)

   End Sub ' btnGenerate_Click

   ' generate random numbers
   Function Generate(ByVal intLow As Integer, _
      ByVal intHigh As Integer) As String

      Return String.Format("{0:D2}", _
         m_objRandom.Next(intLow, intHigh))

   End Function ' Generate
```

```
End Class ' FrmLotteryPicker
```

Answer:

What does this code do?

16.14 What does the following code do?

```
Sub PickRandomNumbers()

   Dim intNumber1 As Integer
   Dim dblNumber As Double
   Dim intNumber2 As Integer
   Dim objRandom As Random = New Random

   intNumber1 = objRandom.Next()
   dblNumber = 5 * objRandom.NextDouble()
   intNumber2 = objRandom.Next(1, 10)
   lblInteger1.Text = Convert.ToString(intNumber1)
   lblDouble1.Text = Convert.ToString(dblNumber)
   lblInteger2.Text = Convert.ToString(intNumber2)
End Sub ' PickRandomNumbers
```

Answer: `intNumber1` gets a positive integer (between 0 and `Int32.MaxValue`), `dblNumber` gets a floating-point number between 0 and 5 (not including 5) and `intNumber2` gets an integer between 1 and 10 (not including 10).

What's wrong with this code?

16.15 This `Sub` procedure should assign a random `Decimal` number (in the range 0 to `Int32.MaxValue`) to `Decimal decNumber`. (Assume that `Option Strict` is `On`.) Find the error(s) in the following code.

```
Sub RandomDecimal()

   Dim decNumber As Decimal
   Dim objRandom As Random = New Random

   decNumber = objRandom.Next()
   lblDisplay.Text = Convert.ToString(decNumber)

End Sub ' RandomDecimal
```

Answers: `Random` objects can produce `Integer`s and `Double`s only; this will yield only an `Integer` random number. [There is no way to have the `Random` class generate a `Decimal` random value, so the solution is a bit contrived. The closest approximation is to use `NextDouble`.]

```
Sub RandomDecimal()

   Dim decNumber As Decimal
   Dim objRandom As Random = New Random

   decNumber = Int32.MaxValue * _
      Convert.ToDecimal(objRandom.NextDouble())
   lblDisplay.Text = Convert.ToString(decNumber)

End Sub ' RandomDecimal
```

Programming Challenge

16.16 (*Multiplication Teacher Application*) Develop an application that helps children learn multiplication. Use random-number generation to produce two positive one-digit integers that display in a question, such as "How much is 6 times 7?" The student should type the answer into a `TextBox`. If the answer is correct, then the application randomly displays one of three messages: **Very Good!**, **Excellent!** or **Great Job!** in a `Label` and displays the next question. If the student is wrong, the `Label` displays the message **No. Please try again**.

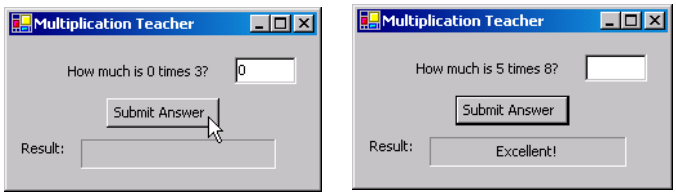


Figure 16.24 Multiplication Teacher application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial16\Exercises\MultiplicationTeacher` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MultiplicationTeacher.sln` in the `MultiplicationTeacher` directory to open the application.

c) ***Generating the questions.*** Add a method into your application (Fig. 16.24) to generate each new question.

d) ***Determining whether the right answer was entered.*** Add code into your application to call the method created in the previous step. After this method has been called, determine whether the student answered the question correctly, and display the appropriate message.

e) ***Displaying a random message.*** Add a procedure `GenerateOutput` that displays a random message congratulating the student for answering correctly. This method should be called if the student answered the question correctly.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter several correct answers and at least one incorrect answer. Verify that **No. Please try again** is displayed when you are incorrect, and one of the other responses is displayed at random when you are correct.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 16.16 Solution
' MultiplicationTeacher.vb

Public Class FrmMultiplicationTeacher
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' create new random object
   Dim m_objRandomObject As Random = New Random

   ' random numbers for questions
   Dim m_intRandomNumber1 As Integer
   Dim m_intRandomNumber2 As Integer
   Dim m_strQuestion As String ' String for the question being asked
   Dim m_intCorrectAnswer As Integer ' correct answer to question

   ' handles load event for FrmMultiplicationTeacher
   Private Sub FrmMultiplicationTeacher_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
```

```
      Handles MyBase.Load

      GenerateQuestion() ' generate a question
   End Sub ' FrmMultiplicationTeacher_Load

   ' handles click event for btnSubmit Button
   Private Sub btnSubmit_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSubmit.Click

      ' retrieve user's answer
      Dim intAnswer As Integer = Convert.ToInt32(Val(txtAnswer.Text))

      txtAnswer.Clear() ' clear the TextBox

      ' check if user answer is correct
      If intAnswer = m_intCorrectAnswer Then
         GenerateOutput() ' display correct message
         GenerateQuestion() ' create another question

      Else ' answer was wrong, try again
         lblResponse.Text = "No. Please try again."
      End If

   End Sub ' btnSubmit_Click

   ' generates a new question
   Sub GenerateQuestion()

      ' create two random numbers
      m_intRandomNumber1 = m_objRandomObject.Next(0, 10)
      m_intRandomNumber2 = m_objRandomObject.Next(0, 10)

      ' record the correct answer
      m_intCorrectAnswer = m_intRandomNumber1 * m_intRandomNumber2

      ' construct the question
      m_strQuestion = "How much is " & _
         m_intRandomNumber1.ToString() & " times " & _
         m_intRandomNumber2.ToString() & "?"

      ' display the question
      lblQuestion.Text = m_strQuestion
   End Sub ' GenerateQuestion

   ' generates correct message
   Sub GenerateOutput()
      Dim intNumber As Integer = m_objRandomObject.Next(0, 3)

      ' show random message
      Select Case intNumber

         Case 0
            lblResponse.Text = "Very Good!"

         Case 1
            lblResponse.Text = "Excellent!"

         Case 2
            lblResponse.Text = "Great Job!"

```

```
        End Select

    End Sub ' GenerateOutput

End Class ' FrmMultiplicationTeacher
```

TUTORIAL 17

Flag Quiz Application

Introducing One-Dimensional Arrays and ComboBoxes

Solutions

Instructor's Manual Exercise Solutions Tutorial 17

MULTIPLE-CHOICE QUESTIONS

17.1 Arrays can be declared to hold values of _____.

a) type `Double` b) type `Integer`
c) type `String` d) any data type

17.2 The elements of an array are related by the fact that they have the same name and _____.

a) constant value b) subscript
c) type d) value

17.3 Method _____ returns the largest index in the array.

a) `GetUpperBound` b) `GetUpperLimit`
c) `GetLargestIndex` d) `GetUpperSubscript`

17.4 The first element in every array is the _____.

a) subscript b) zeroth element
c) length of the array d) smallest value in the array

17.5 Arrays _____.

a) are controls b) always have one dimension
c) keep data in sorted order at all times d) are objects

17.6 The initializer list can _____.

a) be used to determine the size of the array
b) contain a comma-separated list of initial values for the array elements
c) be empty d) All of the above.

17.7 Which method call sorts array `strWords` in ascending order?

a) `Array.Sort(strWords)` b) `strWords.SortArray()`
c) `Array.Sort(strWords, 1)` d) `Sort(strWords)`

17.8 The `ComboBox` control combines a `TextBox` with a _____ control.

a) `DateTimePicker` b) `ListBox`
c) `NumericUpDown` d) `Label`

17.9 To search for a period (`.`) in a `String` called `strTest`, call method _____.

a) `String.Search(strTest, ".")` b) `String.IndexOf(strTest, ".")`
c) `strTest.IndexOf(".")` d) `strTest.Search(".")`

17.10 Property _____ contains the size of an array.

a) `Elements` b) `ArraySize`
c) `Length` d) `Size`

Answers: 17.1) d. 17.2) c. 17.3) a. 17.4) b. 17.5) d. 17.6) d. 17.7) a. 17.8) b. 17.9) c. 17.10) c.

EXERCISES

17.11 ***(Enhanced Flag Quiz Application)*** Enhance the **Flag Quiz** application by counting the number of questions that were answered correctly (Fig. 17.31). After all the questions have been answered, display a message in a `Label` that describes how well the user performed. The following table shows which messages to display:

Number of correct answers	Message
5	`Excellent!`

4	Very good
3	Good
2	Poor
1 or 0	Fail

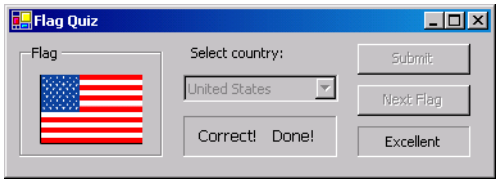


Figure 17.31 Enhanced **Flag Quiz** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial17\Exercises\FlagQuiz2` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `FlagQuiz.sln` in the `FlagQuiz2` directory to open the application.

c) ***Adding a variable to count the number of correct answers.*** Add an instance variable `m_intNumberCorrect`, and initialize it to `0`. You will use this variable to count the number of correct answers submitted by the user.

d) ***Counting the correct answers.*** Increment `m_intNumberCorrect` in the **Submit** `Button`'s event handler whenever the submitted answer is correct.

e) ***Displaying the message.*** Write a procedure `DisplayMessage` that displays a message in `lblScore` depending on the value of `m_intNumberCorrect`. Call this procedure from the **Submit** `Button`'s event handler when the quiz is completed.

f) ***Running the application.*** Select **Debug > Start** to run your application. The finished application should behave as in Fig. 17.31. Run the application a few times, enter a different number of correct answers each time to verify that the correct feedback is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 17.11 Solution
' FlagQuiz.vb

Public Class FrmFlagQuiz
   Inherits System.Windows.Forms.Form

   ' String array stores country names
   Dim m_strOptions As String() = New String() { _
      "Russia", "China", "United States", "Italy", _
      "Australia", "South Africa", "Brazil", "Spain"}

   ' Boolean array tracks displayed flags
   Dim m_blnUsed As Boolean() = _
      New Boolean(m_strOptions.GetUpperBound(0)) {}

   Dim m_intCount As Integer = 1  ' number of flags shown
   Dim m_strCountry As String     ' current flag's country

   Dim m_intNumberCorrect As Integer = 0

   ' Windows Form Designer generated code

   ' handles Flag Quiz Form's Load event
```

```
   Private Sub FrmFlagQuiz_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      Array.Sort(m_strOptions)  ' alphabetize country names

      ' display country names in ComboBox
      cboOptions.DataSource = m_strOptions

      DisplayFlag()  ' display first flag in PictureBox
   End Sub ' FrmFlagQuiz_Load

   ' return full path name of image file as a String
   Function BuildPathName() As String

      ' begin with country name
      Dim strOutput As String = m_strCountry

      ' locate space character if there is one
      Dim intSpace As Integer = strOutput.IndexOf(" ")

      ' remove space from country name if there is one
      If intSpace > 0 Then
         strOutput = strOutput.Remove(intSpace, 1)
      End If

      strOutput = strOutput.ToLower() ' make characters lowercase
      strOutput &= ".png" ' add file extension

      ' add path name
      strOutput = strOutput.Insert(0, _
         System.Environment.CurrentDirectory & "\images\")

      Return strOutput  ' return full path name
   End Function ' BuildPathName

   ' return an unused random number
   Function GetUniqueRandomNumber() As Integer

      Dim objRandom As Random = New Random()
      Dim intRandom As Integer

      ' generate random numbers until unused flag is found
      Do
         intRandom = objRandom.Next(0, m_blnUsed.Length)
      Loop Until m_blnUsed(intRandom) = False

      ' indicate that flag has been used
      m_blnUsed(intRandom) = True

      Return intRandom  ' return index for new flag
   End Function ' GetUniqueRandomNumber

   ' display random flag in PictureBox
   Sub DisplayFlag()

      ' unique index ensures that a flag is used no more than once
      Dim intRandom As Integer = GetUniqueRandomNumber()

      ' retrieve country name from array m_strOptions
      m_strCountry = m_strOptions(intRandom)

```

```
         ' get image's full path name
         Dim strPath As String = BuildPathName()
         picFlag.Image = Image.FromFile(strPath) ' display image
      End Sub ' DisplayFlag

      ' handles Submit Button's Click event
      Private Sub btnSubmit_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnSubmit.Click

         ' retrieve answer from ComboBox
         Dim strResponse As String = _
            Convert.ToString(cboOptions.SelectedValue)

         ' verify answer
         If strResponse = m_strCountry Then
            lblFeedback.Text = "Correct!"
            m_intNumberCorrect += 1  ' update correct answers counter
         Else
            lblFeedback.Text = "Sorry, incorrect."
         End If

         ' inform user if quiz is over
         If m_intCount >= 5 Then  ' quiz is over
            lblFeedback.Text &= "   Done!"
            btnNext.Enabled = False
            btnSubmit.Enabled = False
            cboOptions.Enabled = False

            DisplayScore()
         Else  ' quiz is not over
            btnSubmit.Enabled = False
            btnNext.Enabled = True
         End If

      End Sub ' btnSubmit_Click

      ' handles Next Flag Button's Click event
      Private Sub btnNext_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnNext.Click

         DisplayFlag()  ' display next flag
         lblFeedback.Text = ""  ' clear output

         ' change selected country to first in ComboBox
         cboOptions.SelectedIndex = 0

         m_intCount += 1  ' update number of flags shown

         btnSubmit.Enabled = True
         btnNext.Enabled = False
      End Sub ' btnNext_Click

      ' displays message about number of correct answers
      Sub DisplayScore()

         Select Case m_intNumberCorrect

            Case 5
               lblScore.Text = "Excellent"

            Case 4
```

```
            lblScore.Text = "Very good"

         Case 3
            lblScore.Text = "Good"

         Case 2
            lblScore.Text = "Poor"

         Case Else
            lblScore.Text = "Fail"

      End Select

   End Sub ' DisplayScore

End Class ' FrmFlagQuiz
```

17.12 (*Salary Survey Application*) Use a one-dimensional array to solve the following problem: A company pays its salespeople on a commission basis. The salespeople receive $200 per week, plus 9% of their gross sales for that week. For example, a salesperson who grosses $5000 in sales in a week receives $200 plus 9% of $5000, a total of $650. Write an application (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assuming that each salesperson's salary is truncated to an integer amount): $200–$299, $300–$399, $400–$499, $500–$599, $600–$699, $700–$799, $800–$899, $900–$999 and over $999.

Allow the user to enter the sales for each employee in a `TextBox`. The user should click the **Calculate** `Button` to calculate that salesperson's salary. When the user is done entering this information, clicking the **Show Totals** `Button` should display how many of the salespeople earned salaries in each of the above ranges. The finished application should behave like Fig. 17.32.

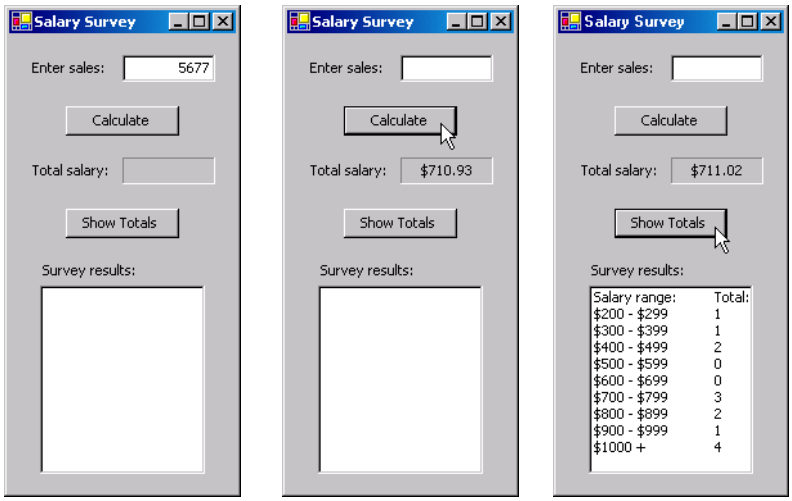


Figure 17.32 Salary Survey GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial17\Exercises\SalarySurvey` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SalarySurvey.sln` in the `SalarySurvey` directory to open the application.

c) ***Creating an array of salary ranges.*** Create a `String` array, and initialize it to contain the salary ranges (the `String`s displayed in the `ListBox`'s first column).

d) ***Create an array that represents the number of salaries in each range.*** Create an empty `Decimal` array to store the number of employees who earn salaries in each range.

e) ***Creating an event handler for the Calculate Button.*** Write event handler `btnCalculate_Click`. Obtain the user input from the **Enter sales:** `TextBox`. Calculate the commission due to the employee and add that amount to the base salary. Increment the element in array `decSalaries` that corresponds to the employee's salary range. This event handler should also display the employee's salary in the **Total salary:** `Label`.

f) ***Writing an event handler for the Show Totals Button.*** Create event handler `btnShowTotals_Click` to display the salary distribution in the `ListBox`. Use a `For...Next` statement to display the range (an element in `strSalaryRanges`) and the number of employees whose salary falls in that range (an element in `decSalaries`).

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter several sales amounts using the **Calculate** `Button`. Click the **Show Totals** `Button` and verify that the proper amounts are displayed for each salary range, based on the salaries calculate from your input.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 17.12 Solution
' SalarySurvey.vb

Public Class FrmSalarySurvey
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' salary ranges
   Dim m_strSalaryRanges As String() = New String() { _
      "$200 - $299", "$300 - $399", "$400 - $499", _
      "$500 - $599", "$600 - $699", "$700 - $799", _
      "$800 - $899", "$900 - $999", "$1000 +  "}

   ' number of employees in each salary range
   Dim m_intSalaries As Integer() = New Integer( _
      m_strSalaryRanges.GetUpperBound(0)) {}

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' obtain total sales
      Dim decSales As Decimal = Convert.ToDecimal( _
         Val(txtInputSales.Text))

      ' employee's base salary
      Dim decTotalSalary As Decimal = 200

      ' add commission to total salary
      decTotalSalary += Convert.ToDecimal(decSales * 0.09)

      ' display salary in a Label
      lblTotalSalary.Text = String.Format("{0:C}", decTotalSalary)

      ' increment the correct counter in array intSalaries
      Select Case decTotalSalary

         Case Is < 300
            m_intSalaries(0) += 1

```

```
            Case Is < 400
               m_intSalaries(1) += 1

            Case Is < 500
               m_intSalaries(2) += 1

            Case Is < 600
               m_intSalaries(3) += 1

            Case Is < 700
               m_intSalaries(4) += 1

            Case Is < 800
               m_intSalaries(5) += 1

            Case Is < 900
               m_intSalaries(6) += 1

            Case Is < 1000
               m_intSalaries(7) += 1

            Case Is >= 1000
               m_intSalaries(8) += 1

         End Select

         txtInputSales.Clear()  ' clear TextBox
      End Sub ' btnCalculate_Click

      ' handles click event for btnShowTotals Button
      Private Sub btnShowTotals_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnShowTotals.Click

         Dim intIndex As Integer = 0  ‘ counter

         ' clear all items in the ListBox
         lstSalaryTotals.Items.Clear()

         ' add header to ListBox
         lstSalaryTotals.Items.Add(“Salary range:” & _
            ControlChars.Tab & “Total:”)

         ' displays total for each salary range
         For intIndex = 0 To m_strSalaryRanges.GetUpperBound(0)
            lstSalaryTotals.Items.Add(m_strSalaryRanges(intIndex) & _
               ControlChars.Tab & m_intSalaries(intIndex))
         Next

      End Sub ' btnShowTotals_Click

   End Class ' FrmSalarySurvey
```

17.13 (***Cafeteria Survey Application***) Twenty students were asked to rate, on the scale from 1 to 10, the quality of the food in the student cafeteria, with 1 being “awful” and 10 being “excellent.” Allow the user input to be entered using a `ComboBox`. Place the 20 responses in an `Integer` array, and determine the frequency of each rating. Display the frequencies as a histogram in a multiline, scrollable `TextBox`. Figure 17.33 demonstrates the completed application.

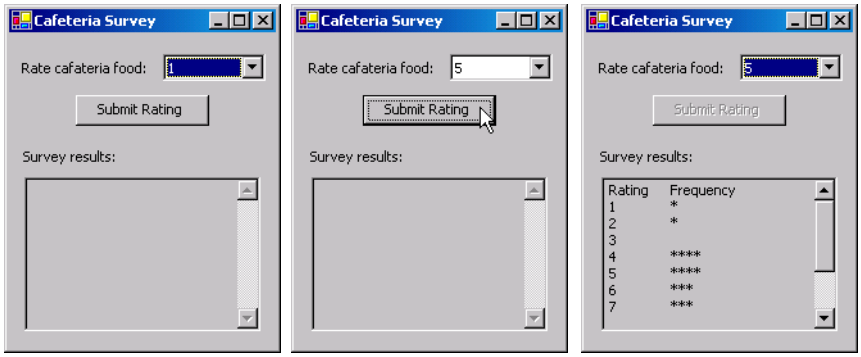


Figure 17.33 Cafeteria Survey GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial17\Exercises\CafeteriaSurvey` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CafeteriaSurvey.sln` in the `CafeteriaSurvey` directory.

c) ***Creating an array of the possible ratings.*** Create an array of 10 consecutive integers, called `m_intChoices` to contain the integers in the range 1–10, inclusive.

d) ***Adding a `ComboBox`.*** Add a `ComboBox` to the GUI as in Fig. 17.33. The `ComboBox` will display the possible ratings. Set property `DropDownStyle` to `DropDownList`.

e) ***Displaying the possible ratings when the application starts.*** Write the event handler for the `Load` event so that the `DataSource` of the `ComboBox` is set to `intChoices` when the application starts.

f) ***Creating an array to store the responses.*** Create an `Integer` array of length `11` named `m_intResponses`. This will be used to store the number of responses in each of the 10 categories (element `0` will not be used).

g) ***Counting the number of responses.*** Create an `Integer` variable named `m_intResponseCounter` to keep track of how many responses have been input.

h) ***Storing the responses.*** Write the event handler `btnSubmit_Click` to increment `m_intResponseCounter`. Store the response in array `m_intResponses`. Call procedure `DisplayHistogram` to display the results.

i) ***Creating procedure `DisplayHistogram`.*** Add a header to the `TextBox`. Use nested `For...Next` loops to display the ratings in the first column. The second column uses asterisks to indicate how many students surveyed submitted the corresponding rating.

j) ***Running the application.*** Select **Debug > Start** to run your application. Enter 20 responses using the **Submit Rating** `Button`. Verify that the resulting histogram displays the responses entered.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 17.13 Solution
' CafeteriaSurvey.vb

Public Class FrmCafeteriaSurvey
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' possible answers
   Dim m_intChoices As Integer() = { _
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

```

```
   ' counter for number of responses
   Dim m_intResponseCounter As Integer = 0

   ' array to keep track of all responses
   Dim m_intResponses As Integer() = New Integer(11) {}

   ' handles Submit Rating Button's Click event
   Private Sub btnSubmit_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSubmit.Click

      ' retrieve user input
      Dim intResponse As Integer = _
         Convert.ToInt32(cboInput.SelectedItem)

      If m_intResponseCounter < 20 Then
         m_intResponses(intResponse) += 1
      End If

      m_intResponseCounter += 1

      If m_intResponseCounter = 20 Then

         ' disable btnSubmit Button so no more
         ' responses can be entered
         btnSubmit.Enabled = False

         DisplayHistogram()
      End If

   End Sub ' btnSubmit_Click

   ' handles Cafeteria Survey Form's Load event
   Private Sub FrmCafeteriaSurvey_Load(ByVal sender As  _
      System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

      cboInput.DataSource = m_intChoices
   End Sub ' FrmCafeteriaSurvey_Load

   ' displays histogram
   Sub DisplayHistogram()

      ' construct output String with the frequencies as a histogram
      Dim strOutput As String = "Rating" & ControlChars.Tab & _
         "Frequency" & ControlChars.CrLf

      Dim intRatings As Integer
      Dim intCounter As Integer

      ' add entry to TextBox for each rating
      For intRatings = 1 To 10

         strOutput &= (intRatings.ToString & ControlChars.Tab)

         ' display asterisk for each user who gave this rating
         For intCounter = 1 To m_intResponses(intRatings)
            strOutput &= "*"
         Next

         strOutput &= ControlChars.CrLf
      Next

```

```
        txtOutput.Text = strOutput  ' display results in TextBox
    End Sub ' DisplayHistogram

End Class ' FrmHistogram
```

What does this code do?

17.14 This procedure declares `intNumbers` as its parameter. What does it return?

```
Function Mystery(ByVal intNumbers As Integer()) As Integer()

   Dim intI As Integer
   Dim intLength As Integer = intNumbers.Length - 1
   Dim intTempArray As Integer() = _
      New Integer(intLength) {}

   For intI = intLength To 0 Step -1
      intTempArray(intLength - intI) = intNumbers(intI)
   Next

   Return intTempArray
End Function ' Mystery
```

Answer: This procedure takes the contents of parameter `intNumbers` and reverses the order of its contents, returning the reversed array to the caller.

What's wrong with this code?

17.15 The code that follows uses a `For`…`Next` loop to sum the elements in an array. Find the error(s) in the following code:

```
Sub SumArray()

   Dim intSum As Integer
   Dim intCounter As Integer
   Dim intNumbers As Integer() = _
      New Integer() {1, 2, 3, 4, 5, 6, 7, 8}

   For intCounter = 0 To intNumbers.Length
      intSum += intNumbers(intCounter)
   Next

End Sub ' SumArray
```

Answer: Array `intNumbers` does have `intNumbers.Length` number of elements, but the indices are zero through `intNumbers.Length - 1`. The `For`…`Next` loop increments `intCounter` beyond the highest index in the array which results in a run-time error. The correct code is as follows:

```
Sub SumArray()
   Dim intSum As Integer
   Dim intCounter As Integer
   Dim intNumbers As Integer() = _
      New Integer() {1, 2, 3, 4, 5, 6, 7, 8}

   For intCounter = 0 To intNumbers.GetUpperBound(0)
      intSum += intNumbers(intCounter)
   Next

End Sub ' SumArray
```

Programming Challenge

17.16 (***Road Sign Test Application***) Write an application that will test the user's knowledge of road signs. Your application should display a random sign image and ask the user to select the sign name from a `ComboBox`. This application should look like Fig. 17.34. *Hint*: The application is similar to the **Flag Quiz** application. You can find the images in `C:\Examples\Tutorial17\Exercises\images`. Remember to set `Option Strict` to `On`.

Figure 17.34 Road Sign Test GUI.

Answer:

```
' Exercise 17.16 Solution
' RoadSignTest.vb

Public Class FrmRoadSignTest
   Inherits System.Windows.Forms.Form

   ' String array stores sign names
   Dim m_strOptions As String() = New String() { _
      "Do Not Enter", "Narrow bridge", "No bicycles", _
      "No left turn", "No Pedestrians", "No U-turn", _
      "Road Narrows", "Stop", "Stop sign ahead", _
      "Traffic signals ahead", "Winding road ahead", _
      "Yield"}

   ' Boolean array tracks displayed signs
   Dim m_blnUsed As Boolean() = _
      New Boolean(m_strOptions.GetUpperBound(0) {}

   Dim m_intCount As Integer = 1 ' number of signs shown
   Dim m_intCorrectAnswer As Integer ' index of current sign

   ' Windows Form Designer generated code

   ' handles Road Sign Test Form's Load event
   Private Sub FrmRoadSignTest_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      Array.Sort(m_strOptions)  ' alphabetize sign names

      ' display sign names in ComboBox
      cboOptions.DataSource = m_strOptions

      DisplaySign()  ' display first sign in PictureBox
   End Sub ' FrmRoadSignTest_Load

   ' return full path name of image file as a String
   Function BuildPathName() As String

      ' return full path name
      Return System.Environment.CurrentDirectory & _
         "\images\sign" & m_intCorrectAnswer & ".png"
   End Function ' BuildPathName

```

```
   ' return an unused random number
   Function GetUniqueRandomNumber() As Integer
      Dim objRandom As Random = New Random()
      Dim intRandom As Integer

      ' generate random numbers until unused sign is found
      Do
         intRandom = objRandom.Next(0, m_blnUsed.Length)
      Loop Until m_blnUsed(intRandom) = False

      ' indicate that sign has been used
      m_blnUsed(intRandom) = True

      Return intRandom  ' return index for new sign
   End Function ' GetUniqueRandomNumber

   ' display random sign in PictureBox
   Sub DisplaySign()

      ' unique index ensures that a sign is used no more than once
      m_intCorrectAnswer = GetUniqueRandomNumber()

      ' get image's full path name
      Dim strPath As String = BuildPathName()

      picSign.Image = Image.FromFile(strPath)  ' display image
   End Sub ' DisplaySign

   ' handles Submit Button's Click event
   Private Sub btnSubmit_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSubmit.Click

      ' retrieve answer from ComboBox
      Dim strResponse As String = _
         Convert.ToString(cboOptions.SelectedValue)

      ' verify answer
      If strResponse = m_strOptions(m_intCorrectAnswer) Then
         lblFeedback.Text = "Correct!"
      Else
         lblFeedback.Text = "Incorrect."
      End If

      ' inform user if test is over
      If m_intCount >= 5 Then  ' test is over
         lblFeedback.Text &= "   Done!"
         btnNext.Enabled = False
         btnSubmit.Enabled = False
         cboOptions.Enabled = False
      Else  ' test is not over
         btnSubmit.Enabled = False
         btnNext.Enabled = True
      End If

   End Sub ' btnSubmit_Click

   ' handles Next Sign Button's Click event
   Private Sub btnNext_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNext.Click

      DisplaySign()  ' display next sign
```

```
        lblFeedback.Text = ""   ' clear output

        ' change selected sign to first in ComboBox
        cboOptions.SelectedIndex = 0

        m_intCount += 1  ' update number of signs shown

        btnSubmit.Enabled = True
        btnNext.Enabled = False
    End Sub ' btnNext_Click

End Class ' FrmRoadSignTest
```

TUTORIAL 18

Sales Data Application

Introducing Two-Dimensional Arrays, `RadioButtons` and the `MSChart` Control

Solutions

Instructor's Manual Exercise Solutions Tutorial 18

MULTIPLE-CHOICE QUESTIONS

18.1 `RadioButton` controls should be prefixed with _____.

a) `rad` b) `rbn`
c) `btn` d) `radbtn`

18.2 A two-dimensional array in which each row contains the same number of columns is called a _____ array.

a) data b) rectangular
c) tabular d) All of the above.

18.3 In an *m*-by-*n* array, the *m* stands for _____.

a) the number of columns in the array b) the total number of array elements
c) the number of rows in the array d) the number of elements in each row

18.4 The statement _____ assigns an array of three columns and five rows to two-dimensional `Integer` array `intArray`.

a) `intArray = New Integer(5, 3)` b) `intArray = New Integer(4, 2)`
c) `intArray = New Integer(4, 3)` d) `intArray = New Integer(5, 2)`

18.5 To change the `MSChart` graph's title size, use the _____ tab of the `MSChart` **Properties** dialog.

a) **Size** b) **Title**
c) **Fonts** d) **Font Size**

18.6 Use a _____ to group `RadioButtons` on the `Form`.

a) `GroupBox` control b) `ComboBox` control
c) `ListBox` control d) None of the above.

18.7 Use the _____ tab of the `MSChart` **Properties** dialog to access properties to include a border around your chart.

a) **Border** b) **Backdrop**
c) **BorderStyle** d) **Background**

18.8 A point of height is approximately equal to _____.

a) 1/72" b) 1"
c) 1/4" d) 1/36"

18.9 The **Chart Type** `GroupBox` of the `MSChart` control's **Properties** dialog is located in the _____ tab.

a) **Chart** b) **Graph**
c) **Series** d) **ChartType**

18.10 The **Fonts** tab of the `MSChart` control's **Properties** dialog allows you to change the font _____.

a) **Style** b) **Width**
c) **Color** d) All of the above.

Answers: 18.1) a. 18.2) b. 18.3) c. 18.4) b. 18.5) c. 18.6) a. 18.7) b. 18.8) a. 18.9) a. 18.10) d.

EXERCISES

18.11 (***Stock Price Application***) It is often useful to track a company's stock price over time by using a line graph. You will learn to create a line graph by using an `MSChart` control in this exercise. Create an application that allows users to enter values for a company's stock

price at the end of six consecutive quarters and graph that data, using an `MSChart` control (Fig. 18.23).

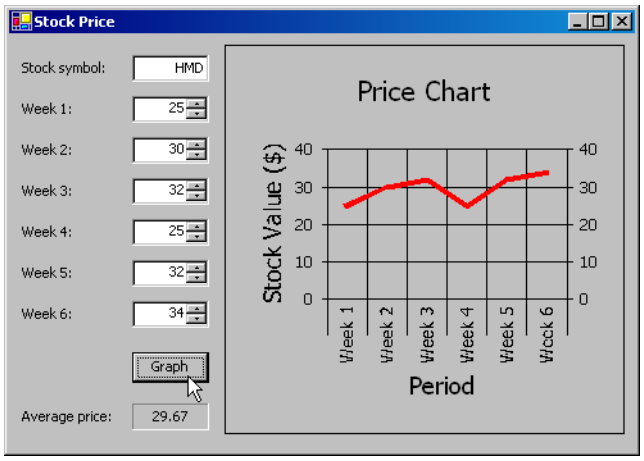


Figure 18.23 Stock Price application GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial18\Exercises\StockPrice` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `StockPrice.sln` in the `StockPrice` directory to open the application.

c) ***Changing the graph type and color***. To change the graph type and color, click the chart in **Design View** and select the **ActiveX -Properties** hyperlink from the **Properties** window to display the **Properties** dialog. In the **Chart** tab's **Chart Type** `GroupBox`, select **Line** (if it is not already selected), and make sure the `RadioButton` is set to **2D**. Click the **Series Color** tab, and, in the **Edge/Line** `GroupBox`, change the color to red (if this color is not already selected).

d) ***Inserting code in the Graph `Button`'s `Click` event handler***. Double click the **Graph** `Button` in **Design** view to generate the **Graph** `Button` `Click` event handler. Insert an `If...Then` statement that verifies that the user entered a stock name. If the value in the `TextBox` is the empty string, the application should display a message dialog.

e) ***Checking the remaining `NumericUpDowns`***. For the chart to function properly, the user must enter values in all `NumericUpDowns`. Add `If...ElseIf...Else` statements to verify that there is a value greater than zero in each `NumericUpDown` (using the `Value` property), and display an error message in a `MessageBox` if any values are missing (contain the zero).

f) ***Calculating the average***. Calculate the average of all of the values from the `NumericUpDowns`, and output the result in the output `Label lblAverage`.

g) ***Inserting data into an array***. Create a 6-by-2 array from the data that will display the week number on the *x*-axis and prices on the *y*-axis. (The week number should be in the first column, and the stock price should be in the second column.) Use the `NumericUpDown`'s `Text` property to retrieve the value stored in the control as a `String`.

h) ***Displaying the chart***. Assign the array to the `ChartData` property to display the data in the array. Remember to set `Label lblMessage`'s `Visible` property to `False` and set the `MSChart` control's `Visible` property to `True`. A control's **`Visible`** property determines if the control is displayed on the `Form` (`True`) or hidden (`False`).

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter the name of a stock, and the stock's price for each week. Click the **Graph** `Button`. Verify that the average displayed is correct, and that the graph displayed shows the proper data. Also, make sure your graph's labels appear as in Fig. 18.23.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 18.11 Solution
' StockPrice.vb

Public Class FrmStockPrice
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Private Sub btnGraph_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnGraph.Click

      ' declaring two-dimensional array
      Dim strPrice(,) As String = New String(5, 1) {}
      Dim dblAverage As Double = 0 ' average stock price
      Dim intCounter As Integer = 0 ' missing stock price counter

      ' if stock name is missing
      If txtStockName.Text = "" Then
         MessageBox.Show("Stock name is required", _
            "Stock Name Missing", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)

      ' if stock prices were missing
      ElseIf updFirst.Value = 0 OrElse updSecond.Value = 0
         OrElse updThird.Value = 0 OrElse updFourth.Value = 0
         OrElse updFifth.Value = 0 OrElse updSixth.Value = 0 Then

         MessageBox.Show("Please fill in all weekly values", _
            "Missing Price", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      Else
         lblMessage.Visible = False
         chStock.Visible = True

         ' calculate average
         dblAverage = (updFirst.Value + updSecond.Value + _
            updThird.Value + updFourth.Value + _
            updFifth.Value + updSixth.Value) / 6

         ' output average price
         lblAverage.Text = String.Format("{0:F}", dblAverage)

         ' put all data in a two-dimensional array
         strPrice = New String(,) {{"Week 1", txtFirst.Text}, _
            {"Week 2", txtSecond.Text}, _
            {"Week 3", txtThird.Text}, _
            {"Week 4", txtFourth.Text}, _
            {"Week 5", txtFifth.Text}, _
            {"Week 6", txtSixth.Text}}

         ' bind data to chart
         chStock.ChartData = strPrice

      End If

   End Sub ' btnGraph_Click
```

```
End Class ' FrmStockPrice
```

18.12 ***(Enhanced Lottery Picker)*** A lottery commission offers four different lottery games to play: three-number, four-number, five-number and five-number + 1 lotteries. In Tutorial 16, your **Lottery Picker** application could select duplicate numbers for each lottery. In this exercise, you enhance the **Lottery Picker** to prevent duplicate numbers for the five-number and five-number + 1 lotteries (Fig. 18.24). According to this new requirement the games are now played as follows:

- Three-number lotteries require players to choose three numbers in the range of 0–9.
- Four-number lotteries require players to choose four numbers, in the range of 0–9.
- Five-number lotteries require players to choose five unique numbers in the range of 1–39.
- Five-number + 1 lotteries require players to choose five unique numbers in the range of 1–49 and an additional unique number in the range of 1–42.

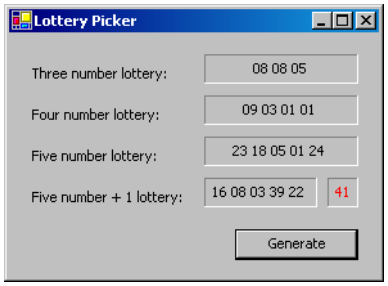


Figure 18.24 Enhanced **Lottery Numbers Picker** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial18\Exercises\EnhancedLotteryPicker` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedLotteryPicker.sln` in the `EnhancedLotteryPicker` directory to open the application.

c) ***Declaring a two-dimensional array to maintain unique random numbers.*** Declare an instance variable `m_blnNumbers` that stores a 2-by-50 `Boolean` array. You will use this array later in this exercise to test whether a lottery number has already been chosen.

d) ***Initializing the array.*** Each time the user clicks the **Generate** `Button`, the application should initialize the array by declaring its rows and setting the initial values. Write a `ClearArray` procedure that uses a `For…Next` statement to assign each value in the `m_blnNumbers` array to `False`.

e) ***Modifying the `Generate Function` procedure.*** You will modify the `Generate Function` procedure to use the `Boolean` array to pick unique random numbers. Begin by writing a statement that generates a random number and assigns its value to an `Integer` variable `intNumber`.

f) ***Determining whether the random number has already been selected.*** Use an `If…Then` statement to determine whether the maximum lottery number is less than `40`. (This happens when the upper limit on the random number equals `40`.) In this case, you will examine the first row of the array. To maintain unique numbers, you will set the value of the element in that row whose index equals the random number to `True` (indicating that it has been picked). For example, if the random number `34` has been picked, `blnNumbers(0)(34)` would contain the value `True`. To test whether a number has been picked, use a `Do While…Loop` statement inside the `If…Then` statement to access that element of the array. If the array element's value is `True`, use the body of the loop to assign a new random number to `intNumber`. If the value in the array is `False`, use the condition in the `Do While…Loop` header to ignore the

body of the loop. Just outside the `Do While…Loop`, include a statement that modifies the array to indicate that the number has now been picked.

g) ***Completing the application***. Use a second `If…Then` statement to determine whether the maximum lottery number is greater than 40. In this case, you will examine the second row of the array. Repeat the process in the previous step. Remember to return the value stored in `intNumber` at the end of the `Generate Function` procedure.

h) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Generate** `Button` and verify that the values displayed fall into the ranges specified in the exercise's description.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 18.12 Solution
' LotteryPicker.vb

Public Class FrmLotteryPicker
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Dim m_objRandom As Random = New Random  ' create Random object

   ' declare new array to store chosen lottery numbers
   Dim m_blnNumbers As Boolean(,) = New Boolean(1, 49) {}

   ' display random lottery numbers
   Private Sub btnGenerate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnGenerate.Click

      ClearArray() ' clear array values

      ' generate three numbers
      lblOutput3.Text = Generate(0, 10) & " " & _
         Generate(0, 10) & " " & Generate(0, 10)

      ' generate four numbers
      lblOutput4.Text = Generate(0, 10) & " " & _
         Generate(0, 10) & " " & Generate(0, 10) & " " _
         & Generate(0, 10)

      ' generate five numbers
      lblOutput5.Text = Generate(1, 40) & " " & _
         Generate(1, 40) & " " & Generate(1, 40) & _
         " " & Generate(1, 40) & " " & Generate(1, 40)

      ' generate five plus one numbers
      lblOutput5Plus1.Text = Generate(1, 50) & " " & _
         Generate(1, 50) & " " & Generate(1, 50) & " " & _
         Generate(1, 50) & " " & Generate(1, 50)

      lblOutputExtra1.Text = Generate(1, 43) ' generate extra numbers
   End Sub ' btnGenerate_Click

   ' generate random numbers
   Function Generate(ByVal intLow As Integer, _
      ByVal intHigh As Integer) As String
```

```

      ' generate random number with given boundaries
      Dim intNumber As Integer = _
         m_objRandom.Next(intLow, intHigh)

      ' use first row for five-number lottery
      If intHigh = 40 Then

         ' select new random number
         Do While m_blnNumbers(0, intNumber) = True
            intNumber = m_objRandom.Next(intLow, intHigh)
         Loop

         m_blnNumbers(0, intNumber) = True ' mark number as used
      End If

      ' use second row for five-number + 1 lottery
      If intHigh > 40 Then

         ' select another random number that is not used
         Do While m_blnNumbers(1, intNumber) = True
            intNumber = m_objRandom.Next(intLow, intHigh)
         Loop

         m_blnNumbers(1, intNumber) = True ' mark number as used
      End If

      Return String.Format("{0:D2}", intNumber)
   End Function ' Generate

   ' assigns all values in array to False
   Sub ClearArray()

      Dim intI As Integer = 0

      ' assign all row 0 cells False
      For intI = 0 To 39
         m_blnNumbers(0, intI) = False
      Next

      ' assign all row 1 cells False
      For intI = 0 To 49
         m_blnNumbers(1, intI) = False
      Next

   End Sub ' ClearArray

End Class ' FrmLotteryPicker
```

18.13 (*Student Grades Application*) A teacher needs an application that computes each student's grade average (on a scale of 0 to 100 points) and the class average for ten students. The application should add a student's name and test average (separated by a tab character) to a `ListBox` and calculate the class grade average each time the user clicks the **Submit Grades** `Button` (Fig. 18.25). The **Submit Grades** `Button` should be disabled after ten students' grades have been entered.

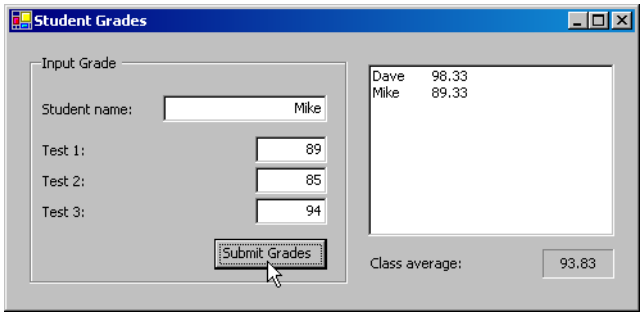


Figure 18.25 Student Grades application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial18\Exercises\StudentGrades` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `StudentGrades.sln` in the `StudentGrades` directory to open the application.

c) ***Declare instance variables.*** Declare an `Integer` counter and a 10-by-2 `String` array as instance variables.

d) ***Coding the Submit Grades `Button`'s `Click` event handler.*** Double click the **Submit Grades** `Button` to generate its `Click` event handler. Write code in the event handler to retrieve input from the `TextBoxes`. Then store the student's name in the first column of the two-dimensional `String` array and the student's test average in the second column of the array. Use a `Function` procedure to calculate the student's test average.

e) ***Computing the class average.*** Add the student's name and the student's test average (separated by a tab character) to the `ListBox`. Then calculate and display the class average, using a `Function` procedure. [*Hint*: You should use the two-dimensional `String` array and the `Integer` counter to calculate the class average.]

f) ***Completing the event handler.*** Increment the counter by one after calculating the class average. If ten students' grades have been entered, disable the **Submit Grades** `Button`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 students and their grades. Make sure that all averages are correct, and that the **Submit Grades** `Button` is disabled after 10 students' grades have been entered.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 18.13 Solution
' StudentGrades.vb

Public Class FrmStudentGrades
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' array of students and test averages
   Dim m_strStudents As String(,) = New String(9, 1) {}

   ' counter for number of grades in array
   Dim m_intCounter As Integer = 0

   ' handles click event for btnSubmit Button
   Private Sub btnSubmit_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSubmit.Click

```

```
        ' extract user input
        Dim dblTest1 As Double = Val(txtTest1.Text)
        Dim dblTest2 As Double = Val(txtTest2.Text)
        Dim dblTest3 As Double = Val(txtTest3.Text)

        ' add student name and test average
        ' to the array of students
        m_strStudents(m_intCounter, 0) = txtName.Text
        m_strStudents(m_intCounter, 1) = _
           String.Format("{0:F}", _
              StudentAverage(dblTest1, dblTest2, dblTest3))

        ' display student name and average in ListBox
        lstNames.Items.Add(m_strStudents(m_intCounter, 0) & _
           ControlChars.Tab & m_strStudents(m_intCounter, 1))

        ' display class average
        lblClassOutput.Text = _
           String.Format("{0:F}", ClassAverage())

        ' increment grade counter
        m_intCounter += 1

        ' maximum of ten students
        If m_intCounter = 10 Then
           btnSubmit.Enabled = False
        End If

     End Sub ' btnSubmit_Click

     ' returns student average
     Function StudentAverage(ByVal intTest1 As Double, _
        ByVal intTest2 As Double, ByVal intTest3 As Double) _
        As Double

        ' return the average of 3 test scores
        Return ((intTest1 + intTest2 + intTest3) / 3)
     End Function ' StudentAverage

     ' returns class average
     Function ClassAverage() As Double
        Dim intCounter As Integer = 0 ' counter variable
        Dim dblTotal As Double = 0.0 ' total test score

        ' iterate through array of students
        For intCounter = 0 To m_intCounter
           dblTotal += _
              Convert.ToDouble(m_strStudents(intCounter, 1))
        Next

        ' return the class average
        Return (dblTotal / (m_intCounter + 1))
     End Function ' ClassAverage

  End Class ' FrmStudentGrades
```

What does this code do?

18.14 What is returned by the following code? Assume that `GetStockPrices` is a `Function` procedure that returns a 2-by-31 array, with the first row containing the stock price at the

beginning of the day and the last row containing the stock price at the end of the day, for each day of the month.

```
Function Mystery() As Integer()
   Dim intPrices As Integer(,) = New Integer(1, 30) {}

   intPrices = GetStockPrices()

   Dim intResult As Integer() = New Integer(30) {}
   Dim intI As Integer

   For intI = 0 To 30
      intResult(intI) = intPrices(0, intI) - intPrices(1, intI)
   Next

   Return intResult
End Function ' Mystery
```

Answer: The `Function` procedure returns a one-dimensional array containing the daily stock price change for each day of the month.

What's wrong with this code?

18.15 Find the error(s) in the following code. The `TwoDArrays` procedure should create a two-dimensional array and initialize all its values to one.

```
Sub TwoDArrays()
   Dim intArray As Integer(,)

   intArray = New Integer(3, 3) {}

   Dim intI As Integer

   ' assign 1 to all cell values
   For intI = 0 To 3
      intArray(intI, intI) = 1
   Next

End Sub ' TwoDArrays
```

Answer: To assign each element in a two-dimensional array two nested `For`…`Next` loops should be used. The corrected code is as follows:

```
Sub TwoDArrays()
   Dim intArray As Integer(,)
   intArray = New Integer(3, 3) {}
   Dim intI As Integer
   Dim intJ As Integer

   ' assign 1 to all cell values
   For intI = 0 To 3

      For intJ = 0 To 3
         intArray(intI, intJ) = 1
      Next

   Next

End Sub ' TwoDArrays
```

Programming Challenge

18.16 (***Enhanced Student Grades Application***) Modify the application in Exercise 18.13 to include a bar graph that displays each student's grade and the class average. The chart should display only after the user enters the grades for ten students (Fig. 18.26). Until then, a Label should display the text, "Enter ten grades to display the chart." Student names and the text `Class Average` should display on the *x*-axis, and grades should display on the *y*-axis. Also write code that ensures that the user has entered values in each `TextBox` when the **Submit Grades** `Button` is clicked.

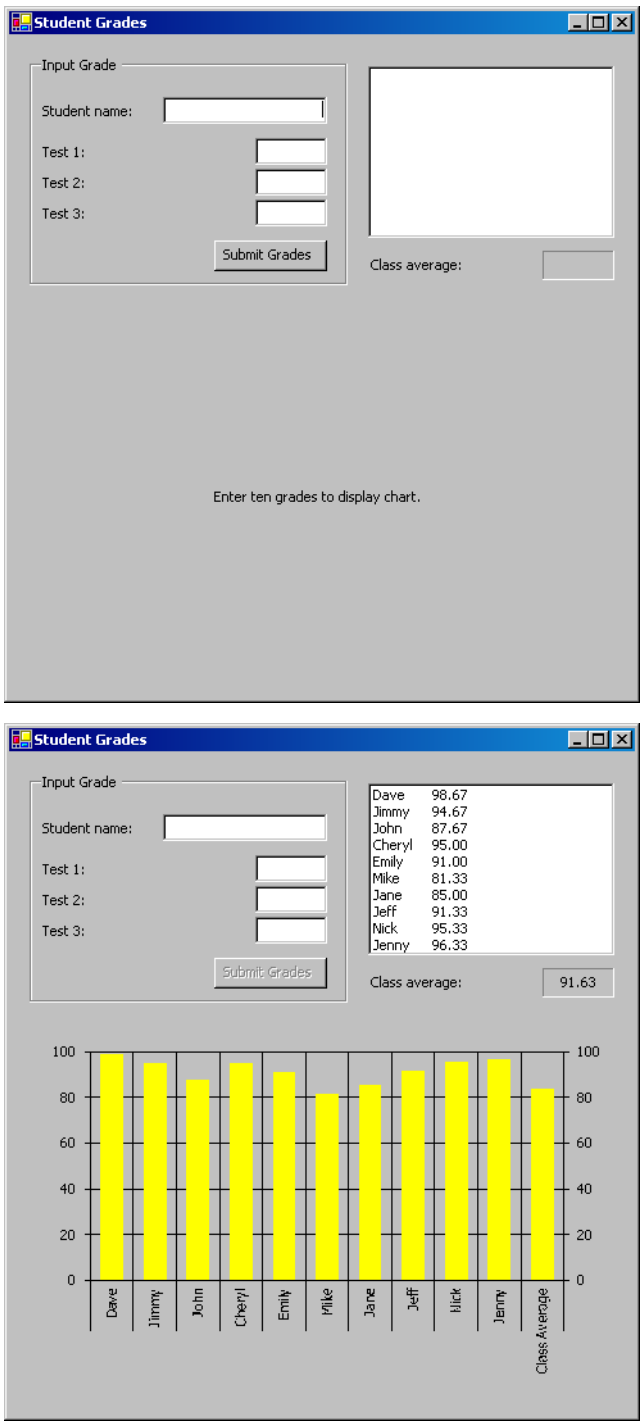


Figure 18.26 Enhanced **Student Grades** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial18\Exercises\EnhancedStudentGrades` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedStudentGrades.sln` in the `EnhancedStudentGrades` directory to open the application. If you have not completed Exercise 18.13, follow the steps in Exercise 18.13 before proceeding to the next step. If you have completed Exercise 18.13, copy the code you wrote into this application.

c) ***Completing the application design***. Place a `Label` on the `Form` that reads "`Enter ten grades to display the chart`." Then insert an `MSChart` control at the bottom of the `Form`. Set its `Visible` property to `False` to hide the graph initially. A control's **`Visible`** property determines if the control is displayed on the `Form` (`True`) or hidden (`False`). Open the `MSChart` control's **Properties** dialog, set the **Chart Type** to a `2D Bar/PictoGraph` and set the **Series Interior Color** property (under the **Series Color** tab) to yellow.

d) ***Inserting code to check input***. Write code that determines whether the user has entered values in each `TextBox`. If the student name is missing, display a `MessageBox` indicating that the user must enter a student name. If any grades are missing, display a `MessageBox` indicating that three grades are required.

e) ***Enhancing the user interface***. Write code to clear each `TextBox` after the user clicks the **Submit Grades** `Button`. Then insert code to set the focus of the application to the **Student Name:** `TextBox`.

f) ***Displaying the chart***. Write a procedure that displays each of the ten students' test averages and the class average in the `MSChart` control. Recall that student names and the text `Class Average` should display on the *x*-axis, and grades should display on the *y*-axis. Finally, set the chart's `Visible` property to `True`, and set the `Visible` property of the `Label` you added in *Step b* to `False`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 sets of grades. Verify that the resulting chart displays the proper data, and that the chart is formatted as in Fig. 18.26.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 18.16 Solution
' StudentGrades.vb

Public Class FrmStudentGrades
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' array of students and test averages
   Dim m_strStudents As String(,) = New String(10, 1) {}

   ' counter for number of grades in array
   Dim m_intCounter As Integer = 0

   ' handles click event for btnSubmit Button
   Private Sub btnSubmit_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSubmit.Click

      ' if student name is missing
      If txtName.Text = "" Then
         MessageBox.Show("Student Name is required", _
            "Student Name Missing", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

```

```
         ' if a test grade is missing
      ElseIf txtTest1.Text = "" _
         OrElse txtTest2.Text = "" _
         OrElse txtTest3.Text = "" Then

         MessageBox.Show("Test grade is missing", _
            "Grade Missing", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)

         ' if all the conditions are satisfied
         ' then perform calculations and display chart
      Else

         ' extract user input
         Dim dblTest1 As Double = Val(txtTest1.Text)
         Dim dblTest2 As Double = Val(txtTest2.Text)
         Dim dblTest3 As Double = Val(txtTest3.Text)

         ' add student name and test average
         ' to the array of students
         m_strStudents(m_intCounter, 0) = txtName.Text
         m_strStudents(m_intCounter, 1) = _
            String.Format("{0:F}", _
               StudentAverage(dblTest1, dblTest2, dblTest3))

         ' display student name and average in ListBox
         lstNames.Items.Add(m_strStudents(m_intCounter, 0) & _
            ControlChars.Tab & m_strStudents(m_intCounter, 1))

         ' display class average
         lblClassOutput.Text = _
            String.Format("{0:F}", ClassAverage())

         ' increment grade counter
         m_intCounter += 1

         ' maximum of ten students
         If m_intCounter = 10 Then
            btnSubmit.Enabled = False
            ShowChart()
         End If

         ' clear TextBoxes and set focus
         txtName.Clear()
         txtTest1.Clear()
         txtTest2.Clear()
         txtTest3.Clear()
         txtName.Focus()
      End If

   End Sub ' btnSubmit_Click

   ' returns student average
   Function StudentAverage(ByVal intTest1 As Double, _
      ByVal intTest2 As Double, ByVal intTest3 As Double) _
      As Double

      Return ((intTest1 + intTest2 + intTest3) / 3)
   End Function ' StudentAverage

   ' returns class average
```

```
   Function ClassAverage() As Double
      Dim intCounter As Integer = 0 ' counter variable
      Dim dblTotal As Double = 0.0 ' store total test scores

      ' iterate through array of students
      For intCounter = 0 To m_intCounter
         dblTotal += _
            Convert.ToDouble(m_strStudents(intCounter, 1))
      Next

      ' return average test score for the class
      Return (dblTotal / (m_intCounter + 1))
   End Function ' ClassAverage

   ' places data in Chart and displays it
   Sub ShowChart()
      lblChart.Visible = False ' hide the Label
      chGrades.Visible = True ' display the grade chart
      m_strStudents(10, 0) = "Class Average"
      m_strStudents(10, 1) = String.Format("{0:F}", ClassAverage())
      chGrades.ChartData = m_strStudents
   End Sub ' ShowChart

End Class ' FrmStudentGrades
```

TUTORIAL 19

Microwave Oven Application

Building Your Own Classes and Objects
Solutions

Instructor's Manual Exercise Solutions Tutorial 19

MULTIPLE-CHOICE QUESTIONS

19.1 A `Button` appears flat if its _____ property is set to `Flat`.

a) `BorderStyle` b) `FlatStyle`
c) `Style` d) `BackStyle`

19.2 Keyword _____ introduces a class definition.

a) `NewClass` b) `ClassDef`
c) `VBClass` d) `Class`

19.3 Keyword _____ is used to create an object.

a) `CreateObject` b) `Instantiate`
c) `Create` d) `New`

19.4 `String` characters are of data type _____.

a) `Char` b) `StringCharacter`
c) `Character` d) `strCharacter`

19.5 The _____ is used to retrieve the value of an instance variable.

a) `Get` accessor of a property b) `Retrieve` method of a class
c) `Client` method of a class d) `Set` accessor of a property

19.6 When you enter the header for a constructor in Visual Studio .NET then press *Enter*, the keywords _____ are created for you.

a) `End Public Class` b) `End Procedure`
c) `End Sub` d) `End`

19.7 An important difference between constructors and other methods is that _____.

a) constructors cannot specify a return data type
b) constructors cannot specify any parameters
c) other methods are implemented as `Sub` procedures
d) constructors can assign values to instance variables

19.8 A class can yield many _____, just as a primitive data type can yield many variables.

a) names b) objects
c) values d) types

19.9 The `Set` accessor enables you to _____.

a) provide range checking b) modify data
c) provide data validation d) All of the above.

19.10 Instance variables declared `Private` are not accessible _____.

a) outside the class b) by other methods of the same class
c) by other members of the same class d) inside the same class

Answers: 19.1) b. 19.2) d. 19.3) d. 19.4) a. 19.5) a. 19.6) c. 19.7) a. 19.8) b. 19.9) d. 19.10) a.

EXERCISES

19.11 (*Triangle Creator Application*) Create an application that allows the user to enter the lengths for the three sides of a triangle as `Integers`. The application should then determine whether the triangle is a right triangle (two sides of the triangle form a 90 degree angle), an equilateral triangle (all sides of equal length) or neither. The application's GUI is completed for you (Fig. 19.49). You must create a class to represent a triangle object and define the event handler for the **Create** `Button`.

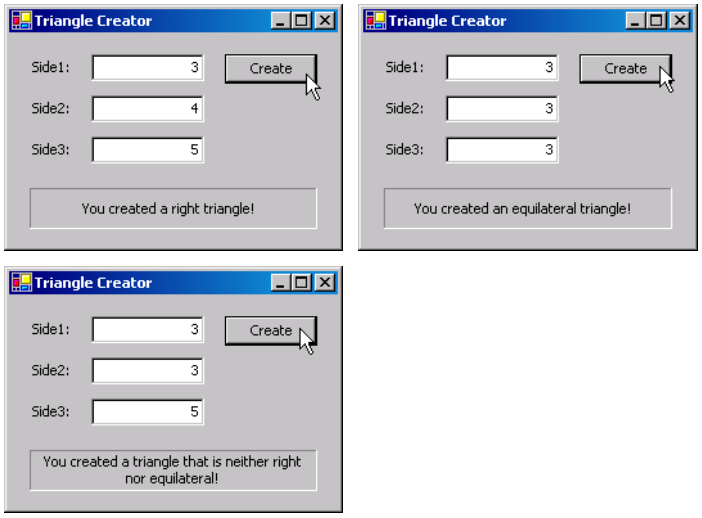


Figure 19.49 **Triangle Creator** application with all possible outputs

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial19\Exercises\Triangle` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Triangle.sln` in the `Triangle` directory to open the application.

c) ***Creating the `Triangle` class.*** Add a class to the project, and name it `Triangle`. This is where you will define the properties of the `Triangle` class.

d) ***Defining the necessary properties.*** Define a constructor that will take the lengths of the three sides of the triangle as arguments. Create three properties that enable clients to access and modify the lengths of the three sides. If the user enters a negative value, that side should be assigned the value zero.

e) ***Adding additional features.*** Create two more properties in the `Triangle` class: One determines whether the sides form a right triangle, the other an equilateral triangle. These properties are considered **read-only**, because you would naturally define only the `Get` accessor. There is no simple `Set` accessor that can make a triangle a right triangle or an equilateral triangle without first modifying the lengths of the triangle's sides. To create a read-only property (where the `Set` accessor is omitted), precede keyword `Property` with the keyword **`ReadOnly`**.

f) ***Adding code to event handler.*** Now that you have created your `Triangle` class, you can use it to create objects in your application. Double click the **Create** `Button` in **Design View** to generate the event handler. Create new variables to store the three lengths from the `TextBoxes`; then, use those values to create a new `Triangle` object.

g) ***Displaying the result.*** Use an `If…ElseIf` statement to determine if the triangle is a right triangle, an equilateral triangle or neither. Display the result in a `Label`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Create various inputs until you have create an equilateral triangle, a right triangle and a triangle that is neither right nor equilateral. Verify that the proper output is displayed for each.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 19.11 Solution
' Triangle.vb
' Represent a triangle

Public Class Triangle

   ' declare three private side values
```

```
    Private m_intSide1 As Integer
    Private m_intSide2 As Integer
    Private m_intSide3 As Integer

    ' Triangle constructor (side1, side2 and side3)
    Public Sub New(ByVal side1Value As Integer, _
       ByVal side2Value As Integer, _
       ByVal side3Value As Integer)

       Side1 = side1Value
       Side2 = side2Value
       Side3 = side3Value
    End Sub ' New

    ' get and set Side1
    Public Property Side1() As Integer

       ' return m_intSide1 value
       Get
          Return m_intSide1 ' return length of side1
       End Get ' end of Get accessor

       ' set side value
       Set(ByVal Value As Integer)

          ' make sure value is non-negative
          If Value > 0 Then
             m_intSide1 = Value
          Else
             m_intSide1 = 0 ' set to zero
          End If

       End Set ' end of Set accessor

    End Property ' Side1

    ' get and set Side2
    Public Property Side2() As Integer

       ' return m_intSide2 value
       Get
          Return m_intSide2 ' return length of side2
       End Get ' end of Get accessor

       ' set m_intSide2 value
       Set(ByVal Value As Integer)

          ' check if value is negative
          If Value > 0 Then
             m_intSide2 = Value ' set value
          Else
             m_intSide2 = 0
          End If

       End Set ' end of Set accessor

    End Property ' Side2

    ' get and set Side3
    Public Property Side3() As Integer

```

```
      ' return m_intSide3 value
      Get
         Return m_intSide3 ' return length of side3
      End Get ' end of Get accessor

      ' set m_intSide3
      Set(ByVal Value As Integer)

         ' make sure value is nonnegative
         If Value > 0 Then
            m_intSide3 = Value ' set value
         Else
            m_intSide3 = 0 ' set to zero
         End If

      End Set ' end of Set accessor

   End Property ' Side3

   ' test if triangle is equilateral
   Public ReadOnly Property Equilateral() As Boolean

      ' check sides, return True or False
      Get

         ' test if sides are equal
         If m_intSide1 = m_intSide2 _
            AndAlso m_intSide1 = m_intSide3 Then

            Return True ' indicate that sides are equal
         Else
            Return False ' indicate triangle is not equal sided
         End If

      End Get ' end of Get accessor

   End Property ' Equilateral

   ' check if sides create right triangle
   Public ReadOnly Property RightTriangle() As Boolean

      ' check sides, return True or False
      Get

         ' check length
         If (m_intSide1 ^ 2) + _
            (m_intSide2 ^ 2) = _
            (m_intSide3 ^ 2) Then

            Return True ' it is a right triangle

            ' check another length combination
         ElseIf (m_intSide1 ^ 2) + _
            (m_intSide3 ^ 2) = _
            (m_intSide2 ^ 2) Then

            Return True ' it is a right triangle

            ' check last length combination
         ElseIf (m_intSide2 ^ 2) + _
            (m_intSide3 ^ 2) = _
```

```
            (m_intSide1 ^ 2) Then

            Return True  ' it is a right triangle

         Else
            Return False ' it is not a right triangle

         End If

      End Get ' end of Get accessor

   End Property ' RightTriangle

End Class ' Triangle
```

```
' Exercise 19.11 Solution
' TriangleCreator.vb

Public Class FrmTriangleCreator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' create and test triangle properties
   Private Sub btnCreate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCreate.Click

      Dim objTriangle As Triangle ' create Triangle reference

      ' values for three sides
      Dim intSide1 As Integer = Convert.ToInt32(Val(txtSide1.Text))
      Dim intSide2 As Integer = Convert.ToInt32(Val(txtSide2.Text))
      Dim intSide3 As Integer = Convert.ToInt32(Val(txtSide3.Text))

      lblDisplay.Text = "" ' clear display Label

      ' create triangle object
      objTriangle = New Triangle(intSide1, intSide2, intSide3)

      ' test for right triangle
      If (objTriangle.RightTriangle = True) Then
         lblDisplay.Text = "You created a right triangle!"

         ' test for equilateral triangle
      ElseIf (objTriangle.Equilateral = True) Then
         lblDisplay.Text = "You created an equilateral triangle!"
      Else

         ' triangle is neither right nor equilateral
         lblDisplay.Text = ("You created a triangle that is " & _
            "neither right nor equilateral!")
      End If

   End Sub ' btnCreate_Click

End Class  ' FrmTriangleCreator
```

19.12 (***Modified Microwave Oven Application***) Modify the tutorial's **Microwave Oven** application to include an additional digit, which would represent the hour. Allow the user to enter up to 9 hours, 59 minutes, and 59 seconds (Fig. 19.50).

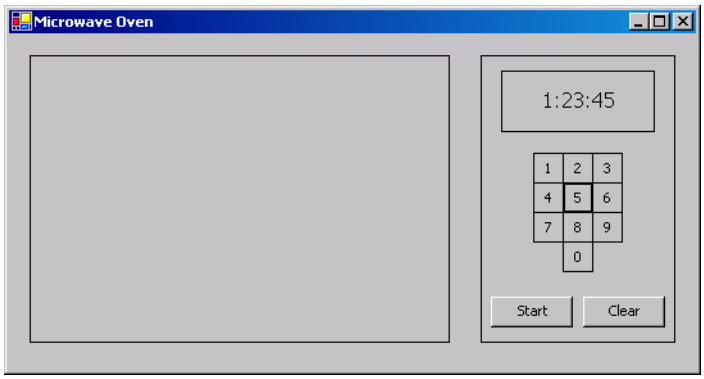


Figure 19.50 **Microwave Oven** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial19\Exercises\MicrowaveOven2` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `MicrowaveOven2.sln` in the `MicrowaveOven2` directory to open the application.

c) ***Adding the hour variable.*** To allow cooking time that includes the hour digit, you will need to modify the `Time` class. Define a new `Private` instance variable to represent the hour. Change the `Time` constructor to take in as its first argument (now `Time` should have three arguments) the hour amount. You will also have to modify the **Start** `Button` event handler and the `DisplayTime` method to include an hour variable.

d) ***Adding the `Hour` property.*** Use the `Minute` and `Second` properties as your template to create the property for the hour. Remember, we are allowing an additional digit to represent the hour (hour < 10).

e) ***Changing the padding amount.*** Change the calls to the `PadLeft` method to be consistent with the new time format.

f) ***Extracting the hour.*** Add a call to the `Substring` method so that hour gets the first digit in the `m_strTime String`. Also, change the calls to the `Substring` method for minute and second so that they extract the proper digits from the `m_strTime String`.

g) ***Accessing the first five digits.*** Change the `If…Then` statement from the `DisplayTime` method to take and display the first five digits entered by the user.

h) ***Edit the `Timer` object.*** Edit the `tmrClock_Tick` event handler to provide changes to hours and its corresponding minutes and seconds.

i) ***Displaying the time.*** Edit the `Format String` so that the display `Label` includes the hour.

j) ***Running the application.*** Select **Debug > Start** to run your application. Enter various times and verify that the application counts down properly. Enter an amount of time that is 10 hours or longer, and verify that the application handles invalid input correctly.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 19.12 Solution
' Time.vb
' Represents time in 24-hour format and contains properties

Public Class Time

```

```
    ' declare Integers for hour, minute and second
    Private m_intHour As Integer
    Private m_intMinute As Integer
    Private m_intSecond As Integer

    ' Time constructor (hour, minute and second supplied)
    Public Sub New(ByVal hourValue As Integer, _
       ByVal minuteValue As Integer, _
       ByVal secondValue As Integer)

       Hour = hourValue     ' invokes Hour set accessor
       Minute = minuteValue ' invokes Minute set accessor
       Second = secondValue ' invokes Second set accessor
    End Sub ' New

    Public Property Hour() As Integer

       ' return value
       Get
          Return m_intHour
       End Get ' Get accessor

       ' set m_intHour value
       Set(ByVal Value As Integer)

          ' if hour value is valid
          If (Value < 10) Then
             m_intHour = Value
          Else
             m_intHour = 0 ' set invalid input to 0
          End If

       End Set ' Set accessor

    End Property ' Hour

    ' property Minute
    Public Property Minute() As Integer

       ' return m_intMinute value
       Get
          Return m_intMinute
       End Get ' end of Get accessor

       ' set m_intMinute value
       Set(ByVal Value As Integer)

          ' if minute value entered is valid
          If (Value < 60) Then
             m_intMinute = Value
          Else
             m_intMinute = 0 ' set invalid input to 0
          End If

       End Set ' end of Set accessor

    End Property ' Minute

    ' property Second
    Public Property Second() As Integer

```

```
        ' return m_intSecond value
        Get
           Return m_intSecond
        End Get ' Get accessor

        ' set m_intSecond value
        Set(ByVal Value As Integer)

           ' if second value entered are invalid
           If (Value < 60) Then
              m_intSecond = Value
           Else
              m_intSecond = 0 ' set invalid input to 0
           End If

        End Set ' Set accessor

     End Property ' Second

  End Class ' Time
```

```
  ' Exercise 19.12 Solution
  ' MicrowaveOven.vb

  Public Class FrmMicrowaveOven
     Inherits System.Windows.Forms.Form

     ' contains time entered as a String
     Private m_strTime As String = ""

     ' holds the time
     Private m_objTime As Time

     ' Windows Form Designer generated code

     ' event handler appends 1 to time string
     Private Sub btnOne_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnOne.Click

        Beep()              ' sound beep
        m_strTime &= "1"    ' append digit to time input
        DisplayTime()       ' display time input properly
     End Sub ' btnOne_Click

     ' event handler appends 2 to time string
     Private Sub btnTwo_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnTwo.Click

        Beep()              ' sound beep
        m_strTime &= "2"    ' append digit to time input
        DisplayTime()       ' display time input properly
     End Sub ' btnTwo_Click

     ' event handler appends 3 to time string
     Private Sub btnThree_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnThree.Click

        Beep()              ' sound beep
        m_strTime &= "3"    ' append digit to time input
        DisplayTime()       ' display time input properly
```

```
    End Sub ' btnThree_Click

    ' event handler appends 4 to time string
    Private Sub btnFour_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnFour.Click

       Beep()              ' sound beep
       m_strTime &= "4"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnFour_Click

    ' event handler appends 5 to time string
    Private Sub btnFive_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnFive.Click

       Beep()              ' sound beep
       m_strTime &= "5"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnFive_Click

    ' event handler appends 6 to time string
    Private Sub btnSix_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnSix.Click

       Beep()              ' sound beep
       m_strTime &= "6"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnSix_Click

    ' event handler appends 7 to time string
    Private Sub btnSeven_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnSeven.Click

       Beep()              ' sound beep
       m_strTime &= "7"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnSeven_Click

    ' event handler appends 8 to time string
    Private Sub btnEight_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnEight.Click

       Beep()              ' sound beep
       m_strTime &= "8"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnEight_Click

    ' event handler appends 9 to time string
    Private Sub btnNine_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnNine.Click

       Beep()              ' sound beep
       m_strTime &= "9"    ' append digit to time input
       DisplayTime()       ' display time input properly
    End Sub ' btnNine_Click

    ' event handler appends 0 to time string
    Private Sub btnZero_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnZero.Click

       Beep()              ' sound beep
```

```
      m_strTime &= "0"    ' append digit to time input
      DisplayTime()       ' display time input properly
   End Sub ' btnZero_Click

   ' event handler starts the microwave oven's cooking process
   Private Sub btnStart_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnStart.Click

      Dim intHour As Integer
      Dim intSecond As Integer
      Dim intMinute As Integer

      ' pad extra spaces in m_strTime with zero
      m_strTime = m_strTime.PadLeft(5, Convert.ToChar("0"))

      ' extract seconds, minutes and hours
      intSecond = Convert.ToInt32(m_strTime.Substring(3))
      intMinute = Convert.ToInt32(m_strTime.Substring(1, 2))
      intHour = Convert.ToInt32(m_strTime.Substring(0, 1))

      ' create Time object to contain time entered by user
      m_objTime = New Time(intHour, intMinute, intSecond)

      ' display the time
      lblDisplay.Text = String.Format("{0:D1}:{1:D2}:{2:D2}", _
         m_objTime.Hour, m_objTime.Minute, m_objTime.Second)

      m_strTime = "" ' clear m_strTime for future input
      tmrClock.Enabled = True            ' begin timer
      pnlWindow.BackColor = Color.Yellow ' turn "light" on
   End Sub ' btnStart_Click

   ' event handler to clear input
   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      ' reset each method or variable to its initial setting
      lblDisplay.Text = "Microwave Oven"
      m_strTime = ""
      m_objTime = New Time(0, 0, 0)
      tmrClock.Enabled = False           ' turn timer off
      pnlWindow.BackColor = pnlWindow.DefaultBackColor

      btnStart.Enabled = True ' enable Start Button
   End Sub ' btnClear_Click

   ' method to display formatted time in timer window
   Private Sub DisplayTime()

      Dim intSecond As Integer
      Dim intMinute As Integer
      Dim intHour As Integer
      Dim strDisplay As String ' the display String

      ' disallow extra input if number of hours > 9
      If m_strTime.Length > 5 Then

         ' take only the first 5 digits
         m_strTime = m_strTime.Substring(0, 5)
      End If

```

```
      ' pad the empty spaces of the display String with "0"
      strDisplay = m_strTime.PadLeft(5, Convert.ToChar("0"))

      ' extract seconds and minutes and hours
      intSecond = Convert.ToInt32(strDisplay.Substring(3))
      intMinute = Convert.ToInt32(strDisplay.Substring(1, 2))
      intHour = Convert.ToInt32(strDisplay.Substring(0, 1))

      ' display number of hours, minutes, and seconds
      lblDisplay.Text = String.Format("{0:D1}:{1:D2}:{2:D2}", _
         intHour, intMinute, intSecond)
   End Sub ' DisplayTime

   ' event handler displays new time each second
   Private Sub tmrClock_Tick(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles tmrClock.Tick

      ' perform countdown, subtract one second
      If m_objTime.Second > 0 Then
         m_objTime.Second -= 1
      ElseIf m_objTime.Minute > 0 Then
         m_objTime.Minute -= 1 ' subtract one minute
         m_objTime.Second = 59 ' reset seconds for new minute
      ElseIf m_objTime.Hour > 0 Then
         m_objTime.Hour -= 1   ' subtract one hour
         m_objTime.Minute = 59 ' reset minutes for new hour
         m_objTime.Second = 59 ' reset seconds for new minute
      Else                                 ' no more seconds
         tmrClock.Enabled = False          ' stop the timer
         Beep()                            ' sound beep
         lblDisplay.Text = "Done!"         ' display "Done" message
         pnlWindow.BackColor = pnlWindow.DefaultBackColor

         Return
      End If

      ' refresh the display time
      lblDisplay.Text = String.Format("{0:D1}:{1:D2}:{2:D2}", _
         m_objTime.Hour, m_objTime.Minute, m_objTime.Second)
   End Sub ' tmrTimer_Tick

End Class ' FrmMicrowaveOven
```

19.13 (*Account Information Application*) The local bank wants you to create an application that will allow them to view their clients' information. The interface is created for you; you need to implement the class (Fig. 19.51). Once the application is completed, the bank manager should be able to click the **Next** or **Previous** `Button` to run through each client's information. The information is stored in four arrays containing first names, last names, account numbers and account balances.

Figure 19.51 Account Information application GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial19\Exercises\AccountInformation` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Determining variables for the class.*** Examine the code from `AccountInformation.vb`, including all the properties that the `Client` object uses to retrieve the information.

d) ***Creating the `Client` class.*** Create a new class, and call it `Client`. Add this class to the project. Define four `Private` instance variables to represent each property value, to ensure that each `Client` object contains all the required information about each client. Use those variables to define a constructor.

e) ***Defining each property.*** Each `Private` variable should have a corresponding property, allowing the user to set or get each `Private` variable's value.

f) ***Adding more information.*** In the `FrmAccountInformation_Load` event handler, add two more accounts. Include names, account numbers, and balances for each corresponding array.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter information for multiple accounts. Click the **Previous** and **Next** `Buttons` to ensure that each account's information is stored properly.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 19.13 Solution
' Client.vb
' represent client balance information

Public Class Client

   Private m_strFirstName As String ' first name
   Private m_strLastName As String  ' last name
   Private m_intAccount As Integer  ' account number
   Private m_intbalance As Decimal  ' account balance

   ' Client constructor, first and last names, account number
   ' and account balance supplied
   Public Sub New(ByVal strFirstName As String, _
      ByVal strLastName As String, ByVal intAccount As Integer, _
      ByVal decBalance As Decimal)

      First = strFirstName
      Last = strLastName
      Account = intAccount
      Balance = decBalance
```

```
   End Sub ' New

   ' property First
   Public Property First() As String

      ' return m_strFirstName
      Get
         Return m_strFirstName ' return first name
      End Get ' end of Get accessor

      ' set first name
      Set(ByVal Value As String)
         m_strFirstName = Value
      End Set ' end of Set accessor

   End Property ' First

   ' property Last
   Public Property Last() As String

      ' return m_strLastName
      Get
         Return m_strLastName ' return last name
      End Get ' end of Get accessor

      ' set last name
      Set(ByVal Value As String)
         m_strLastName = Value
      End Set ' end of Set accessor

   End Property ' Last

   ' account number
   Public Property Account() As Integer

      ' return m_intAccount
      Get
         Return m_intAccount ' return account number
      End Get ' end of Get accessor

      ' set account number
      Set(ByVal intAccountValue As Integer)

         ' account number can not be negative
         If intAccountValue > -1 Then
            m_intAccount = intAccountValue
         Else
            m_intAccount = 0 ' default to zero
         End If

      End Set ' end of Set accessor

   End Property ' Account

   ' account balance
   Public Property Balance() As Decimal

      ' return m_intbalance
      Get
         Return m_intbalance ' return account balance
      End Get ' end of Get accessor
```

```

      ' set the account balance
      Set(ByVal Value As Decimal)
         m_intbalance = Value
      End Set ' end of Set accessor

   End Property ' Balance

End Class ' Client
```

```
' Exercise 19.13 Solution
' AccountInformation.vb

Public Class FrmAccountInformation
   Inherits System.Windows.Forms.Form

   Private m_objName(9) As Client       ' Client object
   Private m_intPosition As Integer = 0 ' current account

   ' Windows Form Designer generated code

   ' create array of Client objects
   Private Sub FrmAccountInformation_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      Dim intCount As Integer ' counter variable

      ' array of first names
      Dim strFirstName() As String = _
         New String() {"John", "Sarah", "Jack", "Adam", "Diane", _
            "David", "Kristin", "Jennifer", "", ""}

      ' array of last names
      Dim strLastName() As String = _
         New String() {"Blue", "White", "Red", "Brown", _
            "Yellow", "Black", "Green", "Orange", "", ""}

      ' array of account numbers
      Dim intAccount() As Integer = _
         New Integer() {1234652, 1234666, 1234678, 1234681, _
            1234690, 1234770, 1234787, 1234887, 0, 0}

      ' array of account balances
      Dim decBalance() As Decimal = _
         New Decimal() {Convert.ToDecimal(1000.78), _
            Convert.ToDecimal(2056.24), Convert.ToDecimal(978.65), _
            Convert.ToDecimal(990.0), Convert.ToDecimal(432.75), _
            Convert.ToDecimal(780.78), Convert.ToDecimal(4590.63), _
            Convert.ToDecimal(7910.11), 0, 0}

      ' loop and create 10 Client objects
      For intCount = 0 To m_objName.GetUpperBound(0)

         ' create new object and store into Client array
         m_objName(intCount) = New Client(strFirstName(intCount), _
            strLastName(intCount), intAccount(intCount), _
            decBalance(intCount))
      Next

```

```
    End Sub ' FrmAccountInformation_Load

    ' display next object
    Private Sub btnNext_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnNext.Click

       m_intPosition += 1 ' increment position

       ' if position is last (top) object
       If m_intPosition > m_objName.GetUpperBound(0) Then
          m_intPosition = 0    ' set to first position in array
          DisplayInformation() ' display information
       Else
          DisplayInformation()
       End If

    End Sub ' btnNext_Click

    ' display previous object
    Private Sub btnPrevious_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnPrevious.Click

       m_intPosition -= 1 ' decrement position

       ' if position is last (bottom) object
       If m_intPosition < 0 Then

          ' set to last position in array
          m_intPosition = m_objName.GetUpperBound(0)
          DisplayInformation()
       Else
          DisplayInformation() ' display information
       End If

    End Sub ' btnPrevious_Click

    ' display information
    Private Sub DisplayInformation()

       ' use m_intPosition as index for each object
       txtFirst.Text = m_objName(m_intPosition).First
       txtLast.Text = m_objName(m_intPosition).Last
       txtAccount.Text = _
          Convert.ToString(m_objName(m_intPosition).Account)

       ' format as currency
       txtBalance.Text = _
          String.Format("{0:C}", m_objName(m_intPosition).Balance)

    End Sub ' DisplayInformation

End Class  ' FrmAccountInformation
```

What does this code do?

19.14 What does the following code do? The first code listing contains the definition of class `Shape`. Each `Shape` object represents a closed shape with a number of sides. The second code listing contains a method (`Mystery`) created by a client of class `Shape`. What does this method do?

```
Public Class Shape

   Private m_intSides As Integer

   ' constructor with number of sides
   Public Sub New(ByVal intSides As Integer)
      Side = intSides
   End Sub ' New

   ' set and get side value
   Public Property Side() As Integer

      ' return m_intSides
      Get
         Return m_intSides
      End Get ' end of Get accessor

      ' set m_intSides
      Set(ByVal Value As Integer)

         If Value > 0 Then
            m_intSides = Value
         Else
            m_intSides = 0
         End If

      End Set ' end of Set accessor

   End Property ' Side

End Class ' Shape
```

```
Public Function Mystery(ByVal objShape As Shape) As String
   Dim strShape As String

   ' determine case with objShape.Side
   Select Case objShape.Side

      Case Is < 3
         strShape = "Not a Shape"

      Case 3
         strShape = "Triangle"

      Case 4
         strShape = "Square"

      Case Else
         strShape = "Polygon"

   End Select

   Return strShape
End Function ' Mystery
```

Answer: The `Shape` class defines a shape with a given number of sides. Method `Mystery` determines the shape of it's `Shape` and returns the name of the shape. Method `Mystery` takes a `Shape` object as an argument.

What's wrong with this code?

19.15 Find the error(s) in the following code. The following method should create a new `Shape` object with `intNumberSides` sides. Assume the `Shape` class from Exercise 19.14.

```
Private Sub ManipulateShape(ByVal intNumberSides As Integer)
   Dim objShape As Shape = New Shape(3)

   Shape.m_intSides = intNumberSides
End Sub ' ManipulateShape
```

Answer: The method should create a `Shape` object with `intNumberSides` sides—not a `Shape` object with 3 sides. Also, a `Private` variable (`m_intSides`) cannot be accessed from outside the class. The correct code is as follows:

```
Private Sub ManipulateShape(ByVal intNumberSides As Integer)

   Dim objShape As Shape = New Shape(intNumberSides)

   ' or

   'Dim objShape As Shape
   'objShape.Side = intNumberSides
End Sub ' ManipulateShape
```

Using the Debugger

19.16 (***View Name Application***) The **View Name** application allows the user to enter the user's first and last name. When the user clicks the **View Name** `Button`, a `MessageBox` that displays the user's first and last name appears. The application creates an instance of `Class Name`. This class uses its property definitions to set the first-name and last-name instance variables. Copy the `Names` directory from `C:\Examples\Tutorial19\Exercises\Debugger` to your `Debugger` directory. Open and run the application. While testing your application, you noticed that the `MessageBox` did not display the correct output. Use the debugger to find the logic error(s) in the application. The application with the correct output is displayed in Fig. 19.52.

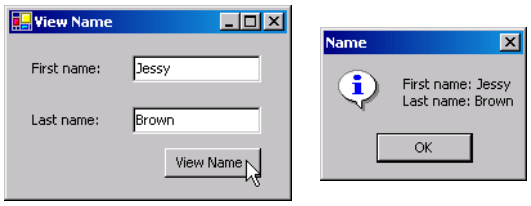


Figure 19.52 **View Name** application with correct output.

Answer:

```
' Exercise 19.16 Solution
' Name.vb
' Name class definition

Public Class Name

   Private m_strFirstName As String
   Private m_strLastName As String

   ' Name constructor, first and last names supplied
   Public Sub New(ByVal strFirstName As String, _
      ByVal strLastName As String)

```

```
      First = strFirstName
      Last = strLastName
   End Sub ' New

   ' property First
   Public Property First() As String

      ' return first name
      Get
         Return m_strFirstName
      End Get

      ' set first name
      Set(ByVal Value As String)

         m_strFirstName = Value
      End Set

   End Property ' First

   ' property Last
   Public Property Last() As String

      ' return last name
      Get
         Return m_strLastName
      End Get

      ' set last name
      Set(ByVal Value As String)

         m_strLastName = Value
      End Set

   End Property ' Second

End Class ' Name
```

Original line assigned `m_strFirstName` to `Value`

Original line assigned data to `m_strFirstName`, rather than to `m_strLastName`

```
' Exercise 19.16 Solution
' ViewName.vb

Public Class FrmViewName
   Inherits System.Windows.Forms.Form

   Private m_objName As Name ' Name object

   ' Windows Form Designer generated code

   Private Sub btnView_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnView.Click

      Dim strOutput As String ' holds first name and last name

      ' create new Name
      m_objName = New Name(txtFirst.Text, txtLast.Text)

      ' assign user's name to strOutput
      strOutput = "First name: " & m_objName.First & _
         ControlChars.CrLf & "Last name: " & m_objName.Last

```

```
        ' output name
        MessageBox.Show(strOutput, "Name", _
           MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Sub ' btnView_Click

End Class  ' FrmViewName
```

Programming Challenge

19.17 (*DVD Burner Application*) Create an application that simulates a DVD burner. Users create a DVD with their choice of title and bonus materials. The GUI is provided for you (Fig. 19.53). You will create a class (`DVDObject`) to represent the DVD object and another class (`Bonus`) to represent bonus materials for a DVD object.

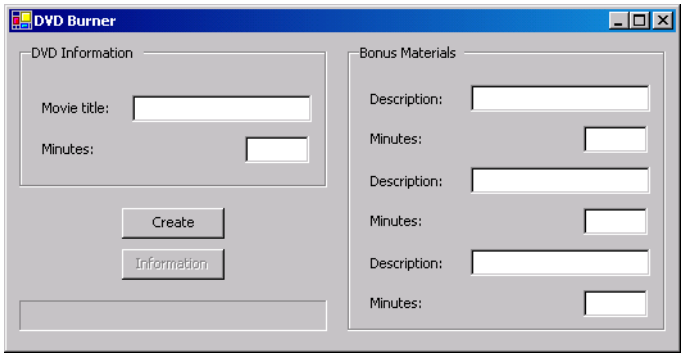


Figure 19.53 DVD Burner application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial19\Exercises\DVDBurner` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `DVDBurner.sln` in the `DVDBurner` directory to open the application.

c) ***Creating the bonus-material object.*** Create a class, and name it `Bonus`. The class's objects will each represent one bonus-material item on the DVD. Each `Bonus` object should have a name (description) and a length (in minutes). Use this tutorial's `Time` class as your guide in creating the properties for the name and length of each bonus material.

d) ***Creating the DVD class.*** Create a class, and name it `DVDObject`. This class contains the movie title and the length of the movie. The class should also include an array of three `Bonus` items.

e) ***Creating the necessary variables.*** Before you define the **Create** `Button`'s event handler, create an `DVDObject` class instance variable. Inside the **Create** `Button`'s event handler, create the necessary variables to store the information from the `TextBoxes` on the GUI. Also, this is where you need to create the array of `Bonus` objects to store the bonus materials.

f) ***Adding bonus-material information.*** Add the description and length of each bonus item to the `Bonus` array you created from the previous step.

g) ***Creating a DVD object.*** Use information about the movie, its title, length and the array of bonus materials to make your DVD object.

h) ***Displaying the output.*** The **Information** `Button`'s `Click` event is already defined for you. Locate the event handler, add a `String` containing the complete information on the DVD object that you created earlier and display this `String` to a `MessageBox`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter information for several DVDs. After information is entered for each, click the **Create** `Button`. Then, click the **Information** `Button` and verify that the information being displayed is correct for your newly created DVD.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 19.17 Solution
' Bonus.vb
' represent bonus items on a DVD

Public Class Bonus

   ' name of bonus material
   Private m_strName As String

   ' length of the bonus material
   Private m_intItemLength As Integer

   ' Bonus constructor, name and item length
   Public Sub New(ByVal nameValue As String, _
      ByVal lengthValue As Integer)

      Name = nameValue
      ItemLength = lengthValue
   End Sub ' New

   ' set or get name of bonus material
   Public Property Name() As String

      ' return m_strName
      Get
         Return m_strName ' return name
      End Get ' end of Get accessor

      ' set description name
      Set(ByVal Value As String)

         ' if description is greater than 20 characters
         If Value.Length > 20 Then

            ' take first 20 characters
            Value = Value.Substring(0, 20)
            m_strName = Value
         Else

            ' set name
            m_strName = Value
         End If

      End Set ' end of Set accessor

   End Property ' Name

   ' set or get amount of items
   Public Property ItemLength() As Integer

      ' return m_intItemLength
      Get
         Return m_intItemLength ' return length
      End Get ' end of Get accessor

      ' set minute value
      Set(ByVal Value As Integer)

         ' make sure minute is non-negative
```

```
         If Value > 0 Then
            m_intItemLength = Value
         Else
            m_intItemLength = 0
         End If

      End Set ' end of Set accessor

   End Property ' ItemLength

End Class ' Bonus
```

```
' Exercise 19.17 Solution
' DVDObject.vb
' represent items on a DVD

Public Class DVDObject
   Private m_strMovieTitle As String      ' name of movie
   Private m_objBonusMaterial() As Bonus  ' array of Bonus objects
   Private m_intMovieLength As Integer    ' length of movie

   ' DVDObject constructor
   Public Sub New(ByVal nameValue As String, _
      ByVal bonusValue() As Bonus, _
      ByVal movieLengthValue As Integer)

      ' call property to set values
      MovieTitle = nameValue
      MovieLength = movieLengthValue

      ' assign bonusValue array to m_objBonusMaterial
      m_objBonusMaterial = bonusValue

   End Sub ' New

   ' set or get movie title
   Public Property MovieTitle() As String

      ' return m_strMovieTitle
      Get
         Return m_strMovieTitle ' return movie title
      End Get ' end of Get accessor

      ' set movie title
      Set(ByVal Value As String)

         ' if title is greater than 20 characters
         If Value.Length > 20 Then

            ' take first 20 characters
            Value = Value.Substring(0, 20)
            m_strMovieTitle = Value
         Else

            ' set title
            m_strMovieTitle = Value
         End If

      End Set ' end of Set accessor

```

```
   End Property ' MovieTitle

   ' ReadOnly get property
   Public ReadOnly Property BonusMaterials() As String

      ' display bonus material information
      Get

         ' information on bonus materials
         Dim strBonusMaterial As String = ""
         Dim intCount As Integer ' counter variable

         ' loop through each bonus material
         For intCount = 0 To (m_objBonusMaterial.Length - 1)

            ' format String to contain minutes and seconds
            strBonusMaterial &= _
               m_objBonusMaterial(intCount).Name() & ": " & _
               m_objBonusMaterial(intCount).ItemLength & _
               " minute(s)." & ControlChars.CrLf
         Next

         Return strBonusMaterial ' return String
      End Get ' end of Get accessor

   End Property ' BonusMaterials

   ' set and get movie length
   Public Property MovieLength() As Integer

      ' return m_intMovieLength
      Get
         Return m_intMovieLength ' return length of movie
      End Get ' end of Get accessor

      ' set minutes for movie
      Set(ByVal Value As Integer)

         ' make sure minute is nonegative
         If Value > 0 Then
            m_intMovieLength = Value
         Else
            m_intMovieLength = 0
         End If

      End Set ' end of Set accessor

   End Property ' MovieLength

End Class ' DVDObject
```

```
' Exercise 19.17 Solution
' DVDBurner.vb

Public Class FrmDVDBurner
   Inherits System.Windows.Forms.Form

   Private m_objDVD As DVDObject  ' create instance variable

   ' Windows Form Designer generated code
```

```

    Private Sub btnCreate_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnCreate.Click

       Dim objBonus(2) As Bonus      ' array of Bonus
       Dim intBonusLength As Integer ' bonus material minutes

       ' store movie name
       Dim strMovieTitle As String = txtTitle.Text

       ' movie minutes
       Dim intMovieMinutes As Integer = _
          Convert.ToInt32(Val(txtMovieMinute.Text))

       ' bonus material description (name)
       Dim strBonus1 As String = txtDescription1.Text
       Dim strBonus2 As String = txtDescription2.Text
       Dim strBonus3 As String = txtDescription3.Text

       ' store minutes from TextBox
       intBonusLength = Convert.ToInt32(Val(txtMinutes1.Text))

       ' add bonus material name and time to array
       objBonus(0) = New Bonus(strBonus1, intBonusLength)

       ' store minutes from TextBox
       intBonusLength = Convert.ToInt32(Val(txtMinutes2.Text))

       ' add bonus material name and time to array
       objBonus(1) = New Bonus(strBonus2, intBonusLength)

       ' store minutes from TextBox
       intBonusLength = Convert.ToInt32(Val(txtMinutes3.Text))

       ' add bonus material name and time to array
       objBonus(2) = New Bonus(strBonus3, intBonusLength)

       ' call constructor for new object
       m_objDVD = _
          New DVDObject(strMovieTitle, objBonus, intMovieMinutes)

       ' let the user know about progress
       lblDisplay.Text = "Your DVD was created successfully!"

       ' enable Information Button
       btnInformation.Enabled = True

    End Sub ' btnCreate_Click

    ' display information about DVD
    Private Sub btnInformation_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnInformation.Click

       Dim strInformation As String ' output String

       lblDisplay.Text = ""         ' clear Label

       ' add title and length to String
       ' add information about bonus materials
       strInformation = m_objDVD.MovieTitle & ": " & _
          m_objDVD.MovieLength & " minute(s)" & _
```

```
            ControlChars.CrLf & "Bonus Materials:" & _
            ControlChars.CrLf & m_objDVD.BonusMaterials

         ' display output in a MessageBox
         MessageBox.Show(strInformation, "DVD Description", _
            MessageBoxButtons.OK, MessageBoxIcon.Information)
      End Sub ' btnInformation_Click

   End Class ' FrmDVDBurner
```

TUTORIAL 20

Shipping Hub Application

Introducing Collections, the `For Each...Next` Statement and Access Keys

Solutions

Instructor's Manual Exercise Solutions Tutorial 20

MULTIPLE-CHOICE QUESTIONS

20.1 _____ are specifically designed to store groups of values.

a) Collections b) Properties
c) Accessors d) None of the above.

20.2 The _____ key provides a quick and convenient way to navigate through controls on a `Form`.

a) *Tab* b) *Enter*
c) *Caps Lock* d) *Alt*

20.3 An `ArrayList` differs from an array in that an `ArrayList` can _____.

a) store objects of any type b) resize dynamically
c) be accessed programmatically d) All of the above.

20.4 The element in a `For Each…Next` statement _____.

a) must be of type `Integer`
b) must be of (or convertible to) the same type as the collection or array type
c) must be of type `ArrayList` d) None of the above.

20.5 The control that receives the focus the first time *Tab* is pressed has a `TabIndex` property set to _____.

a) `First` b) `0`
c) `Next` d) `1`

20.6 Users should be able to use the *Tab* key to transfer the focus to _____.

a) only `Buttons` b) only `TextBoxes`
c) only controls that have an `AcceptTab` property
d) only the controls that receive user input

20.7 To ensure that the proper controls obtain the focus when the *Tab* key is pressed, use the _____.

a) `TabIndex` property b) `TabStop` and `TabIndex` properties
c) `TabStop` property d) `Focus` property

20.8 To add a value to the end of an `ArrayList`, call the _____ method.

a) `Add` b) `AddToEnd`
c) `AddAt` d) `InsertAt`

20.9 To remove a value from a specific index in the `ArrayList`, use method _____.

a) `Remove` b) `RemoveAt`
c) `Delete` d) `DeleteAt`

20.10 To display an ampersand character on a control, type a _____ in its `Text` property.

a) `&_` b) `&`
c) `&&` d) `_&`

Answers: 20.1) a. 20.2) a. 20.3) b. 20.4) b. 20.5) b. 20.6) d. 20.7) b. 20.8) a. 20.9) b. 20.10) c.

EXERCISES

20.11 (***Modified Salary Survey Application***) Modify the **Salary Survey** application you created in Exercise 17.12 by using a `For Each…Next` loop to replace the `For…Next` loop that is used in Tutorial 17 (Fig. 20.27).

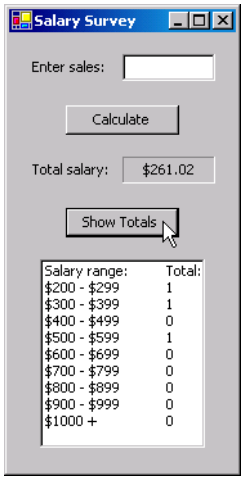


Figure 20.27 Modified **Salary Survey** GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial20\Exercises\SalarySurveyModified` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SalarySurveyModified.sln` in the `SalarySurvey` directory to open the application.

c) ***Locating the event handler.*** In **Design View**, double click the **Show Totals** `Button` to bring up the event handler. The code to handle the `Click` event should include two statements, one to clear the items in the `ListBox` and the other to add a header.

d) ***Creating a counter variable.*** The `For Each...Next` loop allows you to loop through each element in a specified collection. The `For...Next` loop from Exercise 17.12 handles the `String` (`m_strSalaryRanges`) and `Integer` (`m_intSalaries`) arrays. This presents a problem. You cannot loop through both of these arrays using the same element reference. (One is an `Integer`, and the other is a `String`.) To handle this you need to create a common counter variable, one that you will use to loop through the indices of both arrays. This is possible because the lengths of both arrays are the same.

e) ***Adding an element reference.*** It does not matter which array you decide to use in this exercise, because these arrays are of the same length. Declare an element reference with the correct data type.

f) ***Create the `For Each...Next` loop.*** Use the new element reference that you have created along with the array of your choice to create the `For Each...Next` loop statement.

g) ***Adding text to the `ListBox`.*** Adding the statement to output to the `ListBox` is exactly the same as the one from Exercise 17.12. The only difference will be the name of the counter variable that you decide to use.

h) ***Increment the counter variable.*** To successfully loop through both arrays and output the data, you need to increment the counter variable. This ensures that the proper data is added to the `ListBox` through each iteration.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter several sales amounts using the **Calculate** `Button`. Click the **Show Totals** `Button` and verify that the proper amounts are displayed for each salary range, based on the salaries calculate from your input.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 20.11 Solution
' SalarySurvey.vb

```

```
Public Class FrmSalarySurvey
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' salary ranges
   Private m_strSalaryRanges As String() = New String() { _
      "$200 - $299", "$300 - $399", "$400 - $499", _
      "$500 - $599", "$600 - $699", "$700 - $799", _
      "$800 - $899", "$900 - $999", "$1000 +  "}

   ' number of employees in each salary range
   Private m_intSalaries As Integer() = New Integer( _
      m_strSalaryRanges.GetUpperBound(0)) {}

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnCalculate.Click

      ' obtain total sales
      Dim decSales As Decimal = Convert.ToDecimal( _
         Val(txtInputSales.Text))

      ' employee's base salary
      Dim decTotalSalary As Decimal = 200

      ' add commision to total salary
      decTotalSalary += Convert.ToDecimal(decSales * 0.09)

      ' display salary in a Label
      lblTotalSalary.Text = String.Format("{0:C}", decTotalSalary)

      ' increment the correct counter in array m_intSalaries
      Select Case decTotalSalary

         Case Is < 300
            m_intSalaries(0) += 1

         Case Is < 400
            m_intSalaries(1) += 1

         Case Is < 500
            m_intSalaries(2) += 1

         Case Is < 600
            m_intSalaries(3) += 1

         Case Is < 700
            m_intSalaries(4) += 1

         Case Is < 800
            m_intSalaries(5) += 1

         Case Is < 900
            m_intSalaries(6) += 1

         Case Is < 1000
            m_intSalaries(7) += 1

         Case Is >= 1000
            m_intSalaries(8) += 1
```

```
65
66          End Select
67
68          ' clear TextBox
69          txtInputSales.Clear()
70       End Sub ' btnCalculate_Click
71
72       ' handles click event for btnShowTotals Button
73       Private Sub btnShowTotals_Click(ByVal sender As System.Object, _
74          ByVal e As System.EventArgs) Handles btnShowTotals.Click
75
76          Dim intCounter As Integer = 0 ' counter variable
77          Dim strRange As String  ' each range in m_strSalaryRanges
78
79          ' clear all items in the ListBox
80          lstSalaryTotals.Items.Clear()
81
82          ' add header to ListBox
83          lstSalaryTotals.Items.Add("Salary range:" & _
84             ControlChars.Tab & "Total:")
85
86          ' add each element from two arrays to the ListBox
87          For Each strRange In m_strSalaryRanges
88             lstSalaryTotals.Items.Add(m_strSalaryRanges(intCounter) & _
89                ControlChars.Tab & m_intSalaries(intCounter))
90             intCounter += 1 ' increment the counter
91          Next
92
93       End Sub ' btnShowTotals_Click
94
95    End Class ' FrmSalarySurvey
```

20.12 (***Modified Shipping Hub Application***) Modify the **Shipping Hub** application created in this tutorial, so that the user can double click a package in the `lstPackages` `ListBox`. When a package number is double clicked, the package's information should be displayed in a `MessageBox` (Fig. 20.28).

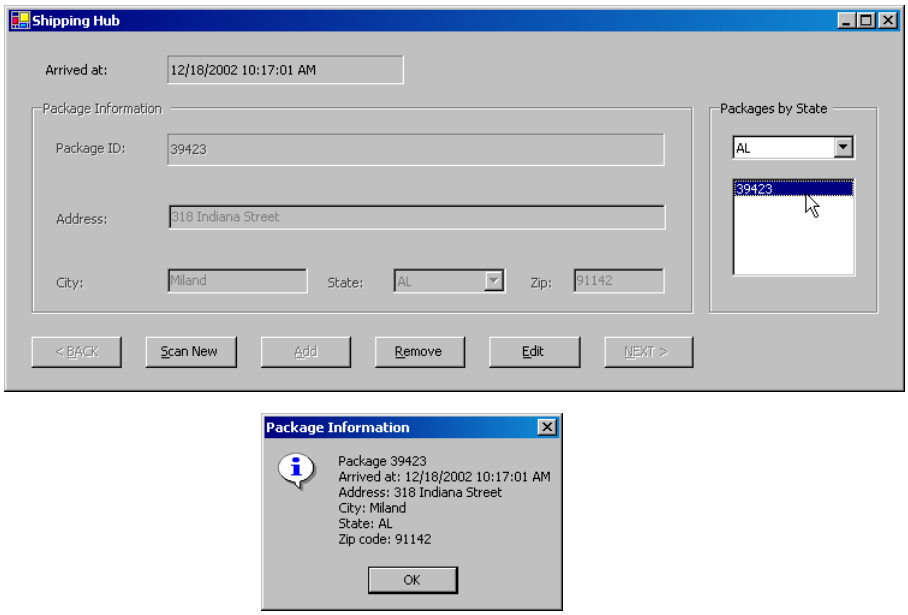


Figure 20.28 **Modified Shipping Hub** application GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial20\Exercises\ShippingHubModified` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ShippingHubModified.sln` in the `ShippingHubModified` directory to open the application.

c) ***Viewing the event handler***. Click **ShippingHub.vb** in the **Solution Explorer** and select **View > Code**. Scroll to the end of code listing to locate the `ListBox`'s `DoubleClick` event handler.

d) ***Initializing necessary variables***. To loop through the packages in the `ArrayList` of `Packages`, you need to create a reference of type `Package`. It is also helpful to create a `String` variable to store the information about the given package. Write code in the `DoubleClick` event handler to declare the `String strPackage`. A `ListBox`'s `DoubleClick` event is raised when the control is double clicked.

e) ***Check whether the user has selected a valid item***. To determine whether the user has selected a valid item (and not an empty element in the `ListBox`), write an `If…Then` statement to make sure that the `ListBox` is not empty when the user selected an item. [*Hint*: A `SelectedIndex` value of `-1` means that no item is currently selected.]

f) ***Writing a `For Each…Next` loop***. Use the Package reference you declared in *Step c* to create a `For Each…Next` loop with the `m_objList` collection.

g) ***Determining whether the current selected package is correct***. Insert an `If…Then` statement to determine whether the current object that is selected from the `m_objList` collection matches the selected item from the `ListBox`. Because the packages are listed in the `ListBox` by their package number, use that information in your `If…Then` statement. Once the correct package is matched, store that package's information in the `strPackage String`.

h) ***Inserting the `Else` statement***. Make sure to notify the user if an invalid item has been selected from the `ListBox`. If this occurs, add a message to the `strPackage String` that will be displayed in the `MessageBox`.

i) ***Displaying the `MessageBox`***. Call the `MessageBox`'s `Show` method to display the text you have added to the `strPackage String`. This displays either the information for the package they have selected or the message telling them they have selected an invalid package.

j) ***Running the application.*** Select **Debug > Start** to run your application. Add several packages. In the **Packages by State** `GroupBox`, select a state for which there are packages being sent. Double click one of the packages listed in the **Packages by State** `ListBox`, and verify that the correct information is displayed in a `MessageBox`.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 20.12 Solution
' ShippingHub.vb

Public Class FrmShippingHub
   Inherits System.Windows.Forms.Form

   Private m_objList As Collections.ArrayList ' list of packages
   Private m_objPackage As Package   ' current package
   Private m_intPosition As Integer  ' position of current package
   Private m_objRandom As Random     ' random number for package id
   Private m_intPackageID As Integer ' individual package number

   ' Windows Form Designer generated code

   ' Form Load event
   Private Sub FrmShippingHub_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
```

```
      Handles MyBase.Load

      m_intPosition = 0          ' set initial position to zero
      m_objRandom = New Random ' create new Random object
      m_intPackageID = m_objRandom.Next(1, 100000) ' new package ID

      ' show first state in ComboBox (using the Items property)
      cboState.Text = Convert.ToString(cboState.Items.Item(0))
      m_objList = New Collections.ArrayList ' list of packages
   End Sub ' FrmShippingHub_Load

   ' Scan New Button Click event
   Private Sub btnNew_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNew.Click

      m_intPackageID += 1 ' increment package id
      m_objPackage = New Package(m_intPackageID) ' create package

      ClearControls() ' clear fields
      lblPackageNumber.Text = _
         m_objPackage.PackageNumber.ToString ' package number
      lblArrivalTime.Text = _
         m_objPackage.ArrivalTime.ToString ' display arrival time

      ' only allow user to add package
      fraAddress.Enabled = True ' disable GroupBox and its controls
      SetButtons(False) ' enable/disable Buttons
      btnAdd.Enabled = True  ' enable Add Button
      btnNew.Enabled = False ' disable Scan New Button
      txtAddress.Focus() ' transfer the focus to txtAddress TextBox
   End Sub ' btnNew_Click

   ' Add Button Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      SetPackage() ' set Package properties from TextBoxes
      m_objList.Add(m_objPackage) ' add package to ArrayList

      fraAddress.Enabled = False ' disable GroupBox and its controls
      SetButtons(True) ' enable appropriate Buttons

      ' package cannot be added until Scan New is clicked
      btnAdd.Enabled = False ' disable Add Button

      ' if package's state displayed, add ID to ListBox
      If cboState.Text = cboViewPackages.Text Then
         lstPackages.Items.Add(m_objPackage.PackageNumber)
      End If

      cboViewPackages.Text = m_objPackage.State ' list packages
      btnNew.Enabled = True ' enable Scan New Button
   End Sub ' btnAdd_Click

   ' Back Button Click event
   Private Sub btnBack_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnBack.Click

      ' move backward one package in the list
      If m_intPosition > 0 Then
         m_intPosition -= 1
```

```
      Else ' wrap to end of list
         m_intPosition = m_objList.Count - 1
      End If

      LoadPackage() ' load package data from item in list
   End Sub ' btnBack_Click

   ' Next Button Click event
   Private Sub btnNext_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNext.Click

      ' move forward one package in the list
      If m_intPosition < m_objList.Count - 1 Then
         m_intPosition += 1
      Else
         m_intPosition = 0 ' wrap to beginning of list
      End If

      LoadPackage() ' load package data from item in list
   End Sub ' btnNext_Click

   ' Remove Button click event
   Private Sub btnRemove_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnRemove.Click

      ' remove ID from ListBox if state displayed
      If cboState.Text = cboViewPackages.Text Then
         lstPackages.Items.Remove(m_objPackage.PackageNumber)
      End If

      m_objList.RemoveAt(m_intPosition) ' remove package from list

      ' load next package in list if there is one
      If m_objList.Count > 0 Then

         ' if not at first position, go to previous one
         If m_intPosition > 0 Then
            m_intPosition -= 1
         End If

         LoadPackage() ' load package data from item in list
      Else
         ClearControls() ' clear fields
      End If

      SetButtons(True) ' enable appropriate Buttons
   End Sub ' btnRemove_Click

   ' Edit/Update Button Click event
   Private Sub btnEditUpdate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnEditUpdate.Click

      ' when Button reads "Edit", allow user to
      ' edit package information only
      If btnEditUpdate.Text = "&Edit" Then
         fraAddress.Enabled = True
         SetButtons(False)
         btnEditUpdate.Enabled = True

```

```
         ' change Button text from "Edit" to "Update"
         btnEditUpdate.Text = "&Update"
      Else

         ' when Button reads "Update" remove the old package
         ' data and add new data from TextBoxes
         SetPackage()
         m_objList.RemoveAt(m_intPosition)
         m_objList.Insert(m_intPosition, m_objPackage)

         ' display state in ComboBox
         cboViewPackages.Text = m_objPackage.State

         ' when done, return to normal operating state
         fraAddress.Enabled = False  ' disable GroupBox
         SetButtons(True) ' enable appropriate Buttons

         ' change Button text from "Update" to "Edit"
         btnEditUpdate.Text = "&Edit"
      End If

   End Sub ' btnEditUpdate_Click

   ' set package properties
   Private Sub SetPackage()
      m_objPackage.Address = txtAddress.Text
      m_objPackage.City = txtCity.Text
      m_objPackage.State = _
        Convert.ToString(cboState.SelectedItem)
      m_objPackage.Zip = Convert.ToInt32(Val(txtZip.Text))
   End Sub ' SetPackage

   ' load package information into Form
   Private Sub LoadPackage()

      ' retrieve package from list
      m_objPackage = CType(m_objList.Item(m_intPosition), _
         Package)

      ' display package data
      txtAddress.Text = m_objPackage.Address
      txtCity.Text = m_objPackage.City
      cboState.Text = m_objPackage.State
      txtZip.Text = m_objPackage.Zip.ToString("00000")
      lblArrivalTime.Text = _
         m_objPackage.ArrivalTime.ToString
      lblPackageNumber.Text = _
         m_objPackage.PackageNumber.ToString
   End Sub ' LoadPackage

   ' clear all the input controls on the Form
   Private Sub ClearControls()
      txtAddress.Clear()
      txtCity.Clear()
      txtZip.Clear()
      cboState.SelectedText = ""
      lblArrivalTime.Text = ""
      lblPackageNumber.Text = ""
   End Sub ' ClearControls

   ' enable/disable Buttons
```

```
   Private Sub SetButtons(ByVal blnState As Boolean)
      btnRemove.Enabled = blnState
      btnEditUpdate.Enabled = blnState
      btnNext.Enabled = blnState
      btnBack.Enabled = blnState

      ' disable navigation if not multiple packages
      If m_objList.Count < 2 Then
         btnNext.Enabled = False
         btnBack.Enabled = False
      End If

      ' if no items, disable Remove and Edit/Update Buttons
      If m_objList.Count = 0 Then
         btnEditUpdate.Enabled = False
         btnRemove.Enabled = False
      End If

   End Sub ' SetButtons

   ' event raised when user selects a new state in ComboBox
   Private Sub cboViewPackages_SelectedIndexChanged( _
      ByVal sender As System.Object, ByVal e As System.EventArgs) _
      Handles cboViewPackages.SelectedIndexChanged

      Dim objViewPackage As Package ' control variable package
      Dim strState As String = _
         Convert.ToString(cboViewPacakges.SelectedItem)

      lstPackages.Items.Clear() ' clear ListBox

      ' list all packages for current state in ListBox
      For Each objViewPackage In m_objList

         ' determine if state package is being shipped to
         ' matches the state selected in the ComboBox
         If objViewPackage.State = strState Then

            ' add package number to the ListBox
            lstPackages.Items.Add(objViewPackage.PackageNumber)
         End If

      Next

   End Sub ' cboViewPackages_SelectedIndexChanged

   ' display package information for selected package
   Private Sub lstPackages_DoubleClick(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles lstPackages.DoubleClick

      Dim objPackageInformation As Package ' temporary package
      Dim strPackage As String = "" ' String for package information

      ' check if the lstPackages ListBox is empty
      If lstPackages.SelectedIndex <> -1 Then

         For Each objPackageInformation In m_objList

            ' if the package currently in objPackageInformation
            ' matches the user's selected package
```

```
            If objPackageInformation.PackageNumber = _
               Convert.ToInt32(lstPackages.SelectedItem) Then
               strPackage &= ("Package " & _
                  objPackageInformation.PackageNumber & _
                  ControlChars.CrLf & _
                  "Arrived at: " & _
                  objPackageInformation.ArrivalTime & _
                  ControlChars.CrLf & _
                  "Address: " & _
                  objPackageInformation.Address & _
                  ControlChars.CrLf & _
                  "City: " & _
                  objPackageInformation.City & _
                  ControlChars.CrLf & _
                  "State: " & objPackageInformation.State & _
                  ControlChars.CrLf & _
                  "Zip code: " & _
                  objPackageInformation.Zip).ToString("00000")

            End If

         Next

      Else

         ' if the user select a blank item in the ListBox
         strPackage = "Please select a package"
      End If

      MessageBox.Show(strPackage, "Package Information", _
         MessageBoxButtons.OK, MessageBoxIcon.Information)

   End Sub ' lstPackages_DoubleClick

End Class ' FrmShippingHub
```

20.13 (***Controls Collection Application***) Visual Basic .NET provides many different types of collections. One such collection is the `Controls` collection, which is used to provide access to all of the controls on a `Form`. Create an application that uses the `Controls` collection and a `For Each…Next` loop to iterate through each control on the `Form`. As each control is encountered, add the control's name to a `ListBox`, and change the control's background color (in Fig. 20.29, `Color.Wheat` is used).

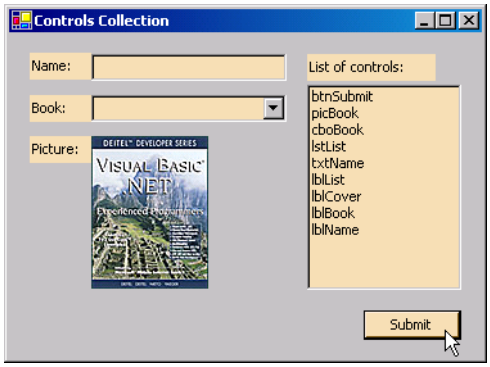


Figure 20.29 Controls Collection GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial20\Exercises\ControlsCollection` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ControlsCollection.sln` in the `ControlsCollection` directory to open the application.

c) ***Generating an event handler***. Switch to Design view. Double click the **Submit** `Button` in design view to create an event handler for the click event.

d) ***Declaring a control variable***. Declare a reference of type `Control`. This reference represents each element in the `For Each...Next` statement as it iterates through each `Control` on the `Form`.

e) ***Clearing the `ListBox`***. To ensure that the information in the `ListBox` is updated each time the **Submit** `Button` is clicked, clear the `ListBox` of all items.

f) ***Writing a `For Each...Next` loop***. To create the `For Each...Next` loop, use the control variable that you created to iterate through the `Form`'s `Controls` collection.

g) ***Adding each control's name to the `ListBox`***. Use the `ListBox`'s `Add` method to insert the name of each control on the `Form`. Recall that a control's `Name` property contains the name of the control.

h) ***Changing the control's background color***. Use the `Control`'s `BackColor` property to change the control's background color. Set the property to a new color using a member of the `Color` structure. [*Hint*: Type the word `Color` followed by the member-access operator to display a list of predefined colors using the *Intellisense* feature.] Note that the color of the `PictureBox` does not appear to change because its image displays in the control's foreground.

i) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Submit** `Button`. Verify that the controls' background colors change, and that all the controls are listed in the **List of controls:** `ListBox`.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 20.13 Solution
' ControlsCollection.vb

Public Class FrmControlsCollection
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Private Sub btnSubmit_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSubmit.Click

      Dim objControl As Control

      lstList.Items.Clear() ' clear ListBox

      ' iterate through controls collection
      For Each objControl In Controls

         ' list name of each control
         lstList.Items.Add(objControl.Name)

         ' change background color
         objControl.BackColor = Color.Wheat
      Next

   End Sub ' btnSubmit_Click

End Class ' FrmControlsCollection
```

What does this code do?

20.14 What is the result of the following code?

```
Dim intList As Collections.ArrayList
Dim intListItems As Integer
Dim strOutput As String

intList = New ArrayList
intList.Add(1)
intList.Add(3)
intList.Add(5)

For Each intListItems In intList
   strOutput &= (" " & intListItems.ToString)
Next

MessageBox.Show(strOutput, "Mystery", _
   MessageBoxButtons.OK, MessageBoxIcon.Information)
```

Answer: This code creates an `ArrayList` and adds to it the values `1`, `3` and `5`. The values are then appended to each other (separated by spaces) using a `For Each…Next` statement, and the result is displayed in a `MessageBox`.

What's wrong with this code?

20.15 This code should iterate through an array of `Packages` in `ArrayList objList` and print each package's number in `Label lblDisplay`. Find the error(s) in the following code.

```
Dim objValue As Collections.ArrayList

For Each objValue In objList
  lblDisplay.Text &= (" " & objValue.PackageNumber)
Next
```

Answer: The reference that specifies the element is of the wrong type to iterate through this list of `Packages`. The element reference should be of type `Package`. The corrected code is listed below:

```
Dim objValue As Package

For Each objValue In objList
  lblDisplay.Text &= (" " & objValue.PackageNumber)
Next
```

Programming Challenge

20.16 (***Enhanced Shipping Hub Application***) Enhance the **Shipping Hub** application created in Exercise 20.12 to allow the user to move a maximum of five packages from the warehouse to a truck for shipping (Fig. 20.30). If you have not completed Exercise 20.12, follow the steps in Exercise 20.12 before proceeding to the next step. If you have completed Exercise 20.12, copy the code you added to the `lstPackages ListBox DoubleClick` event handler to the same event handler in this application before beginning the exercise.

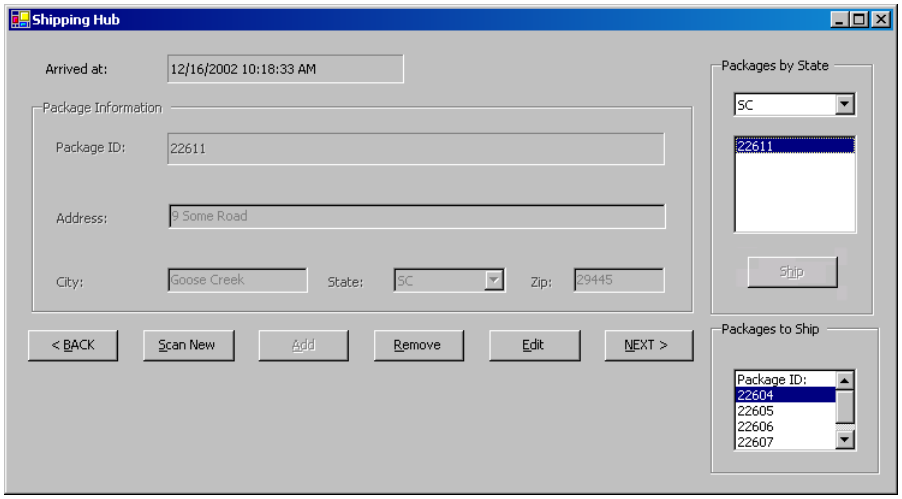


Figure 20.30 Enhanced Shipping Hub GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial20\Exercises\ShippingHubEnhanced` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ShippingHubEnhanced.sln` in the `ShippingHubEnhanced` directory to open the application.

c) ***Enabling the Ship `Button`.*** The **Ship** `Button` should not be enabled until a package is selected in the `lstPackage ListBox`. Double click the `lstPackage ListBox` from design view to define the event handler. Use the `Button`'s `Enabled` property to enable the `Button` if the `SelectedIndex` of the `ListBox` is not `-1`. This means that when the user selects a package from the `ListBox`, the user can send the package to the truck by clicking the **Ship** `Button`. Also, insert a line of code after the `For Each...Next` statement in the `SelectedIndexChanged` event handler to disable the **Ship** `Button` when a user chooses a different state.

d) ***Defining the Ship `Button`'s `Click` Event.*** Double click the **Ship** `Button` in **Design View** to define the `Click` event.

e) ***Incrementing the counter.*** Because you are only allowing five packages to be "shipped," declare an instance variable that will track how many packages have been placed onto the truck. Increment the variable each time the **Ship** `Button` is clicked.

f) ***Creating temporary variables.*** Create two temporary `Package` references to store the correct package's information. Use `objTempPackage` as the reference to the element in the collection type of a `For Each...Next` statement, and the `objTruckPackage` as a reference to the package added to the truck.

g) ***Using the `If...Then...Else` statement.*** Use an `If...Then...Else` statement to allow packages to be placed onto the truck if the number of packages on the truck is less than five.

h) ***Using the `For Each...Next` loop.*** Use a `For Each...Next` loop to iterate through the values in `m_objList`. Each iteration should determine whether the current package is the one selected from the `ListBox`.

i) ***Adding the package to the truck.*** When the `For Each...Next` loop has located the correct package, add that package to the truck by adding the reference to `objTempPackage` to the truck's `ArrayList`, `m_objTruckList`. Then assign the value in `objTempPackage` (the package sent to the truck) to `objTruckPackage`.

j) ***Removing the package.*** When the `For Each...Next` loop completes, remove the package meant for the truck from `m_objList` and the `lstPackages ListBox`.

k) ***Displaying the package in the `ListBox`.*** Use a `For Each...Next` loop that iterates through each package in the `m_objTruckList ArrayList` and displays each package in the `lstTruck ListBox`.

l) ***Refreshing the GUI***. Call the `ClearControls` and `SetButtons` methods to clear the `TextBoxes` and enable the appropriate `Buttons`. Also, set the **Ship** `Button`'s `Enabled` property to `False`.

m) ***Coding the `Else` statement.*** Display a `MessageBox` that notifies the user if the number of packages on the truck is already five. Then disable the **Ship** `Button`.

n) ***Running the application.*** Select **Debug > Start** to run your application. Add several packages. In the **Packages by State** `GroupBox`, select several packages and add them to the **Packages to Ship** `ListBox`. Verify that you can add only 5 packages to this `ListBox`.

o) ***Closing the application.*** Close your running application by clicking its close box.

p) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 20.16 Solution
' ShippingHub.vb

Public Class FrmShippingHub
   Inherits System.Windows.Forms.Form

   Private m_objList As Collections.ArrayList ' list of packages
   Private m_objPackage As Package   ' current package
   Private m_intPosition As Integer  ' position of current package
   Private m_objRandom As Random     ' random number for package id
   Private m_intPackageID As Integer ' individual package number
   Private m_objTruckList As Collections.ArrayList  ' shipment list
   Private m_intCounter As Integer = 0 ' count packages on truck

   ' Windows Form Designer generated code

   ' Form Load event
   Private Sub FrmShippingHub_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      m_intPosition = 0 ' set initial position to zero
      m_objRandom = New Random ' create new Random object
      m_intPackageID = m_objRandom.Next(1, 100000) ' new package ID

      ' show first state in ComboBox (using the Items property)
      cboState.Text = Convert.ToString(cboState.Items.Item(0))
      m_objList = New Collections.ArrayList ' list of packages
      m_objTruckList = New Collections.ArrayList ' truck list
   End Sub ' FrmShippingHub_Load

   ' Scan New Button Click event
   Private Sub btnNew_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNew.Click

      m_intPackageID += 1 ' increment package ID
      m_objPackage = New Package(m_intPackageID) ' create package

      ClearControls() ' clear fields
      lblPackageNumber.Text = _
         m_objPackage.PackageNumber.ToString ' package number
      lblArrivalTime.Text = _
         m_objPackage.ArrivalTime.ToString ' display arrival time

      ' only allow user to add package
      fraAddress.Enabled = True ' disable GroupBox and its controls
```

```
      SetButtons(False) ' enable/disable Buttons
      btnAdd.Enabled = True  ' enable Add Button
      btnNew.Enabled = False ' disable Scan New Button
      txtAddress.Focus() ' transfer the focus to txtAddress TextBox
   End Sub ' btnNew_Click

   ' Add Button Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      SetPackage() ' set Package properties from TextBoxes
      m_objList.Add(m_objPackage) ' add package to ArrayList

      fraAddress.Enabled = False ' disable GroupBox and its controls
      SetButtons(True) ' enable appropriate Buttons

      ' package cannot be added until Scan New is clicked
      btnAdd.Enabled = False ' disable Add Button

      ' if package's state displayed, add ID to ListBox
      If cboState.Text = cboViewPackages.Text Then
         lstPackages.Items.Add(m_objPackage.PackageNumber)
      End If

      cboViewPackages.Text = m_objPackage.State ' list packages
      btnNew.Enabled = True ' enable Scan New Button
   End Sub ' btnAdd_Click

   ' Back Button Click event
   Private Sub btnBack_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnBack.Click

      ' move backward one package in the list
      If m_intPosition > 0 Then
         m_intPosition -= 1
      Else ' wrap to end of list
         m_intPosition = m_objList.Count - 1
      End If

      LoadPackage() ' load package data from item in list
   End Sub ' btnBack_Click

   ' Next Button Click event
   Private Sub btnNext_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNext.Click

      ' move forward one package in the list
      If m_intPosition < m_objList.Count - 1 Then
         m_intPosition += 1
      Else
         m_intPosition = 0 ' wrap to beginning of list
      End If

      LoadPackage() ' load package data from item in list
   End Sub ' btnNext_Click

   ' Remove Button click event
   Private Sub btnRemove_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnRemove.Click

```

```
      ' remove ID from ListBox if state displayed
      If cboState.Text = cboViewPackages.Text Then
         lstPackages.Items.Remove(m_objPackage.PackageNumber)
      End If

      m_objList.RemoveAt(m_intPosition) ' remove package from list

      ' load next package in list if there is one
      If m_objList.Count > 0 Then

         ' if not at first position, go to previous one
         If m_intPosition > 0 Then
            m_intPosition -= 1
         End If

         LoadPackage() ' load package data from item in list
      Else
         ClearControls() ' clear fields
      End If

      SetButtons(True) ' enable appropriate Buttons
   End Sub ' btnRemove_Click

   ' Edit/Update Button Click event
   Private Sub btnEditUpdate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnEditUpdate.Click

      ' when Button reads "Edit", allow user to
      ' edit package information only
      If btnEditUpdate.Text = "&Edit" Then
         fraAddress.Enabled = True
         SetButtons(False)
         btnEditUpdate.Enabled = True

         ' change Button text from "Edit" to "Update"
         btnEditUpdate.Text = "&Update"
      Else

         ' when Button reads "Update" remove the old package
         ' data and add new data from TextBoxes
         SetPackage()
         m_objList.RemoveAt(m_intPosition)
         m_objList.Insert(m_intPosition, m_objPackage)

         ' display state in ComboBox
         cboViewPackages.Text = m_objPackage.State

         ' when done, return to normal operating state
         fraAddress.Enabled = False  ' disable GroupBox
         SetButtons(True) ' enable appropriate Buttons

         ' change Button text from "Update" to "Edit"
         btnEditUpdate.Text = "&Edit"
      End If

   End Sub ' btnEditUpdate_Click

   ' set package properties
   Private Sub SetPackage()
      m_objPackage.Address = txtAddress.Text
```

```
      m_objPackage.City = txtCity.Text
      m_objPackage.State = _
        Convert.ToString(cboState.SelectedItem)
      m_objPackage.Zip = Convert.ToInt32(Val(txtZip.Text))
   End Sub ' SetPackage

   ' load package information into Form
   Private Sub LoadPackage()

      ' retrieve package from list
      m_objPackage = CType(m_objList.Item(m_intPosition), _
         Package)

      ' display package data
      txtAddress.Text = m_objPackage.Address
      txtCity.Text = m_objPackage.City
      cboState.Text = m_objPackage.State
      txtZip.Text = m_objPackage.Zip.ToString("00000")
      lblArrivalTime.Text = _
         m_objPackage.ArrivalTime.ToString
      lblPackageNumber.Text = _
         m_objPackage.PackageNumber.ToString
   End Sub ' LoadPackage

   ' clear all the input controls on the Form
   Private Sub ClearControls()
      txtAddress.Clear()
      txtCity.Clear()
      txtZip.Clear()
      cboState.SelectedText = ""
      lblArrivalTime.Text = ""
      lblPackageNumber.Text = ""
   End Sub ' ClearControls

   ' enable/disable Buttons
   Private Sub SetButtons(ByVal blnState As Boolean)
      btnRemove.Enabled = blnState
      btnEditUpdate.Enabled = blnState
      btnNext.Enabled = blnState
      btnBack.Enabled = blnState

      ' disable navigation if not multiple packages
      If m_objList.Count < 2 Then
         btnNext.Enabled = False
         btnBack.Enabled = False
      End If

      ' if no items, disable Remove and Edit/Update Buttons
      If m_objList.Count = 0 Then
         btnEditUpdate.Enabled = False
         btnRemove.Enabled = False
      End If

   End Sub ' SetButtons

   ' event raised when user selects a new state in ComboBox
   Private Sub cboViewPackages_SelectedIndexChanged( _
      ByVal sender As System.Object, ByVal e As System.EventArgs) _
      Handles cboViewPackages.SelectedIndexChanged

      Dim objViewPackage As Package ' control variable package
```

```
      Dim strState As String = _
         Convert.ToString(cboViewPacakges.SelectedItem)

      lstPackages.Items.Clear() ' clear ListBox

      ' list all packages for current state in ListBox
      For Each objViewPackage In m_objList

         ' determine if state package is being shipped to
         ' matches the state selected in the ComboBox
         If objViewPackage.State = strState Then

            ' add package number to the ListBox
            lstPackages.Items.Add(objViewPackage.PackageNumber)
         End If

      Next

      btnShip.Enabled = False ' disable Ship Button

   End Sub ' cboViewPackages_SelectedIndexChanged

   ' displaying package information for selected package

   Private Sub lstPackages_DoubleClick(ByVal sender As _
     Object, ByVal e As System.EventArgs) _
     Handles lstPackages.DoubleClick

      Dim objPackageInformation As Package ' temporary package
      Dim strPackage As String = "" ' String for package information

      ' check if the lstPackages ListBox is empty
      If lstPackages.SelectedIndex <> -1 Then

         For Each objPackageInformation In m_objList

            ' if the package currently in objPackageInformation
            ' matches the user's selected package
            If objPackageInformation.PackageNumber = _
               Convert.ToDouble(lstPackages.SelectedItem) Then
               strPackage &= "Package " & _
                  objPackageInformation.PackageNumber & _
                  ControlChars.CrLf & _
                  "Arrived at: " & _
                  objPackageInformation.ArrivalTime & _
                  ControlChars.CrLf & _
                  "Address: " & _
                  objPackageInformation.Address & _
                  ControlChars.CrLf & _
                  "City: " & _
                  objPackageInformation.City & _
                  ControlChars.CrLf & _
                  "State: " & objPackageInformation.State & _
                  ControlChars.CrLf & _
                  "Zip code: " & _
                  objPackageInformation.Zip.ToString("00000"))

            End If

         Next

```

```
      Else

         ' if the user select a blank item in the ListBox
         strPackage = "Please select a package"
      End If

      MessageBox.Show(strPackage, "Package Information", _
         MessageBoxButtons.OK, MessageBoxIcon.Information)

   End Sub ' lstPackages_DoubleClick

   ' allow packages to be shipped
   Private Sub btnShip_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnShip.Click

      Dim objTempPackage As Package ' temporary package
      Dim objTruckPackage As Package ' package to remove

      m_intCounter += 1 ' increment package count

      ' if there is less than 6 packages in m_intCounter
      If m_intCounter <= 5 Then
         ' for each package from the m_objList ArrayList
         For Each objTempPackage In m_objList

            ' move package to truck
            If objTempPackage.PackageNumber = _
               Convert.ToInt32(lstPackages.SelectedItem) Then
               m_objTruckList.Add(objTempPackage)
               objTruckPackage = objTempPackage
            End If

         Next

         ' remove the package from warehouse
         m_objList.Remove(objTruckPackage)

         ' remove selected package
         lstPackages.Items.Remove(lstPackages.SelectedItem)

         lstTruck.Items.Clear() ' clear ListBox
         lstTruck.Items.Add("Package ID:") ' add header

         ' list all packages in ListBox
         For Each objViewPackage In m_objTruckList

            ' add package to lstTruck ListBox
            lstTruck.Items.Add(objViewPackage.PackageNumber)
         Next

         btnShip.Enabled = False ' disable the Ship Button
         ClearControls() ' clear the TextBoxes
         SetButtons(True) ' enable appropriate Buttons

      Else
         MessageBox.Show("Truck can only hold 5 packages", _
            "Limit Exceeded", _
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

         btnShip.Enabled = True ' enable the Ship Button
```

```
      End If

   End Sub ' btnShip_Click

   ' disable Ship button when no package is selected
   Private Sub lstPackages_SelectedIndexChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles lstPackages.SelectedIndexChanged

      If lstPackages.SelectedIndex <> -1 Then
         btnShip.Enabled = True ' enable the Ship Button
      End If

   End Sub ' lstPackages_SelectedIndexChanged

End Class ' FrmShippingHub
```

TUTORIAL 21

"Cat and Mouse" Painter Application

Introducing the `Graphics` Object and Mouse Events

Solutions

Instructor's Manual Exercise Solutions Tutorial 21

MULTIPLE-CHOICE QUESTIONS

21.1 The *x*- and *y*-coordinates of the `MouseEventArgs` object are relative to _____.

a) the screen
b) the application
c) the `Form` or control that contains the control that raised the event
d) None of the above.

21.2 The _____ method of the `Graphics` class draws a solid ellipse.

a) `FillEllipse`
b) `Ellipse`
c) `SolidEllipse`
d) `FilledEllipse`

21.3 The _____ object passed to a mouse event handler contains information about the mouse event that was raised.

a) `EventHandler`
b) `MouseEventHandler`
c) `MouseEventArgs`
d) `EventArgs`

21.4 The _____ event is raised when a mouse button is pressed.

a) `MousePress`
b) `MouseClick`
c) `MouseDown`
d) `MouseButtonDown`

21.5 A _____ is used to fill a shape with color using a `Graphics` object.

a) painter
b) brush
c) paint bucket
d) marker

21.6 A(n) _____ event is raised every time the mouse interacts with a control.

a) control
b) mouse pointer
c) mouse
d) user

21.7 The _____ property of `MouseEventArgs` specifies which mouse button was pressed.

a) `Source`
b) `Button`
c) `WhichButton`
d) `ButtonPressed`

21.8 The _____ class contains methods for drawing text, lines, rectangles and other shapes.

a) `Pictures`
b) `Drawings`
c) `Graphics`
d) `Illustrations`

21.9 An ellipse with its _____ is a circle.

a) height twice the length of its width
b) width set to zero
c) height half the length of its width
d) height equal to its width

21.10 The _____ method creates a `Graphics` object.

a) `NewGraphics`
b) `CreateGraphics`
c) `PaintGraphics`
d) `InitializeGraphics`

Answers: 21.1) c. 21.2) a. 21.3) c. 21.4) c. 21.5) b. 21.6) c. 21.7) b. 21.8) c. 21.9) d. 21.10) b.

EXERCISES

21.11 ***(Line Length Application)*** The **Line Length** application should draw a straight black line on the `Form` and calculate the length of the line(Fig. 21.27). The line should begin at the coordinates where the mouse button is pressed and should stop at the point where the mouse button is released. The application should display the line's length (that is, the distance between the two endpoints) in the `Label` **Length =**. Use the following formula to calculate

the line's length, where (x_1, y_1) is the first endpoint (the coordinates where the mouse button is pressed) and (x_2, y_2) is the second endpoint (the coordinates where the mouse button is released). To calculate the distance (or length) between the two points, use the following equation:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

To draw a straight line, you need to use the **`DrawLine`** method on a `Graphics` object. When drawing lines, you should use a **`Pen`** object, which is an object used to specify characteristics of lines and curves. Use the following method call to draw a black line between the two points using a `Graphics` object reference `objGraphic`:

```
objGraphic.DrawLine(New Pen(Color.Black),x1 , y1 , x2 , y2 )
```

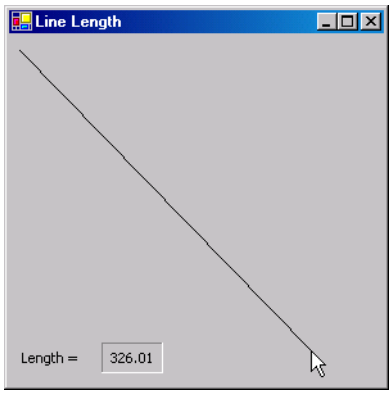


Figure 21.27 **Line Length** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial21\Exercises\LineLength` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `LineLength.sln` in the `LineLength` directory to open the application.

c) ***Declaring instance variables.*** Declare and initialize a reference to a `Graphics` object that you will use to draw a line. Then declare four `Integers` in which you will store the *x*- and *y*-coordinates of the two points.

d) ***Adding a `MouseDown` event handler.*** Create a `MouseDown` event handler. Add code to the `MouseDown` event handler to store the coordinates of the first endpoint of the line.

e) ***Creating the `Distance` method.*** Define a `Function` procedure named `Length` that returns the distance between two endpoints as a `Double`. The `Function` procedure should use the following statement to perform the line length calculation, where `intXDistance` is the difference between the *x*-coordinates of the two points and `intYDistance` is the difference between the *y*-coordinates of the two points:

```
Math.Sqrt((intXDistance ^ 2) + (intYDistance ^ 2))
```

f) ***Adding a `MouseUp` event handler.*** Create a `MouseUp` event handler. First store the coordinates of the line's second endpoint. Then call the `Length` method to obtain the distance between the two endpoints (the line's length). Finally, display the line on the `Form` and the line's length in the **Length =** `Label`, as in Fig. 21.27.

g) ***Running the application.*** Select **Debug > Start** to run your application. Draw several lines and view their lengths. Verify that the length values are accurate.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 21.11 Solution
' LineLength.vb

Public Class FrmLineLength
   Inherits System.Windows.Forms.Form

   ' create and initialize Graphics object
   Private m_objGraphic As Graphics = CreateGraphics()

   Private m_intX1 As Integer  ' first point's x-coordinate
   Private m_intY1 As Integer  ' first point's y-coordinate
   Private m_intX2 As Integer  ' second point's x-coordinate
   Private m_intY2 As Integer  ' second point's y-coordinate

   ' Windows Form Designer generated code

   ' handles FrmLineLength's MouseUp event
   Private Sub FrmLineLength_MouseDown(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles MyBase.MouseDown

      lblLength.Text = ""  ' clear Label

      ' get x- and y- coordinates of mouse click
      m_intX1 = e.X
      m_intY1 = e.Y
   End Sub ' FrmLineLength_MouseDown

   ' returns distance between two points
   Private Function Length() As Double

      ' horizontal distance
      Dim intXDistance As Integer = m_intX1 - m_intX2

      ' vertical distance
      Dim intYDistance As Integer = m_intY1 - m_intY2

      Return Math.Sqrt((intXDistance ^ 2) + (intYDistance ^ 2))
   End Function ' Length

   ' handles FrmLineLength's MouseUp event
   Private Sub FrmLineLength_MouseUp(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles MyBase.MouseUp

      ' final point
      m_intX2 = e.X
      m_intY2 = e.Y

      ' distance between two points
      Dim dblDistance As Double = Length()

      ' draw line connecting the two points
      m_objGraphic.DrawLine(New Pen(Color.Black), _
         m_intX1, m_intY1, m_intX2, m_intY2)

      ' display distance in Label
      lblLength.Text = String.Format("{0:F}", dblDistance)
   End Sub ' FrmLineLength_MouseUp
```

```

End Class ' FrmLineLength
```

21.12 ***(Circle Painter Application)*** The **Circle Painter** application should draw a blue circle with a randomly chosen size when the user presses a mouse button anywhere over the `Form` (Fig. 21.28). The application should randomly select a circle diameter in the range from `5` to `199`, inclusive. To draw a blue circle with a given diameter (`intDiameter`), use the following statement:

```
objGraphic.DrawEllipse(New Pen(Color.Blue), e.X, e.Y, _
   intDiameter, intDiameter)
```

The `DrawEllipse` method, when passed a `Pen` (instead of a brush) as an argument, draws the outline of an ellipse. Recall that an ellipse is a circle if the height and width arguments are the same (in this case, the randomly selected `intDiameter`). Use the *x*- and *y*-coordinates of the `MouseDown` event as the *x*- and *y*- coordinates of the circle's bounding box (that is, the second and third arguments to the `DrawEllipse` method). Notice that the first argument to the `DrawEllipse` method is a `Pen` object. See Exercise 21.11 for a description of `Pen`.

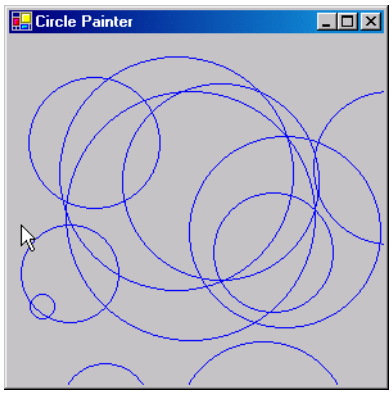


Figure 21.28 **Circle Painter** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial21\Exercises\CirclePainter` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CirclePainter.sln` in the `CirclePainter` directory to open the application.

c) ***Adding a `MouseDown` event handler.*** Create a `MouseDown` event handler. In the event handler, retrieve the *x*- and *y*-coordinates of the location the mouse pointer when a mouse button was pressed. Then generate a random number to use as the circle's diameter, using a `Random` object, and store it in a variable. Finally, call the `DrawEllipse` method on a reference to a `Graphics` object to draw a blue circle on the `Form` with the diameter generated by the `Random` object.

d) ***Running the application.*** Select **Debug > Start** to run your application. Draw several blue circles and make sure that they are of different sizes.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 21.12 Solution
' CirclePainter.vb

Public Class FrmCirclePainter
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code
```

```
    ' handles MouseDown event for FrmCirclePainter
    Private Sub FrmCirclePainter_MouseDown(ByVal sender As _
       Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
       Handles MyBase.MouseDown

       ' initialize Graphics object
       Dim objGraphic As Graphics = CreateGraphics()

       ' initialize Random object
       Dim objRandom As Random = New Random

       ' generate a random circle diameter
       Dim intDiameter As Integer = objRandom.Next(5, 200)

       ' draw a blue circle
       objGraphic.DrawEllipse(New Pen(Color.Blue), e.X, e.Y, _
          intDiameter, intDiameter)

    End Sub ' FrmCirclePainter_MouseDown

 End Class ' FrmCirclePainter
```

21.13 ***(Advanced Circle Painter Application)*** In this exercise, you will enhance the application you created in Exercise 21.12. The advanced **Circle Painter** application should draw blue circles with a randomly generated diameter when the user presses the left mouse button. When the user presses the right mouse button, the application should draw a red circle with a randomly generated diameter (Fig. 21.29).

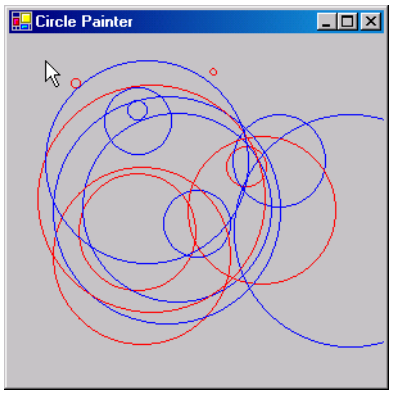


Figure 21.29 Advanced **Circle Painter** application's GUI.

a) ***Copying the template to your working directory.*** Make a copy of the `CirclePainter` directory from Exercise 21.12 in your `C:\SimplyVB` directory. Rename the copied directory `AdvancedCirclePainter`. If you have not completed Exercise 21.12, follow the steps in Exercise 21.12 to complete the application.

b) ***Opening the application's template file.*** Double click `CirclePainter.sln` file in the `AdvancedCirclePainter` directory to open the application.

c) ***Drawing the appropriate circle.*** Use the `Button` property of the `MouseEventArgs` reference, `e`, to determine which mouse button was pressed. Finally, call the `DrawEllipse` method on a reference to a `Graphics` object to draw a blue circle on the `Form` if the left mouse button was clicked, or a red circle if the right mouse button was clicked.

d) ***Running the application.*** Select **Debug > Start** to run your application. Draw several blue circles of different sizes using the left mouse button, then draw several red circles of different sizes using the right mouse button.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 21.13 Solution
' CirclePainter.vb

Public Class FrmCirclePainter
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles MouseDown event for FrmCirclePainter
   Private Sub FrmCirclePainter_MouseDown(ByVal sender As _
      Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles MyBase.MouseDown

      ' initialize Graphics object
      Dim objGraphic As Graphics = CreateGraphics()

      ' initialize Random object
      Dim objRandom As Random = New Random

      ' generate a random circle diameter
      Dim intDiameter As Integer = objRandom.Next(5, 200)

      ' left mouse button pressed
      If e.Button = MouseButtons.Left Then

         ' draw a blue circle if left mouse button pressed
         objGraphic.DrawEllipse(New Pen(Color.Blue), e.X, e.Y, _
            intDiameter, intDiameter)

      ' right mouse button pressed
      ElseIf e.Button = MouseButtons.Right Then

         ' draw a red circle if right mouse button pressed
         objGraphic.DrawEllipse(New Pen(Color.Red), e.X, e.Y, _
            intDiameter, intDiameter)

      End If

   End Sub ' FrmCirclePainter_MouseDown

End Class ' FrmCirclePainter
```

What does this code do?

21.14 Consider the code in Fig. 21.26. Suppose we change the `MouseMove` event handler to the code below. What happens when the user moves the mouse? Assume that a `Label lblDisplay` has been placed on the `Form`.

```
Private Sub FrmPainter_MouseMove(ByVal sender As Object, _
   ByVal e As System.Windows.Forms.MouseEventArgs)
   Handles MyBase.MouseMove

   lblDisplay.Text = "I'm at " & e.X & ", " & e.Y & "."
End Sub ' FrmPainter_MouseMove
```

Answer: The `Label` continuously displays the mouse pointer's current position on the `Form`.

What's wrong with this code?

21.15 The following code should draw a `BlueViolet` circle of diameter `4` that corresponds to the movement of the mouse. Find the error(s) in the following code:

```
Private Sub FrmPainter_MouseMove(ByVal sender As Object, _
   ByVal e As System.Windows.Forms.MouseEventArgs) _
   Handles MyBase.MouseMove

   If m_blnshouldPaint = True Then
      Dim objGraphic As Graphics = Graphics()

      objGraphic.FillEllipse = ( _
         New SolidBrush(Color.BlueViolet), e.Y, e.X, 5, 4)
   End If
End Sub ' FrmPainter_MouseMove
```

Answer: The position arguments in the `FillEllipse` method have been transposed. Use method `CreateGraphics` to initialize a `Graphics` object. A circle's height and width must be equal, so the fourth argument passed to method `FillEllipse` should be `4`. There should be no assignment operator between the word `FillEllipse` and the parenthesis. The corrected code is as follows:

```
Private Sub FrmPainter_MouseMove(ByVal sender As Object, _
   ByVal e As System.Windows.Forms.MouseEventArgs)  _
   Handles MyBase.MouseMove

   If m_blnshouldPaint = True Then
      Dim objGraphic As Graphics = CreateGraphics()

      objGraphic.FillEllipse( _
         New SolidBrush(Color.BlueViolet), e.X, e.Y, 4, 4)
   End If
End Sub ' FrmPainter_MouseMove
```

Programming Challenge

21.16 ***(Advanced Painter Application)*** Extend the `Painter` application to enable a user to change the size and color of the circles drawn.

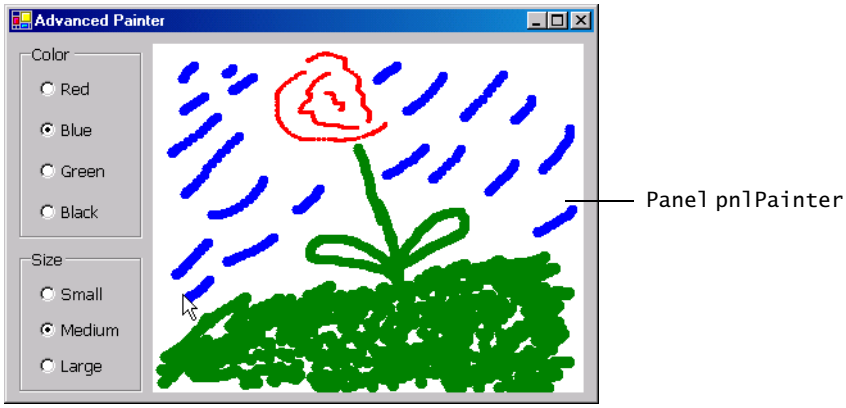


Figure 21.30 Advanced Painter application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial21\Exercises\AdvancedPainter` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AdvancedPainter.sln` in the `AdvancedPainter` directory to open the application (Fig. 21.30).

c) ***Understanding the provided instance variables.*** The template already provides you with four instance variables. Variable `m_objBrushColor` is a `Color` value that specifies the color of the brush used in the **Advanced Painter** application. The `m_blnShouldPaint` and `m_blnShouldErase` variable perform the same functions as in this tutorial's **Painter** application. The `m_intDiameter` variable stores the diameter of the circle to be drawn.

d) ***Declaring an enumeration to store the circle diameter sizes.*** Declare an enumeration `Sizes` to store the possible values of `m_intDiameter`. Set constant `SMALL` to `4`, `MEDIUM` to `8` and `LARGE` to `10`.

e) ***Adding event handlers for the Color `RadioButtons`.*** The **Color** `RadioButton`'s event handlers should set `m_objBrushColor` to their specified colors (`Color.Red`, `Color.Blue`, `Color.Green` or `Color.Black`).

f) ***Adding event handlers for the Size `RadioButtons`.*** The **Size** `RadioButton`'s event handlers should set `m_intDiameter` to `Sizes.SMALL` (for the **Small** `RadioButton`), `Sizes.MEDIUM` (for the **Medium** `RadioButton`) or `Sizes.LARGE` (for the **Large** `RadioButton`).

g) ***Adding a mouse event handler to a `Panel`.*** To associate mouse events with the `Panel`, select `pnlPainter` from the Class Name `ComboBox`. Then select the appropriate mouse event from the Method Name `ComboBox`.

h) ***Coding the `MouseDown` and `MouseUp` event handlers.*** The `MouseUp` and `MouseDown` event handlers behave exactly as they do in the **Painter** application.

i) ***Coding the `MouseMove` event handler.*** The `MouseMove` event handler behaves as the one in **Painter** application does. The color of the brush that draws the circle when `m_blnShouldPaint` is `True` is specified by `m_objBrushColor`. The eraser color is specified by the `Panel`'s `BackColor` property.

j) ***Running the application.*** Select **Debug > Start** to run your application. Start drawing on the `Panel` using different brush sizes and colors. Use the right mouse button to erase part of your drawing.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 21.16 Solution
' AdvancedPainter.vb

Public Class FrmAdvancedPainter
   Inherits System.Windows.Forms.Form

   ' create Color value and initialize to Black
   Private m_objBrushColor As Color = Color.Black

   ' specify whether application should paint
   Private m_blnShouldPaint As Boolean = False

   ' specify whether application should erase
   Private m_blnShouldErase As Boolean = False

   ' diameter of MouseDown circle (initially set to small)
   Private m_intDiameter As Integer = 4

   ' size constants for diameter of MouseDown circle
   Private Enum Sizes
      SMALL = 4
      MEDIUM = 8
      LARGE = 10
   End Enum
```

```

    ' Windows Form Designer generated code

    ' handles radRed's CheckChanged event
    Private Sub radRed_CheckedChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles radRed.CheckedChanged

       ' set brush color to red
       If radRed.Checked = True Then
          m_objBrushColor = Color.Red
       End If

    End Sub ' radRed_CheckedChanged

    ' handles radBlue's CheckChanged event
    Private Sub radBlue_CheckedChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles radBlue.CheckedChanged

       ' set brush color to blue
       If radBlue.Checked = True Then
          m_objBrushColor = Color.Blue
       End If

    End Sub ' radBlue_CheckedChanged

    ' handles radGreen's CheckChanged event
    Private Sub radGreen_CheckedChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles radGreen.CheckedChanged

       ' set brush color to green
       If radGreen.Checked = True Then
          m_objBrushColor = Color.Green
       End If

    End Sub ' radGreen_CheckedChanged

    ' handles radBlack's CheckChanged event
    Private Sub radBlack_CheckedChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles radBlack.CheckedChanged

       ' set brush color to black
       If radBlack.Checked = True Then
          m_objBrushColor = Color.Black
       End If

    End Sub ' radBlack_CheckedChanged

    ' handles radSmall's CheckChanged event
    Private Sub radSmall_CheckedChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles radSmall.CheckedChanged

       ' draw small circles
       If radSmall.Checked = True Then
          m_intDiameter = Sizes.SMALL
       End If

```

```
   End Sub ' radSmall_CheckedChanged

   ' handles radSMedium's CheckChanged event
   Private Sub radMedium_CheckedChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles radMedium.CheckedChanged

      ' draw medium circles
      If radMedium.Checked = True Then
         m_intDiameter = Sizes.MEDIUM
      End If

   End Sub ' radMedium_CheckedChanged

   ' handles radLarge's CheckChanged event
   Private Sub radLarge_CheckedChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles radLarge.CheckedChanged

      ' draw large circles
      If radLarge.Checked = True Then
         m_intDiameter = Sizes.LARGE
      End If

   End Sub ' radLarge_CheckedChanged

   ' draw when mouse button pressed down
   Private Sub pnlPainter_MouseDown(ByVal sender As _
      Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles pnlPainter.MouseDown

      ' draw if left mouse button held down
      If e.Button = MouseButtons.Left Then
         m_blnShouldPaint = True

      ' erase if right mouse button held down
      ElseIf e.Button = MouseButtons.Right Then
         m_blnShouldErase = True
      End If

   End Sub ' pnlPainter_MouseDown

   ' stop drawing after mouse released
   Private Sub pnlPainter_MouseUp(ByVal sender As _
      Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles pnlPainter.MouseUp

      m_blnShouldPaint = False  ' do not draw
      m_blnShouldErase = False  ' do not erase
   End Sub ' pnlPainter_MouseUp

   ' draw when mouse moves if mouse down
   Private Sub pnlPainter_MouseMove(ByVal sender As _
      Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles pnlPainter.MouseMove

      ' create Graphics object for the Panel
      Dim objGraphic As Graphics = pnlPainter.CreateGraphics()

      ' draw circles with specified brush color and size
      If m_blnShouldPaint = True Then
```

```

         objGraphic.FillEllipse(New SolidBrush(m_objBrushColor), _
            e.X, e.Y, m_intDiameter, m_intDiameter)

      ' draw circles with Panel's background color and specified size
      ElseIf m_blnShouldErase = True Then

         objGraphic.FillEllipse( _
            New SolidBrush(pnlPainter.BackColor), _
            e.X, e.Y, m_intDiameter, m_intDiameter)

      End If

   End Sub ' pnlPainter_MouseMove

End Class ' FrmAdvancedPainter
```

TUTORIAL 22

Typing Application

Introducing Keyboard Events, Menus and Dialogs

Solutions

Instructor's Manual Exercise Solutions Tutorial 23

MULTIPLE-CHOICE QUESTIONS

22.1 When creating a menu, typing a _____ in front of a menu item name will create an access shortcut for that item.

a) &
b) !
c) $
d) #

22.2 *Alt*, *Shift* and *Control* are _____ keys.

a) modifier
b) ASCII
c) function
d) special

22.3 `KeyChar` is a property of _____.

a) `KeyEventArgs`
b) `Key`
c) `KeyArgs`
d) `KeyPressEventArgs`

22.4 Typing a hyphen (`-`) as a menu item's `Text` property will create a(n) _____.

a) separator bar
b) access shortcut
c) new submenu
d) keyboard shortcut

22.5 A _____ provides a group of related commands for Windows applications.

a) separator bar
b) hot key
c) menu
d) margin indicator bar

22.6 The _____ enumeration specifies key codes and modifiers.

a) `Keyboard`
b) `Key`
c) `KeyboardTypes`
d) `Keys`

22.7 The _____ event is raised when a key is pressed by the user.

a) `KeyPress`
b) `KeyHeld`
c) `KeyDown`
d) Both a and c.

22.8 Which of the following is not a keyboard event?

a) `KeyPress`
b) `KeyDown`
c) `KeyUp`
d) `KeyClicked`

22.9 Which of the following is not a structure?

a) `Char`
b) `Color`
c) `String`
d) `Date`

22.10 The _____ type allows you to determine which `Button` the user clicked to exit a dialog.

a) `DialogButtons`
b) `DialogResult`
c) `Buttons`
d) `ButtonResult`

Answers: 22.1) a. 22.2) a. 22.3) d. 22.4) a. 22.5) c. 22.6) d. 22.7) d. 22.8) d. 22.9) c. 22.10) b.

EXERCISES

22.11 (***Inventory Application with Keyboard Events***) Enhance the **Inventory** application that you developed in Tutorial 4 to prevent the user from entering input that is not a number. Use keyboard events to allow the user to press the number keys, the left and right arrows and the *Backspace* keys. If a key other than these is pressed, display a `MessageBox` instructing the user to enter a number (Fig. 22.30).

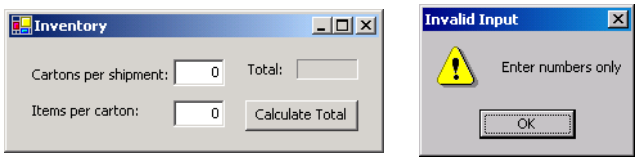


Figure 22.30 Enhanced **Inventory** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial22\Exercises\KeyEventInventory` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `KeyEventInventory.sln` in the `KeyEventInventory` directory to open the application.

c) ***Adding the `KeyDown` event handler for the first `TextBox`.*** Add an empty `KeyDown` event handler for the **Cartons per shipment:** `TextBox`.

d) ***Adding a `Select Case` statement.*** Add a `Select Case` statement to the `KeyDown` event handler that determines whether a number key, a left or right arrow or the *Backspace* key was pressed.

e) ***Adding the `Case Else` statement.*** Add a `Case Else` statement that will determine whether a key other than a valid one for this application was pressed. If an invalid key was pressed, display a `MessageBox` that instructs the user to enter a number.

f) ***Adding the `KeyDown` event handler for the second `TextBox`.*** Repeat *Steps c–e*, only this time create a `KeyDown` event handler for the **Items per carton:** `TextBox`. This event handler should perform the same functionality as the one for the **Cartons per shipment:** `TextBox`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Try entering letters or pressing the up and down arrow keys in the `TextBoxes`. A `MessageBox` should be displayed. Enter valid input and click the **Calculate Total** `Button`. Verify that the correct output is displayed.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 22.11 Solution
' Inventory.vb

Public Class FrmInventory
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      ' multiply values input and display result in Label
      lblTotalResult.Text = _
         (Val(txtCartons.Text) _
         * Val(txtItems.Text)).ToString

   End Sub ' btnCalculate_Click

   Private Sub txtCartons_KeyDown(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.KeyEventArgs) _
      Handles txtCartons.KeyDown

      Dim result As DialogResult ' store result of MessageBox

```

```
      Select Case e.KeyData
         Case Keys.D0 To Keys.D9 ' numbers

         Case Keys.Back ' backspace

         Case Keys.Enter ' enter

         Case Keys.Left, Keys.Right ' arrows

         Case Else ' all other keys

            ' show MessageBox
            result = MessageBox.Show("Enter numbers only", _
               "Invalid Input", MessageBoxButtons.OK, _
               MessageBoxIcon.Exclamation)

            ' clear TextBox if invalid input entered
            If result = DialogResult.OK Then
               txtCartons.Clear()
            End If

      End Select

   End Sub ' txtCartons_KeyDown

   Private Sub txtItems_KeyDown(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.KeyEventArgs) _
      Handles txtItems.KeyDown

      Dim result As DialogResult ' store result of MessageBox

      Select Case e.KeyData
         Case Keys.D0 To Keys.D9 ' numbers

         Case Keys.Back ' backspace

         Case Keys.Enter ' enter

         Case Keys.Left, Keys.Right ' arrows

         Case Else ' all other keys

            ' show MessageBox
            result = MessageBox.Show("Enter numbers only", _
               "Invalid Input", MessageBoxButtons.OK, _
               MessageBoxIcon.Exclamation)

            ' clear TextBox if invalid input entered
            If result = DialogResult.OK Then
               txtItems.Clear()
            End If

      End Select

   End Sub ' txtItems_KeyDown

End Class ' FrmInventory
```

22.12 (*Bouncing Ball Game*) Write an application that allows the user to play a game, the goal of which is to prevent a bouncing ball from falling off the bottom of the `Form`. When the

user presses the *S* key, a blue ball will bounce off the top, left and right sides (the "walls") of the `Form`. There should be a horizontal bar on the bottom of the `Form`, which serves as a paddle to prevent the ball from hitting the bottom of the `Form`. (The ball can bounce off the paddle, but not the bottom of the `Form`.) The user can move the paddle using the left and right arrow keys. If the ball hits the paddle, the ball should bounce up, and the game should continue. If the ball hits the bottom of the `Form`, the game should end. The paddle's width should decrease every 20 seconds to make the game more challenging. The GUI is provided for you (Fig. 22.31).

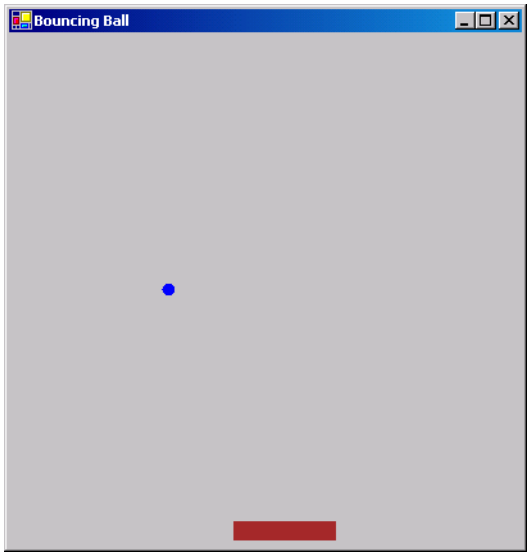


Figure 22.31 Bouncing Ball application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial22\Exercises\BouncingBall` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `BouncingBall.sln` in the `BouncingBall` directory to open the application.

c) ***Creating the `KeyDown` event handler.*** Insert a `KeyDown` event handler for the `Form`.

d) ***Writing code to start the game.*** Write an `If...Then` statement in the `KeyDown` event handler that tests whether the user presses the *S* key. You can use the `KeyDown` event handler for the *S* key in this case because you do not care whether the user presses an uppercase *S* or a lowercase *S*. If the user presses the *S* key, start the two `Timers` that are provided in the template.

e) ***Inserting code to move the paddle left.*** Write an `If...Then` statement that tests if the user pressed the left-arrow key and if the paddle's horizontal position is greater than zero. If the paddle's horizontal position equals zero, the left edge of the paddle is touching the left wall and the paddle should not be allowed to move farther to the left. If both the conditions in the `If…Then` are true, decrease the paddle's *x*-position by `10`.

f) ***Inserting code to move the paddle right.*** Write an `If...Then` statement that tests if the user pressed the right-arrow key and whether the paddle's *x*-coordinate is less than the width of the `Form` minus the width of the paddle. If the paddle's x-coordinate equals the `Form`'s width minus the width of the paddle, the paddle's right edge is touching the right wall and the paddle should not be allowed to move farther to the right. If both the conditions in the `If…Then` statement are true, increase the paddle's *x*-coordinate by `10`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Press the *S* key to begin the game and use the paddle to keep the bouncing ball from dropping off the `Form`. Continue doing this until 20 seconds have passed, and verify that the paddle is decreased in size at that time.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 22.12 Solution
' BouncingBall.vb

Public Class FrmBouncingBall
   Inherits System.Windows.Forms.Form

   Private intX As Integer ' ball's x-coordinate
   Private intY As Integer ' ball's y-coordinate
   Private intRectangleX As Integer ' paddle's x-coordinate
   Private intRectangleWidth As Integer ' paddle's width

   Private intDeltaX As Integer ' ball's x rate of change
   Private intDeltaY As Integer ' ball's y rate of change

   Private blnXLeft As Boolean ' tests if ball can move left
   Private blnYUp As Boolean ' tests if ball can move up

   Private Const intMAX_X As Integer = 400 ' x boundary
   Private Const intMAX_Y As Integer = 400 ' y boundary

   ' object to generate random numbers
   Private objRandom As Random = New Random

   ' Windows Form Designer generated code

   Private Sub FrmBouncingBall_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      intX = objRandom.Next(100, 301) ' ball's initial x
      intY = objRandom.Next(100, 301) ' ball's initial y
      intRectangleX = 175 ' rectangle's intial x position
      intRectangleWidth = 80 ' rectangle's intial width

      blnXLeft = False ' ball can move left
      blnYUp = False ' ball can move up
      intDeltaX = 2 ' move ball 2 positions right
      intDeltaY = 2 ' move ball 2 positions down
   End Sub ' FrmBouncingBall_Load

   Protected Overrides Sub OnPaint( _
      ByVal e As System.Windows.Forms.PaintEventArgs)

      ' create graphics object
      Dim objGraphic As Graphics = CreateGraphics()

      ' create new brush
      Dim objBrush As SolidBrush = New SolidBrush(Color.Blue)

      ' draw ball
      objGraphic.FillEllipse(objBrush, intX, intY, 10, 10)

      ' set color for, and draw paddle
      objBrush.Color = Color.Brown
      objGraphic.FillRectangle( _
         objBrush, intRectangleX, 380, intRectangleWidth, 15)
   End Sub ' OnPaint
```

```

    Private Sub tmrMoveBall_Tick(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles tmrMoveBall.Tick

       ' determine new x position
       If blnXLeft = True Then
          intX += intDeltaX
       Else
          intX -= intDeltaX
       End If

       ' determine new y position
       If blnYUp Then
          intY += intDeltaY
       Else
          intY -= intDeltaY
       End If

       If intY <= 0 Then
          blnYUp = True
          intDeltaY = objRandom.Next(2, 6)
       ElseIf intY >= 370 AndAlso intX >= intRectangleX _
          AndAlso intX <= (intRectangleX + intRectangleWidth) Then
          blnYUp = False
          intDeltaY = objRandom.Next(2, 6)
       ElseIf intY >= 410 Then ' end game if ball hits floor
          tmrMoveBall.Enabled = False
          tmrShrinkSlider.Enabled = False
          MessageBox.Show("Game Over")
       End If

       If intX <= 0 Then
          blnXLeft = True
          intDeltaX = objRandom.Next(2, 6)
       ElseIf intX >= intMAX_X - 10 Then
          blnXLeft = False
          intDeltaX = objRandom.Next(2, 6)
       End If

       Invalidate() ' Refresh Form

    End Sub ' tmrMoveBall_Tick

    ' shrinks the paddle every 20 seconds
    Private Sub tmrShrinkSlider_Tick(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles tmrShrinkSlider.Tick

       ' shrink paddle if paddle greater than twice ball's width
       If intRectangleWidth >= 20 Then
          intRectangleWidth = Convert.ToInt32( _
             intRectangleWidth / 2)
       End If

    End Sub ' tmrShrinkSlider_Tick

    ' handles KeyDown event
    Private Sub FrmBouncingBall_KeyDown(ByVal sender As Object, _
       ByVal e As System.Windows.Forms.KeyEventArgs) Handles _
       MyBase.KeyDown

```

```
      If e.KeyCode = Keys.S Then ' start game if user presses S key
         tmrMoveBall.Enabled = True ' start tmrMoveBall
         tmrShrinkSlider.Enabled = True ' start tmrShrinkSlider
      End If

      If e.KeyCode = Keys.Left AndAlso intRectangleX >= 0 Then
         intRectangleX -= 10 ' move paddle to the left
      End If

      If e.KeyCode = Keys.Right AndAlso _
         (intRectangleX <= intMAX_X - intRectangleWidth) Then

         intRectangleX += 10 ' move paddle to the right
      End If

   End Sub ' FrmBouncingBall_KeyDown

End Class ' FrmBouncingBall
```

22.13 (*Modified Painter Application*) Modify the **Painter** application that you developed in Tutorial 21 to include menus that allow the user to select the size and color of the painted ellipses and the color of the `Form` (Fig. 22.32). (The menus replace the `RadioButtons`.) Also, add a multiline `TextBox` that allows the user to type text to accompany the painting. The user should be able to use menus to select the font style and color of the text and the background color of the `TextBox`.

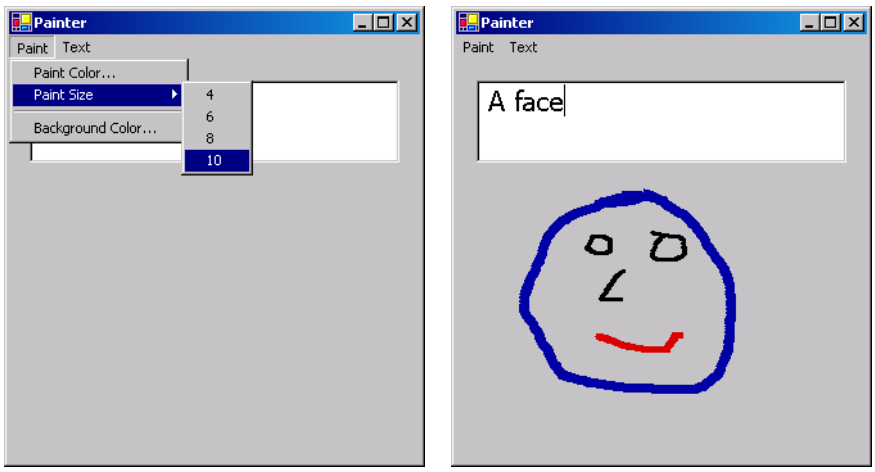


Figure 22.32 Enhanced **Painter** GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial22\Exercises\ModifiedPainter` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `ModifiedPainter.sln` in the `ModifiedPainter` directory to open the application.

c) ***Creating the menus.*** Create two menus. The first one should be titled **Paint** and should contain a **Paint Color...** menu item, a **Paint Size** submenu that contains menu items **4**, **6**, **8** and **10**, a separator bar and a **Background Color...** menu item. The second menu should be titled **Text** and have **Text Color...** and **Font...** menu items, a separator bar and a **TextBox Color...** menu item.

d) ***Changing the paint color.*** Add an event handler for the **Paint Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the value stored in `m_paintColor`.

e) ***Changing the paint size.*** Add an event handler for each of the **Size** submenu's menu items. Each event handler should change the value stored in `m_intDiamter` to the

value displayed on the menu (that is, clicking the **4** menu item will change the value of `m_intDiameter` to 4).

f) ***Changing the background color***. Add an event handler for the **Background Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the value stored in `m_backgroundColor` and also change the `BackColor` property of the `Form`. To change the background color of the `Form`, assign the value specifying the background color to `BackColor`. For instance, the statement `BackColor = Color.White` changes the background color of the `Form` to white.

g) ***Changing the text color***. Add an event handler for the **Text Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the color of the text displayed in the `TextBox`.

h) ***Changing the text style***. Add an event handler for the **Font...** menu item. This event handler should display a **Font** dialog that allows the user to change the style of the text displayed in the `TextBox`.

i) ***Changing the `TextBox`'s background color***. Add an event handler for the **TextBox Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the background color of the `TextBox`.

j) ***Running the application.*** Select **Debug > Start** to run your application. Use the menus to draw shapes of various colors and brush sizes. Enter text to describe your drawing. Use the other menu options to change the color of the `Form`, the `TextBox` and the text in the `TextBox`.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 22.13 Solution
' Painter.vb

Public Class FrmPainter
   Inherits System.Windows.Forms.Form

   ' specify whether moving the mouse should erase
   Private m_blnShouldErase As Boolean

   ' specify whether moving the mouse should draw
   Private m_blnShouldPaint As Boolean

   ' set diameter of MouseDown circle
   Private m_intDiameter As Integer
   Private m_paintColor As Color ' paint color

   Private m_backgroundColor As Color ' background color

   ' create Graphics object
   Private m_objGraphic As Graphics

   ' Windows Form Designer generated code

   Private Sub FrmPainter_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      m_blnShouldErase = False ' should not be painting
      m_blnShouldPaint = False ' should not be erasing
      m_intDiameter = 8 ' set paint size
      m_paintColor = Color.BlueViolet ' set paint color
      m_backgroundColor = FrmPainter.DefaultBackColor
      m_objGraphic = CreateGraphics() ' initialize graphics object
   End Sub ' FrmPainter_Load

```

```
    ' handles FrmPainter's MouseDown event
    Private Sub FrmPainter_MouseDown(ByVal sender As Object, _
       ByVal e As System.Windows.Forms.MouseEventArgs) _
       Handles MyBase.MouseDown

       ' draw on Form if the left button is held down
       If e.Button = MouseButtons.Left Then
          m_blnShouldPaint = True

       ' erase blue-violet circles if right button is held down
       ElseIf e.Button = MouseButtons.Right Then
          m_blnShouldErase = True
       End If

    End Sub ' FrmPainter_MouseDown

    ' handles FrmPainter's MouseUp event
    Private Sub FrmPainter_MouseUp(ByVal sender As Object, _
       ByVal e As System.Windows.Forms.MouseEventArgs) _
       Handles MyBase.MouseUp

       m_blnShouldPaint = False  ' do not draw on the Form
       m_blnShouldErase = False  ' do not erase

    End Sub ' FrmPainter_MouseUp

    ' handles FrmPainter's MouseMove event
    Private Sub FrmPainter_MouseMove(ByVal sender As Object, _
       ByVal e As System.Windows.Forms.MouseEventArgs) _
       Handles MyBase.MouseMove

       ' draw circle if left mouse button is pressed
       If m_blnShouldPaint Then
          m_objGraphic.FillEllipse(New SolidBrush(m_paintColor), _
             e.X, e.Y, m_intDiameter, m_intDiameter)

       ' mouse pointer "erases" if right mouse button is pressed
       ElseIf m_blnShouldErase Then
          m_objGraphic.FillEllipse( _
             New SolidBrush(m_backgroundColor), _
             e.X, e.Y, m_intDiameter, m_intDiameter)
       End If

    End Sub ' FrmPainter_MouseMove

    Private Sub mnuitmColor_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles mnuitmColor.Click

       Dim dlgColorDialog As New ColorDialog   ' Color Dialog
       Dim result As DialogResult ' stores Button clicked

       dlgColorDialog.FullOpen = True ' show all colors
       result = dlgColorDialog.ShowDialog

       ' do nothing if user clicked dialog's Cancel Button
       If result = DialogResult.Cancel Then
          Return
       End If

       ' assign new color to Paint object
       m_paintColor = dlgColorDialog.Color
```

```
   End Sub ' mnuitmColor_Click

   Private Sub mnuitmFour_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles mnuitmFour.Click

      m_intDiameter = 4 ' set paint size to four
   End Sub ' mnuitmFour_Click

   Private Sub mnuitmSix_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles mnuitmSix.Click

      m_intDiameter = 6 ' set paint size to six
   End Sub ' mnuitmSix_Click

   Private Sub mnuitmEight_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles mnuitmEight.Click

      m_intDiameter = 8 ' set paint size to eight
   End Sub ' mnuitmEight_Click

   Private Sub mnuitmTen_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles mnuitmTen.Click

      m_intDiameter = 10 ' set paint size to ten
   End Sub ' mnuitmTen_Click

   Private Sub mnuitmBackground_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles mnuitmBackground.Click

      Dim dlgColorDialog As New ColorDialog   ' Color Dialog
      Dim result As DialogResult ' stores Button clicked

      dlgColorDialog.FullOpen = True ' show all colors
      result = dlgColorDialog.ShowDialog()

      ' do nothing if user clicked dialog's Cancel Button
      If result = DialogResult.Cancel Then
         Return
      End If

      m_backgroundColor = dlgColorDialog.Color ' set "erase" color
      BackColor = dlgColorDialog.Color ' set Form's color
   End Sub ' mnuitmBackground_Click

   Private Sub mnuitmTextColor_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles mnuitmTextColor.Click

      Dim dlgColorDialog As New ColorDialog   ' Color Dialog
      Dim result As DialogResult ' stores Button clicked

      dlgColorDialog.FullOpen = True ' show all colors
      result = dlgColorDialog.ShowDialog()

      ' do nothing if user clicked dialog's Cancel Button
      If result = DialogResult.Cancel Then
         Return
      End If

      ' assign new color to text
      txtOutput.ForeColor = dlgColorDialog.Color
```

```
   End Sub ' mnuitmTextColor_Click

   Private Sub mnuitmFont_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles mnuitmFont.Click

      Dim dlgFontDialog As New FontDialog   ' Font Dialog
      Dim result As DialogResult ' stores Button clicked

      ' show dialog and get result
      result = dlgFontDialog.ShowDialog()

      ' do nothing if user clicked dialog's Cancel Button
      If result = DialogResult.Cancel Then
         Return
      End If

      ' assign new font value to TextBox
      txtOutput.Font = dlgFontDialog.Font
   End Sub ' mnuitmFont_Click

   Private Sub mnuitmTextBoxColor_Click(ByVal sender As _
      System.Object,  ByVal e As System.EventArgs) _
      Handles mnuitmTextBoxColor.Click

      Dim dlgColorDialog As New ColorDialog   ' Color Dialog
      Dim result As DialogResult ' stores Button clicked

      dlgColorDialog.FullOpen = True ' show all colors
      result = dlgColorDialog.ShowDialog()

      ' do nothing if user clicked dialog's Cancel Button
      If result = DialogResult.Cancel Then
         Return
      End If

      ' assign background color of TextBox
      txtOutput.BackColor = dlgColorDialog.Color
   End Sub ' mnuitmTextBoxColor_Click

End Class ' FrmPainter
```

What does this code do?

22.14 What is the result of the following code?

```
Private Sub mnuitmColor_Click(ByVal sender As _
   System.Object, ByVal e As System.EventArgs) _
   Handles mnuitmColor.Click

   Dim dlgColorDialog As ColorDialog = New ColorDialog
   Dim result As DialogResult

   dlgColorDialog.FullOpen = True

   result = dlgColorDialog.ShowDialog()

   If result = DialogResult.Cancel Then
     Return
   End If

```

```
   BackColor = dlgColorDialog.Color
End Sub ' mnuitmColor_Click
```

Answer: When this event handler executes, the **Color** dialog is displayed. If the user chooses a color, that color is assigned to the `BackColor` property of `lblDisplay`. If the dialog's **Cancel** `Button` is clicked, the dialog closes and this event handler terminates.

What's wrong with this code?

22.15 This code should allow a user to pick a font from a **Font** dialog and set the text in `txtDisplay` to that font. Find the error(s) in the following code, assuming that a `TextBox` named `txtDisplay` exists on a `Form`.

```
Private Sub Fonts()
   Dim dlgFontDialog As FontDialog

   dlgFontDialog = New FontDialog
   dlgFontDialog.ShowDialog()
   txtDisplay.Font = dlgFontDialog.Font
End Sub ' Fonts
```

Answer: The code should check whether the user clicked the **Cancel** `Button` in the dialog. If the user clicked **Cancel**, the method should be terminated. If the user did not click **Cancel**, the text should be set to the selected style.

```
Private Sub Fonts()
   Dim result As DialogResult
   Dim dlgFontDialog As FontDialog

   dlgFontDialog = New FontDialog()
   result = dlgFontDialog.ShowDialog()

   If result = DialogResult.Cancel Then
      Return
   Else
      txtDisplay.Font = dlgFontDialog.Font
   End If

End Sub ' Fonts
```

Programming Challenge

22.16 (***Dvorak Keyboard Application***) Create an application that simulates the letters on the Dvorak keyboard. A Dvorak keyboard allows faster typing by placing the most commonly used keys in the most accessible locations. Use keyboard events to create an application similar to the **Typing Application**, except that it simulates the Dvorak keyboard instead of the standard keyboard. The correct Dvorak key should be highlighted on the virtual keyboard and the correct character should be displayed in the `TextBox`. The keys and characters map as follows:

- On the top row, the *P* key of the Dvorak keyboard maps to the *R* key on a standard keyboard, and the *L* key of the Dvorak keyboard maps to the *P* key on a standard keyboard.
- On the middle row, the *A* key remains in the same position and the *S* key on the Dvorak keyboard maps to the semicolon key on the standard keyboard.
- On the bottom row, the *Q* key on the Dvorak keyboard maps to the *X* key on the standard keyboard and the *Z* key maps to the question mark key.
- All of the other keys on the Dvorak keyboard map to the location shown in Fig. 22.33.

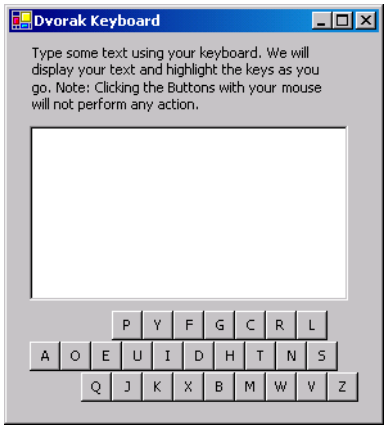


Figure 22.33 Dvorak Keyboard GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial22\Exercises\DvorakKeyboard` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `DvorakKeyboard.sln` in the `DvorakKeyboard` directory to open the application.

c) ***Creating the `KeyPress` event handler.*** Add a `KeyPress` event handler for the `TextBox`.

d) ***Creating a `Select Case` statement.*** Add a `Select Case` statement to the `KeyPress` event handler. The `Select Case` statement should test whether all of the letter keys on the Dvorak keyboard were pressed except for the *S*, *W*, *V* and *Z* keys. If a Dvorak key was pressed, highlight it on the GUI and display the character in the `TextBox`.

e) ***Creating a `KeyDown` event handler.*** Add a `KeyDown` event handler for the `TextBox`. The *S*, *W*, *V* and *Z* keys do not map to a letter key on the standard keyboard; therefore, a `KeyDown` event handler must be used to determine whether one of these keys was pressed.

f) ***Adding a `Select Case` statement.*** Add a `Select Case` statement to your `KeyDown` event handler that determines whether *S*, *W*, *V* or *Z* was pressed. If one of these keys was pressed, highlight the key, and add the character to the `TextBox`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Use your keyboard to enter text. Verify that the text entered is correct based on the rules in the exercise description. Make sure the correct `Buttons` on the `Form` are highlighted as you enter text.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 22.16 Solution
' DvorakKeyboard.vb

Public Class FrmDvorakKeyboard
   Inherits System.Windows.Forms.Form

   ' reference to last Button pressed
   Dim m_btnLastButton As Button

   ' Windows Form Designer generated code

   Private Sub txtOutput_KeyUp(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.KeyEventArgs) _
      Handles txtOutput.KeyUp

```

```
         ResetColor()
      End Sub ' txtOutput_KeyUp

      ' highlight Button passed as argument
      Private Sub ChangeColor(ByVal btnButton As Button)

         ResetColor()
         btnButton.BackColor = Color.LightGoldenrodYellow
         m_btnLastButton = btnButton
      End Sub ' ChangeColor

      ' changes m_btnLastButton's color if it refers to a Button
      Private Sub ResetColor()

         If IsNothing(m_btnLastButton) = False Then
            m_btnLastButton.BackColor = _
               m_btnLastButton.DefaultBackColor

         End If ' m_btnLastButton is not Nothing

      End Sub ' ResetColor

      ' handles Form KeyPress Event
      Private Sub txtOutput_KeyPress(ByVal sender As Object, _
         ByVal e As System.Windows.Forms.KeyPressEventArgs) _
         Handles txtOutput.KeyPress

         ' convert pressed key to uppercase
         Select Case Char.ToUpper(e.KeyChar)

            ' following cases test if key pressed was a letter
            Case Convert.ToChar(Keys.A) ' a maps to a key
               ChangeColor(btnA)
               txtOutput.Text &= "a"

            Case Convert.ToChar(Keys.B) ' x maps to b key
               ChangeColor(btnX)
               txtOutput.Text &= "x"

            Case Convert.ToChar(Keys.C) ' j maps to c key
               ChangeColor(btnJ)
               txtOutput.Text &= "j"

            Case Convert.ToChar(Keys.D) ' e maps to d key
               ChangeColor(btnE)
               txtOutput.Text &= "e"

            Case Convert.ToChar(Keys.F) ' u maps to f key
               ChangeColor(btnU)
               txtOutput.Text &= "u"

            Case Convert.ToChar(Keys.G) ' i maps to g key
               ChangeColor(btnI)
               txtOutput.Text &= "i"

            Case Convert.ToChar(Keys.H) ' d maps to h key
               ChangeColor(btnD)
               txtOutput.Text &= "d"

            Case Convert.ToChar(Keys.I) ' c maps to i key
               ChangeColor(btnC)
```

```
            txtOutput.Text &= "c"

         Case Convert.ToChar(Keys.J) ' h maps to j key
            ChangeColor(btnH)
            txtOutput.Text &= "h"

         Case Convert.ToChar(Keys.K) ' t maps to k key
            ChangeColor(btnT)
            txtOutput.Text &= "t"

         Case Convert.ToChar(Keys.L) ' n maps to l key
            ChangeColor(btnN)
            txtOutput.Text &= "n"

         Case Convert.ToChar(Keys.M) ' m maps to m key
            ChangeColor(btnM)
            txtOutput.Text &= "m"

         Case Convert.ToChar(Keys.N) ' b maps to n key
            ChangeColor(btnB)
            txtOutput.Text &= "b"

         Case Convert.ToChar(Keys.O) ' r maps to o key
            ChangeColor(btnR)
            txtOutput.Text &= "r"

         Case Convert.ToChar(Keys.P) ' l maps to p key
            ChangeColor(btnL)
            txtOutput.Text &= "l"

         Case Convert.ToChar(Keys.R) ' p maps to r key
            ChangeColor(btnP)
            txtOutput.Text &= "p"

         Case Convert.ToChar(Keys.S) ' o maps to s key
            ChangeColor(btnO)
            txtOutput.Text &= "o"

         Case Convert.ToChar(Keys.T) ' y maps to t key
            ChangeColor(btnY)
            txtOutput.Text &= "y"

         Case Convert.ToChar(Keys.U) ' g maps to u key
            ChangeColor(btnG)
            txtOutput.Text &= "g"

         Case Convert.ToChar(Keys.V) ' k maps to v key
            ChangeColor(btnK)
            txtOutput.Text &= "k"

         Case Convert.ToChar(Keys.X) ' q maps to x key
            ChangeColor(btnQ)
            txtOutput.Text &= "q"

         Case Convert.ToChar(Keys.Y) ' f maps to y key
            ChangeColor(btnF)
            txtOutput.Text &= "f"

      End Select ' ends test for letters

   End Sub ' txtOutput_KeyPress
```

```

   ' handles KeyDown event
   Private Sub txtOutput_KeyDown(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.KeyEventArgs) _
      Handles txtOutput.KeyDown

      Select Case e.KeyData

         ' use KeyDown for these keys because the Dvorak
         ' representation does not map to letters of QWERTY keyboard
         Case Keys.OemSemicolon ' s maps to semicolon key
            ChangeColor(btnS)
            txtOutput.Text &= "s"

         Case Keys.Oemcomma ' w maps to comma key
            ChangeColor(btnW)
            txtOutput.Text &= "w"

         Case Keys.OemPeriod ' y maps to period key
            ChangeColor(btnV)
            txtOutput.Text &= "v"

         Case Keys.OemQuestion ' z maps t question mark key
            ChangeColor(btnZ)
            txtOutput.Text &= "z"

      End Select

   End Sub ' txtOutput_KeyDown

End Class ' FrmDvorakKeyboard
```

TUTORIAL 23

Screen Scraping Application

Introducing `String` Processing Solutions

Instructor's Manual Exercise Solutions Tutorial 23

MULTIPLE CHOICE QUESTIONS

23.1 Extracting desired information from Web pages is called _____.

a) Web crawling
b) screen scraping
c) querying
d) redirection

23.2 If `IndexOf` method does not find the specified substring, it returns _____.

a) `False`
b) `0`
c) `-1`
d) None of the above.

23.3 The `String` class allows you to _____ `Strings`.

a) search
b) retrieve characters from
c) replace characters in
d) All of the above.

23.4 _____ is a technology for describing Web content.

a) Class `String`
b) A `String` literal
c) HTML
d) A screen scraper

23.5 The `String` class is located in the _____ namespace.

a) `String`
b) `System.Strings`
c) `System.IO`
d) `System`

23.6 The _____ method creates a new `String` object by copying part of an existing `String` object.

a) `StringCopy`
b) `Substring`
c) `CopyString`
d) `CopySubString`

23.7 All `String` objects are _____.

a) the same size
b) always equal to each other
c) preceded by at least one whitespace character
d) immutable

23.8 The `IndexOf` method does not examine any characters that occur prior to the _____.

a) starting index
b) first match
c) last character of the `String`
d) None of the above.

23.9 The _____ method determines whether a `String` ends with a particular substring.

a) `CheckEnd`
b) `StringEnd`
c) `EndsWith`
d) `EndIs`

23.10 The `Trim` method removes all whitespace characters that appear _____ a `String`.

a) in
b) at the beginning of
c) at the end of
d) at the beginning and end of

Answers: 23.1) b. 23.2) c. 23.3) d. 23.4) c. 23.5) d. 23.6) b. 23.7) d. 23.8) a. 23.9) c. 23.10) d.

EXERCISES

23.11 (*Supply Cost Calculator Application*) Write an application that calculates the cost of all the supplies added to the user's shopping list (Fig. 23.18). The application should contain two `ListBoxes`. The first `ListBox` contains all the supplies offered and their respective prices. Users should be able to select the desired supplies from the first `ListBox` and add them to the second `ListBox`. Provide a **Calculate** `Button` that displays the total price for the user's shopping list (the contents of the second `ListBox`).

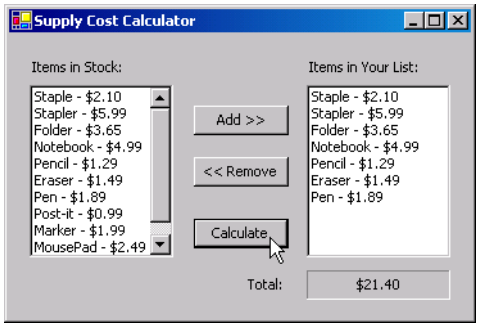


Figure 23.18 Supply Calculator application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial23\Exercises\SupplyCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SupplyCalculator.sln` in the `SupplyCalculator` directory to open the application.

c) ***Adding code to the Add >> `Button`.*** Double click the **Add >>** `Button` to create an empty event handler. Add code to the event handler that adds the selected item from the first `ListBox` to the `lstStock` `ListBox`. Make sure to check that at least one item is selected in the first `ListBox` before attempting to add an item to the `lstStock` `ListBox`.

d) ***Enabling the `Buttons`.*** Once the user adds something to the `lstStock` `ListBox`, set the `Enabled` properties of the **<< Remove** and **Calculate** `Buttons` to `True`.

e) ***Deselecting the items.*** Once the items are added to the `lstStock` `ListBox`, make sure that those items are deselected in the `lstSupply` `ListBox`. Also, clear the **Total:** `Label` to indicate to the user that a new total price must be calculated.

f) ***Adding code to the << Remove `Button`.*** Double click the **<< Remove** `Button` to create an empty event handler. Use a `Do While` loop to remove any selected items in the `lstStock` `ListBox`. Make sure to check that at least one item is selected before attempting to remove an item. [*Hint*: Method `lstStock.Items.RemoveAt(intIndex)` will remove the item located at `intIndex` from the `lstStock` `ListBox`.]

g) ***Adding code to the Calculate `Button`.*** Double click the **Calculate** `Button` to create an empty event handler. Use a `For...Next` statement to loop through all the items in the `lstStock` `ListBox`. Convert each item from the `ListBox` into a `String`. Then use the `String` method `Substring` to extract the price of each item.

h) ***Displaying the total.*** Convert the `String` representing each item's price to a `Decimal`, and add this to the overall total (of type `Decimal`). Remember to output the value in currency format.

i) ***Running the application.*** Select **Debug > Start** to run your application. Use the **Add >>** and **<< Remove** `Buttons` to add and remove items from the **Items in Your List:** `ListBox`. Click the **Calculate** `Button` and verify that the total price displayed is correct.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 23.11 Solution
' SupplyCalculator.vb

Public Class FrmSupplyCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' remove item from shopping list
```

```
   Private Sub btnRemove_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnRemove.Click

      ' if there is at least one item selected
      If lstStock.SelectedIndex <> -1 Then

         ' remove items while there are selected items
         Do While lstStock.SelectedIndex <> -1

            ' remove item at the selected index
            lstStock.Items.RemoveAt(lstStock.SelectedIndex)
         Loop

      Else
         ' display message if there is no item selected
         MessageBox.Show("Please select item to remove", _
            "Error Removing", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
      End If

      ' if there is no item
      If lstStock.Items.Count < 1 Then

         ' disable the Remove and Calculate Buttons
         btnRemove.Enabled = False
         btnCalculate.Enabled = False
      End If

      lblTotal.Text = "" ' clear lblTotal Label
   End Sub ' btnRemove_click

   ' add shopping item to list
   Private Sub btnAdd_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnAdd.Click

      ' if there is at least one item selected
      If lstSupply.SelectedIndex <> -1 Then

         ' add each item to lstStock ListBox
         lstStock.Items.Add(lstSupply.SelectedItem)

         btnRemove.Enabled = True ' enable the Remove Button
         btnCalculate.Enabled = True ' enable the Calculate Button
         lstSupply.SelectedIndex = -1 ' unselect items
         lblTotal.Text = "" ' clear lblTotal

      End If

   End Sub ' btnAdd_Click

   ' calculate total price for shopping list
   Private Sub btnCalculate_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculate.Click

      Dim decTotal As Decimal ' total amount
      Dim strPrice As String ' temporary price variable
      Dim intCounter As Integer ' counter variable

```

```
        ' run through list of item(s)
        For intCounter = 0 To lstStock.Items.Count - 1

           ' retrieve price from items
           strPrice = lstStock.Items(intCounter).ToString

           ' get substring starting after the $
           strPrice = strPrice.Substring(strPrice.IndexOf("$") + 1)

           ' add price of each item to total
           decTotal += Convert.ToDecimal(strPrice)
        Next

        ' display total
        lblTotal.Text = String.Format("{0:C}", decTotal)
     End Sub ' btnCalculate_Click

  End Class ' FrmSupplyCalculator
```

23.12 (*Encryption Application*) Write an application that encrypts a message from the user (Fig. 23.19). The application should be able to encrypt the message in two different ways: substitution cipher and transposition cipher (both described below). The user should be able to enter the message in a `TextBox` and select the desired method of encryption. Display the encrypted message in a `Label`.

In a substitution cipher, every character in the English alphabet is represented by a different character in the substitution alphabet. Every time a letter occurs in the English sentence, it is replaced by the letter in the corresponding index of the substitution string. In a transposition cipher, two `String`s are created. The first new `String` contains all the characters at the even indices of the input `String`. The second new `String` contains all of the characters at the odd indices. The new `String`s are the encrypted text. For example a transposition cipher for the word "code" would be: "cd oe."

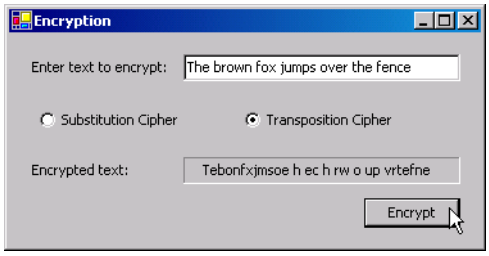


Figure 23.19 Text Encryption application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial23\Exercises\Encryption` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Encryption.sln` in the `Encryption` directory to open the application.

c) ***Adding code to the Encrypt `Button`.*** Double click the **Encrypt** `Button` to create an empty event handler.

d) ***Determine the cipher method.*** Use `If`…`Then`…`Else` statements to determine which method of encryption the user has selected and call the appropriate procedure.

e) ***Locating the `SubstitutionCipher` method.*** Locate the `SubstitutionCipher` procedure. The English and substitution alphabet `String`s have been defined for you in this procedure.

f) ***Converting the text input to lowercase.*** Add code to the `SubstitutionCipher` method that uses the `ToLower` method of class `String` to make all the characters in the input string (`txtPlainText.Text`) lowercase.

g) ***Performing the substitution encryption***. Use nested `For`…`Next` loops to iterate through each character of the input `String`. When each character from the input `String` is found in the `String` holding the English alphabet, replace the character in the input `String` with the character located at the same index in the substitution `String`.

h) ***Display the `String`***. Now that the `String` has been substituted with all the corresponding cipher characters, assign the cipher `String` to the `lblCipherText` `Label`.

i) ***Locating the `TranspositionCipher` method***. Locate the `TranspositionCipher` method. Define three variables—a counter variable and two `String`s (each representing a word).

j) ***Extracting the first word***. Use a `Do While`…`Loop` to retrieve all the "even" indices (starting from 0) from the input `String`. Increment the counter variable by `2` each time, and add the characters located at even indices to the first `String` created in *Step h*.

k) ***Extracting the second word***. Use another `Do While`…`Loop` to retrieve all the "odd" indices (starting from 1) from the same input `String`. Increment the counter variable by `2`, and add the characters at odd indices to the second `String` that you created in *Step h*.

l) ***Output the result***. Add the two `String`s together with a space in between, and output the result to the `lblCipherText` `Label`.

m) ***Running the application.*** Select **Debug > Start** to run your application. Enter text into the **Enter text to encrypt:** `TextBox`. Select the **Substitution Cipher** `RadioButton` and click the **Encrypt** `Button`. Verify that the output is the properly encrypted text using the substitution cipher. Select the **Transposition Cipher** `RadioButton` and click the **Encrypt** `Button`. Verify that the output is the properly encrypted text using the transposition cipher.

n) ***Closing the application.*** Close your running application by clicking its close box.

o) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 23.12 Solution
' Encryption.vb

Public Class FrmEncryption
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' using the substitution cipher
   Private Sub SubstitutionCipher()

      ' normal alphabet String
      Dim strNormalAlphabet As String = _
         "abcdefghijklmnopqrstuvwxyz .!?,"

      ' substitution alphabet String
      Dim strCipherAlphabet As String = _
         "cdefg.hijk!lmn opqr?stuv,wxyzab"

      Dim intIndex1 As Integer ' index variable for For...Next loop
      Dim intIndex2 As Integer ' inner index variable
      Dim strPlain As String ' String entered by the user
      Dim strCipher As String ' encrypted String

      lblCipherText.Text = "" ' clear output TextBox

      ' change all the characters to lower case
      strPlain = txtPlainText.Text.ToLower
```

```

      ' iterate through the length of the String
      For intIndex1 = 0 To txtPlainText.Text.Length - 1

         ' iterate through alphabet and special character( .!?,")
         For intIndex2 = 0 To 30

            ' compare characters
            If strPlain.Chars(intIndex1) = _
               strNormalAlphabet.Chars(intIndex2) Then

               ' build encrypted text
               strCipher &= strCipherAlphabet.Chars(intIndex2)

            End If

         Next

      Next

      lblCipherText.Text = strCipher ' output the encrypted String
   End Sub ' SubstitutionCipher

   ' using the transposition cipher
   Private Sub TranspositionCipher()
      Dim intCounter As Integer = 0 ' counter variable
      Dim strFirstWord As String ' first word
      Dim strLastWord As String ' second word

      ' create first word from the "even" index
      Do While intCounter < txtPlainText.Text.Length

         ' add character from specified location to strFirstWord
         strFirstWord &= txtPlainText.Text.Chars(intCounter)
         intCounter += 2 ' increment counter by 2
      Loop ' loop through the entire String

      ' create second word from the "odd" indices
      intCounter = 1

      Do While intCounter < txtPlainText.Text.Length

         ' add character from specified location to strLastWord
         strLastWord &= txtPlainText.Text.Chars(intCounter)
         intCounter += 2 ' increment counter by 2
      Loop ' loop through the entire String

      ' output encrypted text
      lblCipherText.Text = strFirstWord & " " & strLastWord
   End Sub ' TranspositionCipher

   ' encrypt a String of characters
   Private Sub btnEncrypt_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEncrypt.Click

      ' determine the selected RadioButton
      If radSubstitution.Checked = True Then
         SubstitutionCipher() ' call SubstitutionCipher
      Else
         TranspositionCipher() ' call TranspositionCipher
      End If
```

```
    End Sub ' btnEncrypt_Click

End Class ' FrmEncryption
```

23.13 (*Anagram Game Application*) Write an **Anagram Game** that contains an array of pre-set words (Fig. 23.20). The game should randomly select a word and scramble its letters. A `Label` displays the scrambled word for the user to guess. If the user guesses correctly, display a message, and repeat the process with a different word. If the guess is incorrect, display a message, and let the user try again.

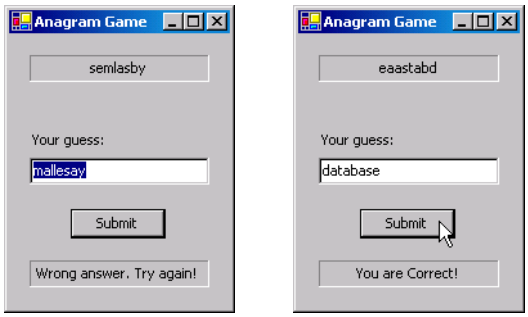


Figure 23.20 **Anagram Game** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial23\Exercises\Anagram` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `Anagram.sln` in the `Anagram` directory to open the application.

c) ***Locating the `GenerateAnagram` method.*** Locate the `GenerateAnagram` method. It is the first method after the `FrmAnagram_Load` event handler.

d) ***Picking a random word.*** Generate a random number to use as the index of the word in the `m_strAnagram` array. Retrieve word from the `m_strAnagram` array, using the first random number as an index. Store the word in another `String` variable. Generate a second random number to store the index of a character to be moved.

e) ***Generate the scrambled word.*** Use a `For`…`Next` statement to iterate through the word 20 times. Each time the loop executes, pass the second random number created in *Step c* to the `Chars` property of class `String`. Append the character returned by `Chars` to the end of the `String`, and remove it from its original position. Next, generate a new random number to move a different character during the next iteration of the loop. Remember to output the final word to the `lblAnagram` `Label`.

f) ***Defining the Submit `Button`.*** Double click the **Submit** `Button` to generate an empty event handler.

g) ***Testing the user's input.*** Use an `If`…`Then`…`Else` statement to determine whether the user's input matches the actual word. If the user is correct, clear and place the focus on the `TextBox` and generate a new word. Otherwise, select the user's text and place focus on the `TextBox`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Submit correct answers and incorrect answers, and verify that the appropriate message is displayed each time.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 23.13 Solution
' Anagram.vb

Public Class FrmAnagram
   Inherits System.Windows.Forms.Form
```

```

    ' Windows Form Designer generated code

    ' array of words to be scrambled
    Private m_strAnagram As String() = New String() {"controls", _
       "events", "properties", "visual", "program", "application", _
       "basic", "debugger", "database", "files", "inheritance", _
       "assembly", "multimedia", "procedures", "functions", _
       "arrays", "strings", "collections", "integration", _
       "structures"}

    Private m_objRandom As Random = New Random ' random number
    Private m_intRandomNumber As Integer ' random index variable
    Private m_strScrambled As String ' randomly chosen word

    ' generate new scrambled word
    Private Sub FrmAnagram_Load(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles MyBase.Load

       GenerateAnagram() ' generate new word
    End Sub ' FrmAnagram_Load

    ' scramble words
    Private Sub GenerateAnagram()

       ' generate new random number
       m_intRandomNumber = m_objRandom.Next(0, 19)

       ' select new word from array with m_intRandomNumber index
       m_strScrambled = m_strAnagram(m_intRandomNumber)

       ' generate new random index
       Dim intRandomIndex As Integer = _
          m_objRandom.Next(0, m_strScrambled.Length - 1)

       Dim intCounter As Integer ' loop counter variable

       ' loop to generate scrambled word
       For intCounter = 0 To 20

          ' attach character at the end of string
          m_strScrambled &= m_strScrambled.Chars(intRandomIndex)

          ' remove character from the word
          m_strScrambled = m_strScrambled.Remove(intRandomIndex, 1)

          ' new random index
          intRandomIndex = _
             m_objRandom.Next(0, m_strScrambled.Length - 1)

       Next

       lblAnagram.Text = m_strScrambled ' display scrambled word
    End Sub ' GenerateAnagram

    ' check if the user's answer is correct
    Private Sub btnSubmit_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnSubmit.Click

```

```
        ' answer is correct
        If txtGuess.Text = m_strAnagram(m_intRandomNumber) Then
           lblResult.Text = "You are Correct!"
           GenerateAnagram() ' generate new word
           txtGuess.Clear() ' clear the TextBox
           txtGuess.Focus() ' place focus on TextBox
        Else

           ' answer is incorrect
           lblResult.Text = "Wrong answer. Try again!"
           txtGuess.Focus() ' place focus on TextBox
           txtGuess.SelectAll() ' select the answer
        End If

     End Sub ' btnSubmit_Click

  End Class ' FrmAnagram
```

What does this code do?

23.14 What is assigned to `strResult` when the following code executes?

```
Dim strWord1 As String = "CHORUS"
Dim strWord2 As String = "d i n o s a u r"
Dim strWord3 As String = "The theme is string."
Dim strResult As String

strResult = strWord1.ToLower()
strResult = strResult.Substring(4)
strWord2 = strWord2.Replace(" ", "")
strWord2 = strWord2.Substring(4, 4)
strResult = strWord2 & strResult

strWord3 = strWord3.Substring(strWord3.IndexOf(" ") + 1, 3)

strResult = strWord3.Insert(3, strResult)
```

Answer: After assigning initial values to `String`s `strWord1`, `strWord2` and `strWord3`, the code above changes the word `"CHORUS"` to all lowercase letters, resulting in `"chorus"` being assigned to `strResult`. `strResult` then is assigned the substring of `"chorus"` beginning at the fifth character (index 4), `"u"`, resulting in `strResult`'s value as `"us"`. The next line of code deletes spaces from `strWord2`, resulting in the word `"dinosaur"`. The substring of `"dinosaur"` beginning at the fifth character and of length 4 (`"saur"`) is then assigned to `strWord2`. Then the value of `strResult` (`"us"`) is appended to the end of `strWord2` (`"us"`) and placed in `strResult`, yielding `"saurus"`. Following that, `strWord3` is assigned a substring of itself beginning one character after the first space character and of length 3 (`"the"`). Finally, the value `"saurus"` (`strResult`) is inserted into `strWord3` at the location of the fourth character (the end of the string) and assigned to `strResult`. The final value of `strResult` is the word `"thesaurus"`.

What's wrong with this code?

23.15 This code should remove all commas from `strTest` and convert all lowercase letters to uppercase letters. Find the error(s) in the following code.

```
Dim strTest As String = "Bug,2,Bug"

strTest = strTest.ToUpper()
strTest = strTest.Replace("")
```

Answer: `Replace` method takes two arguments: one substring to search for, and another substring to replace all matching occurrences of the first argument. The proper call to method `Replace` is shown in the following code.

```
Dim strTest As String = "Bug,2,Bug"

strTest = strTest.ToUpper()
strTest = strTest.Replace(",", "")
```

Programming Challenge

23.16 (***Pig Latin Application***) Write an application that encodes English language phrases into pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following method to form the pig Latin words:

> *To form a pig Latin word from an English-language phrase, the translation proceeds one word at a time. To translate an English word into a pig Latin word, place the first letter of the English word (if it is not a vowel) at the end of the English word and add the letters "ay." If the first letter of the English word is a vowel, place it at the end of the word and add "y." Using this method, the word "jump" becomes "umpjay", the word "the" becomes "hetay" and the word "ace" becomes "ceay." Blanks between words remain blanks.*

Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks, and all words have two or more letters. Enable the user to input a sentence. The `TranslateToPigLatin` method should translate the sentence into pig Latin, word by word. [*Hint*: You will need to use the `Join` and `Split` methods of class `String` demonstrated in Fig. 23.16 to form the pig Latin phrases].

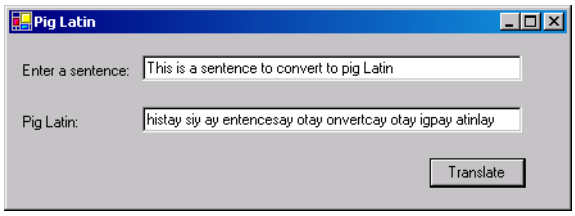


Figure 23.21 Pig Latin Application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial23\Exercises\PigLatin` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `PigLatin.sln` in the `PigLatin` directory to open the application.

c) ***Splitting the sentence.*** Use method `Split` on the `String` passed to the `TranslateToPigLatin` method. Assign the result of this operation to `strWords`.

d) ***Retrieving the word's first letter.*** Declare a `For…Next` loop that iterates through your array of words. As you iterate through the array, store each word's first letter in `strTemporary`.

e) ***Determining the suffix.*** Use `If…Then…Else` statements to determine the suffix for each word. Store this suffix in `strSuffix`.

f) ***Generating new words.*** Generate the new words by arranging each word's pieces in the proper order.

g) ***Returning the new sentence.*** When the `For…Next` loop finishes, use method `Join` to combine all of the elements in `strWords`, and `Return` the new pig Latin sentence.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a sentence and click the **Translate** `Button`. Verify that the sentence is correctly converted into Pig Latin.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 23.16 Solution
' PigLatin.vb

Public Class FrmPigLatin
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' receive sentence from user and send to TranslateToPigLatin
   Private Sub btnTranslate_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnTranslate.Click

      ' retrieve English phrase from user
      Dim strPhrase As String = txtInput.Text

      ' display output
      txtOutput.Text = TranslateToPigLatin(strPhrase)
   End Sub ' btnTranslate_Click

   ' translates the string input by the user
   ' from English to pig Latin
   Private Function TranslateToPigLatin( _
      ByVal strEnglishPhrase As String) As String

      Dim strWords As String() ' array to hold each word
      Dim strSuffix As String ' suffix for the end of each word
      Dim intIndex As Integer ' index to iterate through the array
      Dim strTemporary As String ' temporary string

      strWords = strEnglishPhrase.Split() ' split words

      For intIndex = 0 To strWords.Length - 1

         ' get first letter of each word
         strTemporary = strWords(intIndex).Substring(0, 1).ToLower

         ' check if each word starts with a vowel
         If strTemporary = "a" OrElse _
            strTemporary = "e" OrElse _
            strTemporary = "i" OrElse _
            strTemporary = "o" OrElse _
            strTemporary = "u" Then

            strSuffix = "y"
         Else ' if not, suffix is different
            strSuffix = "ay"
         End If

         ' swap letters to create new word
         strWords(intIndex) = strWords(intIndex).Substring(1) & _
            strTemporary & strSuffix

      Next

      ' put words together and return the whole sentence
      Return String.Join(" ", strWords)
   End Function ' TranslateToPigLatin
```

```

End Class ' FrmPigLatin
```

TUTORIAL 24

Ticket Information Application

Introducing Sequential-Access Files
Solutions

Instructor's Manual Exercise Solutions Tutorial 24

MULTIPLE-CHOICE QUESTIONS

24.1 Data maintained in a file is called _____.

a) persistent data b) bits
c) secondary data d) databases

24.2 Methods from the _____ class can be used to write data to a file.

a) `StreamReader` b) `FileWriter`
c) `StreamWriter` d) `WriteFile`

24.3 Namespace _____ provides the classes and methods that you need to use to perform file processing.

a) `System.IO` b) `System.Files`
c) `System.Stream` d) `System.Windows.Forms`

24.4 Sometimes a group of related files is called a _____.

a) field b) database
c) collection d) byte

24.5 A(n) _____ allows the user to select a file to open.

a) `CreateFileDialog` b) `OpenFileDialog`
c) `MessageBox` d) None of the above.

24.6 Digits, letters and special symbols are referred to as _____.

a) constants b) `Integers`
c) `Strings` d) characters

24.7 The _____ method reads a line from a file.

a) `ReadLine` b) `Read`
c) `ReadAll` d) `ReadToNewline`

24.8 A _____ contains information that is read in the order that it was written to the file.

a) sequential-access file b) text file
c) `StreamReader` d) `StreamWriter`

24.9 The smallest data item that a computer can support is called a _____.

a) character set b) character
c) special symbol d) bit

24.10 Methods from the _____ class can be used to read data from a file.

a) `StreamWriter` b) `FileReader`
c) `StreamReader` d) `ReadFile`

Answers: 24.1) a. 24.2) c. 24.3) a. 24.4) b. 24.5) b. 24.6) d. 24.7) a. 24.8) a. 24.9) d. 24.10) c.

EXERCISES

24.11 (*Birthday Saver Application*) Create an application that stores people's names and birthdays in a file (Fig. 24.35). The user creates a file and inputs each person's first name, last name and birthday on the `Form`. The information is then written to the file.

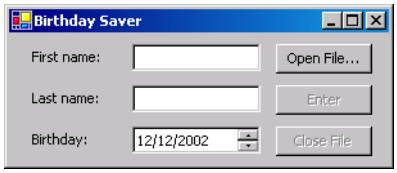


Figure 24.35 Birthday Saver application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial24\Exercises\BirthdaySaver` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `BirthdaySaver.sln` in the `BirthdaySaver` directory to open the application (Fig. 24.35).

c) ***Adding and customizing an `OpenFileDialog` component.*** Add an `OpenFileDialog` component to the `Form`. Change its `Name` property to `objOpenFileDialog`. Set the `CheckFileExists` property to `False`.

d) ***Importing namespace `System.IO`.*** Import `System.IO` to allow file processing.

e) ***Declaring a `StreamWriter` object.*** Declare a `StreamWriter` object that can be used throughout the entire class.

f) ***Defining the Open File... `Button`'s `Click` event handler.*** Double click the **Open File...** `Button` to create the `btnOpen_Click` event handler. Write code to display the **Open** dialog. If the user clicks the **Cancel** `Button` in the dialog, then the event handler should perform no further actions. Otherwise, determine whether the user provided a file name that has the `.txt` extension (indicating a text file). If the user did not, display a `MessageBox` asking the user to select an appropriate file. If the user specified a valid file name, perform *Step f*.

g) ***Initializing the `StreamWriter`.*** Initialize the `StreamWriter` in the `btnOpenFile_Click` event handler, passing the user-input file name as an argument. Allow the user to append information to the file by passing the `Boolean` value `True` as the second argument to the `StreamWriter`.

h) ***Defining the Enter `Button`'s `Click` event handler.*** Double click the **Enter** `Button` to create the event handler `btnEnter_Click`. This event handler should write the entire name of the person on one line in the file. Then the person's birthday should be written on the next line in the file. Finally, the `TextBoxes` on the `Form` should be cleared, and the `DateTimePicker`'s value should be set back to the current date.

i) ***Defining the Close File `Button`'s `Click` event handler.*** Double click the **Close File** `Button` to create the `btnClose_Click` event handler. Close the `StreamWriter` connection in this event handler.

j) ***Running the application.*** Select **Debug > Start** to run your application. Open a file by clicking **Open File...** `Button`. After a file has been opened, use the input fields provided to enter birthday information. After each person's name and birthday are typed in, click the **Enter** `Button`. When you are finished, close the file by clicking the **Close File** `Button`. Browse to the file and ensure that its contents contain the birthday information that you entered.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 24.11 Solution
' BirthdaySaver.vb

Imports System.IO

Public Class FrmBirthdaySaver
   Inherits System.Windows.Forms.Form

   ' StreamWriter used to write to file
```

```
   Private m_objOutput As StreamWriter

   ' Windows Form Designer generated code

   ' Open File... Button's Click event
   Private Sub btnOpen_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnOpen.Click

      ' display Open File... dialog
      Dim result As DialogResult = objOpenFileDialog.ShowDialog()

      ' exit event handler if user clicked Cancel Button
      If result = DialogResult.Cancel Then
         Return
      End If

      ' get specified file name
      Dim strFileName As String = objOpenFileDialog.FileName

      ' show error if user specified invalid file
      If strFileName.EndsWith(".txt") = False Then
         MessageBox.Show("File name must end with .txt", _
            "Invalid File Type", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)

      Else
         btnOpen.Enabled = False
         btnEnter.Enabled = True
         btnClose.Enabled = True

         m_objOutput = New StreamWriter(strFileName, True)
      End If

   End Sub ' btnOpen_Click

   ' clear all user input
   Sub ClearUserInput()
      txtFirstName.Clear()
      txtLastName.Clear()
      dtpBirthday.Value = Date.Now
   End Sub ' ClearUserInput

   ' handles Enter Button's Click event
   Private Sub btnEnter_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEnter.Click

      ' write user input to file
      m_objOutput.WriteLine(txtFirstName.Text & " " & _
         txtLastName.Text)
      m_objOutput.WriteLine(dtpBirthday.Value.Month & "/" & _
         dtpBirthday.Value.Day & "/" & dtpBirthday.Value.Year)
      ClearUserInput()
   End Sub ' btnEnter_Click

   ' handles Close File Button's Click event
   Private Sub btnClose_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClose.Click

      m_objOutput.Close()  ' close stream
      btnOpen.Enabled = True
      btnEnter.Enabled = False
```

```
        btnClose.Enabled = False
    End Sub ' btnClose_Click

End Class ' FrmBirthdaySaver
```

24.12 (*Photo Album Application*) Create an application that displays images for the user, as shown in Fig. 24.36. This application should display the current image in a large `PictureBox` and display the previous and next images in smaller `PictureBoxes`. A description of the book represented by the large image should be displayed in a multiline `TextBox`. The application should use the `Directory` class's methods to facilitate the displaying of the images.

Figure 24.36 Photo Album application GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial24\Exercises\PhotoAlbum` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `PhotoAlbum.sln` in the `PhotoAlbum` directory to open the application.

c) ***Creating instance variables.*** Create instance variable `m_intCurrent` to represent the current image that is displayed, and set the variable to `0`. Create the `m_strLargeImage` array (to store the path names of five large images), the `m_strSmallImage` array (to store the path names of five small images) and the `m_strDescriptions` array (to store the descriptions of the five books represented by the images).

d) ***Defining the `RetrieveData` procedure.*** Create a `Sub` procedure named `RetrieveData` to store the path names of the larger images in `m_strLargeImages` and the path names of the smaller images in `m_strSmallImage`. Use the `Directory` class's `GetCurrentDirectory` method to determine the directory path for the `images\large` and `images\small` folders. Sequential-access file `books.txt` stores the file name of each image. The file is organized such that the file name of the small and large images are on the first line. These files have similar names. The small image's file name ends with _thumb.jpg (that is, *filename*`_thumb.jpg`) while the large image's file name ends with _large.jpg (that is, *filename*`_large.jpg`). The description of the book, which should be stored in array `m_strDescriptions`, follows the file name.

e) ***Defining the `DisplayPicture` procedure.*** Create a `Sub` procedure named `DisplayPicture` to display the current image in the large `PictureBox` and to display the previous and next images in the smaller `PictureBoxes`.

f) ***Using `If`…`Then`…`Else` in the `DisplayPicture` procedure.*** Use an `If`…`Then`…`Else` statement to display the images on the `Form`. If the `Integer` instance variable is `0`, display the image of the first book. Also, display the next book's image in the next image `PictureBox`. However, because there is no previous image, nothing should be displayed in the previous image `PictureBox`, and the **Previous Image** `Button` should be disabled. If the last image is displayed in the large `PictureBox`, then disable the **Next Image** `Button`, and do not display anything in the next image `PictureBox`. Otherwise, all three `PictureBoxes` should display their corresponding images, and the **Previous Image** and **Next Image** `Buttons` should be enabled.

g) ***Defining the `FrmPhotoAlbum_Load` event handler.*** Double click the `Form` to create the `FrmPhotoAlbum_Load` event handler. Invoke methods `RetrieveData` and `DisplayPicture` in this event handler.

h) ***Defining the `btnPrevious_Click` event handler.*** Double click the **Previous Image** `Button` to create the `btnPrevious_Click` event handler. In this event handler, decrease the `Integer` instance variable by `1` and invoke procedure `DisplayPicture`.

i) ***Defining the `btnNext_Click` event handler.*** Double click the **Next Image** `Button` to create the `btnNext_Click` event handler. In this event handler, increment the `Integer` instance variable by `1` and invoke the `DisplayPicture` procedure.

j) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Previous Image** and **Next Image** `Buttons` to ensure that the proper images and descriptions are displayed.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 24.12 Solution
' PhotoAlbum.vb

Imports System.IO

Public Class FrmPhotoAlbum
   Inherits System.Windows.Forms.Form

   ' represents current image's index
   Private m_intCurrent As Integer = 0

   Private m_strLargeImage As String() = New String(5) {}
   Private m_strSmallImage As String() = New String(5) {}
   Private m_strDescriptions As String() = New String(5) {}

   ' Windows Form Designer generated code

   ' handles Form's Load event
   Private Sub FrmPhotoAlbum_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      RetrieveData()
      DisplayPicture()  ' display first image
   End Sub ' FrmPhotoAlbum_Load

   ' handles Previous Image Button's Click event
   Private Sub btnPrevious_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnPrevious.Click

      m_intCurrent -= 1
      DisplayPicture()  ' display new images
   End Sub ' btnPrevious_Click

   ' handles Next Image Button's Click event
```

```
   Private Sub btnNext_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNext.Click

      m_intCurrent += 1
      DisplayPicture()  ' display new images
   End Sub ' btnNext_Click

   ' extract descriptions from file and images from the directory
   Private Sub RetrieveData()

      ' create directory path for large images
      Dim strLargeDirectory As String = _
         (Directory.GetCurrentDirectory & "\images\large\")

      ' create directory path for small images
      Dim strSmallDirectory As String = _
         (Directory.GetCurrentDirectory & "\images\small\")

      ' initialize StreamReader to read lines from file
      Dim objInput As StreamReader = New StreamReader("books.txt")

      ' read first image name before entering loop
      Dim strImageName As String = objInput.ReadLine

      Dim intCounter As Integer = 0

      ' loop through lines in file
      Do While strImageName <> ""

         m_strLargeImage(intCounter) = (strLargeDirectory _
            & strImageName & "_large.jpg")
         m_strSmallImage(intCounter) = (strSmallDirectory _
            & strImageName & "_thumb.jpg")
         m_strDescriptions(intCounter) = objInput.ReadLine

         ' read next line in file
         strImageName = objInput.ReadLine

         intCounter += 1
      Loop

   End Sub ' RetrieveData

   ' displays images
   Private Sub DisplayPicture()

      ' set main image
      picMain.Image = _
            Image.FromFile(m_strLargeImage(m_intCurrent))

      ' if index is 0 (first image), do not show previous image
      If m_intCurrent = 0 Then
         picPrevious.Image = Nothing ' do not show previous image

         ' preview next image
         picNext.Image = _
            Image.FromFile(m_strSmallImage(m_intCurrent + 1))

         btnPrevious.Enabled = False  ' disable Previous Button

      ' if index corresponds to last item in array,
```

```
         ' do not show next image
         ElseIf m_intCurrent = m_strLargeImage.GetUpperBound(0) Then
            picPrevious.Image = _
               Image.FromFile(m_strSmallImage(m_intCurrent - 1))

            picNext.Image = Nothing  ' do not show Next image
            btnNext.Enabled = False  ' disable Next Button

         ' show previous, current and next image
         Else
            picPrevious.Image = _
               Image.FromFile(m_strSmallImage(m_intCurrent - 1))

            picNext.Image = _
               Image.FromFile(m_strSmallImage(m_intCurrent + 1))

            ' enable Buttons
            btnPrevious.Enabled = True
            btnNext.Enabled = True
         End If

         ' set description
         txtDescription.Text = m_strDescriptions(m_intCurrent)
      End Sub ' DisplayPicture

   End Class ' FrmPhotoAlbum
```

24.13 (*Car Reservation Application*) Create an application that allows a user to reserve a car for the specified day. The small car reservation company can rent out only four cars per day. Let the application allow the user to specify a certain day. If four cars have already been reserved for that day, then indicate to the user that no vehicles are available.

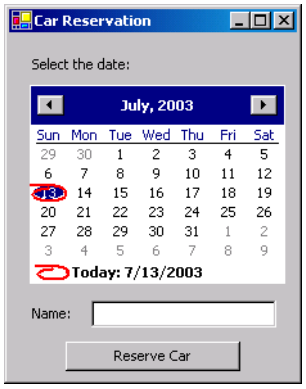


Figure 24.37 CarReservation application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial24\Exercises\CarReservation` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CarReservation.sln` in the `CarReservation` directory to open the application.

c) ***Adding a `MonthCalendar` control to the `Form`.*** Drag and drop a `MonthCalendar` control on the `Form`. Set the `Location` property of the control to `16, 32`.

d) ***Importing `System.IO` namespace.*** Import namespace `System.IO` to allow file processing.

e) ***Defining the `FrmReserve_Load` event handler.*** Double click the `Form` to create the `FrmReserve_Load` event handler.

f) ***Defining a `Function` procedure.*** Create a `Function` procedure named `NumberOfReservations` that takes one argument of type `Date`. The procedure should create a `StreamReader` that reads from the `reservations.txt` file. Use a `Do While Loop` to allow the `StreamReader` to search through the entire `reservations.txt` file to see how many cars have been rented for the day selected by the user. The procedure should close the `StreamReader` connection and return the number of cars rented for the day selected.

g) ***Defining a `Sub` procedure.*** Create a `Sub` procedure named `CheckReservations`. This procedure should invoke the `NumberOfReservations` `Function` procedure, passing in the user-selected day as an argument. The `CheckReservations` method should then retrieve the number returned by `NumberOfReservations` and determine if four cars have been rented for that day. If four cars have been rented, then display a message dialog to the user stating that no cars are available that day for rental. If fewer than four cars have been rented for that day, create a `StreamWriter` object, passing `reservations.txt` as the first argument `True` as the second argument. Write the day and the user's name to the `reservations.txt` file and display a message dialog to the user stating that a car has been reserved.

h) ***Defining the `btnReserve_Click` event handler.*** Double click the **Reserve Car** `Button` to create the `btnReserve_Click` event handler. In this event handler, invoke the `CheckReservations` procedure and clear the **Name:** `TextBox`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter several reservations, including four reservations for the same day. Enter a reservation for a day that already has four reservations to ensure that a message dialog will be displayed.

j) ***Closing the application.*** Close your running application by clicking its close box. Open `reservations.txt` to ensure that the proper data has been stored (based on the reservations entered in *Step i*).

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 24.13 Solution
' CarReservation.vb

Imports System.IO

Public Class FrmReserve
   Inherits System.Windows.Forms.Form

  ' Windows Form Designer generated code

  ' method to determine number of cars rented for that day
   Function NumberOfReservations(ByVal objDay As Date) As Integer

      ' set to the value of selected day in month view
      Dim intChosenDay As Integer = objDay.Day
      Dim intFileDay As String
      Dim intCars As Integer = 0

      ' StreamReader reads lines from file to strLine
      Dim objFile As StreamReader
      Dim strLine As String

      ' initialize StreamReader
      objFile = New StreamReader("reservations.txt")

      ' read the first line before entering loop
      strLine = objFile.ReadLine

      ' loop through all file data
```

```
      Do While strLine <> ""

         intFileDay = strLine

         ' if days match, increment count
         If intFileDay = intChosenDay.ToString Then
            intCars += 1

         End If

         ' read name in
         objFile.ReadLine()

         ' read day of next event in file
         strLine = objFile.ReadLine
      Loop

      objFile.Close()

      Return intCars
   End Function ' NumberOfReservations

   ' method to check reservations for chosen day
   Sub CheckReservations()

      Dim intCount As Integer
      Dim intReservations As Integer

      ' gets data for selected day and stores in intReservations
      intReservations = NumberOfReservations(mvwDate.SelectionStart)

      ' determine if user can reserve car that day
      If intReservations >= 4 Then
         MessageBox.Show("Sorry, all cars have been reserved" & _
            " for this day. Please select another day.", _
            "No cars available.", _
            MessageBoxButtons.OK, MessageBoxIcon.Information)

      Else

         ' create StreamWriter to write to file
         Dim objWrite As StreamWriter = _
            New StreamWriter("Reservations.txt", True)

         objWrite.WriteLine(mvwDate.SelectionStart.Day)
         objWrite.WriteLine(txtName.Text)
         objWrite.Close()

         MessageBox.Show("A car has been reserved for you", _
            "Car reserved", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)
      End If

   End Sub ' CheckReservations

   ' invoke when Reserve Button clicked
   Private Sub btnReserve_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnReserve.Click

      ' determine if name was provided
      If txtName.Text = "" Then
```

```
            MessageBox.Show("You must provide a name", _
               "Name Required.", MessageBoxButtons.OK, _
               MessageBoxIcon.Information)
         Else
            CheckReservations()
            txtName.Clear()
         End If
      End Sub ' btnReserve_Click

   End Class ' FrmReserve
```

What does this code do?

24.14 What is the result of the following code? Assume the application imports namespace `System.IO`.

```
Dim strPath1 As String = "oldfile.txt"
Dim strPath2 As String = "newfile.txt"
Dim strLine As String

Dim objStreamWriter As StreamWriter
objStreamWriter = New StreamWriter(strPath2)

Dim objStreamReader As StreamReader
objStreamReader = New StreamReader(strPath1)

strLine = objStreamReader.ReadLine()

Do While strLine <> ""
   objStreamWriter.WriteLine(strLine)
   strLine = objStreamReader.ReadLine()
Loop

objStreamWriter.Close()
objStreamReader.Close()
```

Answer: The code copies the contents of `oldfile.txt` to `newfile.txt`. After the file transfer is completed, the files are closed by closing the `StreamReader` and `StreamWriter`. Note that if a blank line is encountered, copying stops at the blank line.

What's wrong with this code?

24.15 Find the error(s) in the following code, which is supposed to read a line from `somefile.txt`, convert the line to uppercase and then append it to `somefile.txt`. Assume the application imports namespace `System.IO`.

```
Dim strPath As String = "somefile.txt"
Dim strContents As String

Dim objStreamWriter As StreamWriter
objStreamWriter = New StreamWriter(strPath, True)

Dim objStreamReader As StreamReader
objStreamReader = New StreamReader(strPath)

strContents = objStreamReader.ReadLine

strContents = strContents.ToUpper()

objStreamWriter.Write(strContents)

```

```
objStreamWriter.Close()
objStreamReader.Close()
```

Answer: Once the `StreamWriter` opens the file specified by `strPath`, the file is marked open. While open, no other object can open the file. Thus an exception is thrown when `StreamReader` tries to open the same file. A solution to this problem involves performing the reading operations before the writing operations. Notice in the code below that the `StreamReader` is closed before the `StreamWriter` is opened.

```
Dim strPath As String = "somefile.txt"
Dim strContents As String

Dim objStreamReader As StreamReader
objStreamReader = New StreamReader(strPath)

strContents = objStreamReader.ReadLine

strContents = strContents.ToUpper()

objStreamReader.Close()

Dim objStreamWriter As StreamWriter
objStreamWriter = New StreamWriter(strPath, True)

objStreamWriter.Write(strContents)

objStreamWriter.Close()
```

Programming Challenge

24.16 (*File Scrape Application*) Create an application similar to the screen scraping application of Tutorial 23, that opens a user-specified file and searches the file for the price of a book, returning it to the user (Fig. 24.38). [*Hints*: You will need to use the `ReadToEnd` method of class `StreamReader` to retrieve the entire contents of the files. The book price appears, for example, in the sample `booklist.htm` file as `Our Price: <b>$59.99</b>`.]

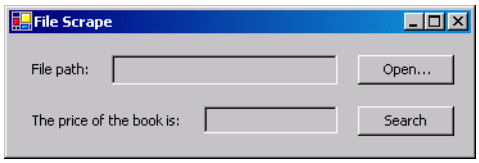


Figure 24.38 File scrape application GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial24\Exercises\FileScrape` directory to your `C:\SimplyVB` directory. Notice that two HTML files—`booklist.htm` and `bookpool.htm`—are provided for you.

b) ***Opening the application's template file.*** Double click `FileScrape.sln` in the `FileScrape` directory to open the application.

c) ***Creating an event handler.*** Create an event handler for the **Open...** `Button` that allows the user to select a file to search for prices.

d) ***Creating a second event handler.*** Create an event handler for the **Search** `Button`. This event handler should search the specified HTML file for the book price. When the price is found, display it in the output `Label`.

e) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Open...** `Button` and select one of the `.htm` files provided in the `FileScrape` directory. Click the **Search** `Button` and view the price of the book. For `booklist.htm`, the price should be $59.99 and `bookpool.htm` the price should be $39.50.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 24.16 Solution
' FileScrapeFig. 24.35.vb

Imports System.IO

Public Class FrmScreenScrape
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' display OpenFile dialog when Open... Button is clicked
   Private Sub btnOpen_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnOpen.Click

      Dim result As DialogResult

      ' show dialog to user for selecting file
      result = objOpenFileDialog.ShowDialog()

      If result = DialogResult.Cancel Then

         Return  ' if user cancels, do nothing
      Else

         ' store file name that user selected
         txtPath.Text = objOpenFileDialog.FileName
      End If

   End Sub ' btnOpen_Click

   ' search file for price
   Private Sub btnSearch_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSearch.Click

      Dim strFilePath As String
      Dim intMatchLocation As Integer
      Dim intRankBegin As Integer
      Dim intRankEnd As Integer

      ' if TextBox is empty, show an error message and return
      If txtPath.Text = "" Then
         MessageBox.Show("No path selected", _
            "Path Not Chosen", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
         Return
      Else

         ' set filepath to value in txtPath
         strFilePath = txtPath.Text
      End If

      ' create stream reader to open and read text file
      Dim objReader As StreamReader = New StreamReader(strFilePath)

      ' read file from beginning to end
      Dim strContents As String = objReader.ReadToEnd()
```

```
57
58          ' close the file to free resource
59          objReader.Close()
60
61          ' find locations of text in file
62          intMatchLocation = strContents.IndexOf("Our Price:", 0)
63          intRankBegin = strContents.IndexOf("$", intMatchLocation)
64          intRankEnd = strContents.IndexOf("<", intRankBegin)
65
66          ' extract price from file
67          txtPrice.Text = strContents.Substring(intRankBegin, _
68             (intRankEnd - intRankBegin))
69
70       End Sub ' btnSearch_Click
71
72    End Class ' FrmScreenScrape
```

TUTORIAL 25

ATM Application

Introducing Database Programming
Solutions

Instructor's Manual Exercises for Tutorial 25

[*Note:* The solutions for this tutorial use relative paths when connecting to a database. We have done this so that the solutions can be run from any location. The student has only learned to connect to databases using absolute paths, so their solutions must be run from a specific location.]

MULTIPLE-CHOICE QUESTIONS

25.1 A _____ provides mechanisms for storing and organizing data in a manner that is consistent with a database's format.

a) relational database
b) connection object
c) data command
d) database management system

25.2 An entire row in a database table is known as a _____.

a) record
b) field
c) column
d) primary key

25.3 A primary key is used to _____.

a) create rows in a database
b) identify fields in a database
c) distinguish between records in a table
d) read information from a database

25.4 A data command object allows you to _____.

a) connect to a database
b) read information from a database
c) execute a statement to retrieve or modify a database
d) create a database

25.5 A data reader can _____.

a) retrieve information from a database
b) modify information stored in a database
c) establish a connection to a database
d) close a connection to a database

25.6 In a `SELECT` statement, what follows the `SELECT` keyword?

a) the name of the table
b) the name of the field
c) the name of the database
d) the criteria that the record must meet

25.7 What does the following `SELECT` statement do?

```
SELECT Age FROM People WHERE LastName = 'Purple'
```

a) It selects the age of the person (or people) with the last name `Purple` from the `People` table.
b) It selects the value `Purple` from the `Age` table of the `People` database.
c) It selects the age of the person with the last name `Purple` from the `People` database.
d) It selects the `People` field from the `Age` table with the `LastName` value `Purple`.

25.8 The SQL _____ modifies information in a database.

a) `SELECT` statement
b) `MODIFY` statement
c) `CHANGE` statement
d) `UPDATE` statement

25.9 Which of the following statements modifies the `Accounts` table's PIN field? The account number is 2?

a) `SELECT PIN FROM Accounts WHERE AccountNumber = 2`
b) `SELECT Accounts FROM AccountNumber = 2 WHERE PIN`
c) `UPDATE Accounts SET PIN=1243 WHERE AccountNumber = 2`
d) `UPDATE PIN=1243 SET AccountNumber = 2 WHERE Accounts`

25.10 A _____ is an organized collection of data.

a) record
b) database
c) data reader
d) primary key

Answers: 25.1) d. 25.2) a. 25.3) c. 25.4) c. 25.5) a. 25.6) b. 25.7) a. 25.8) d. 25.9) c. 25.10) b.

EXERCISES

25.11 ***(Stock Portfolio Application)*** A stock broker wants an application that will display a client's stock portfolio (Fig. 25.37). All the companies that the user holds stock in should be displayed in a `ComboBox` when the application is loaded. When the user selects a company from the `ComboBox` and clicks the **Stock Information** `Button`, the stock information for that company should be displayed in `Label`s.

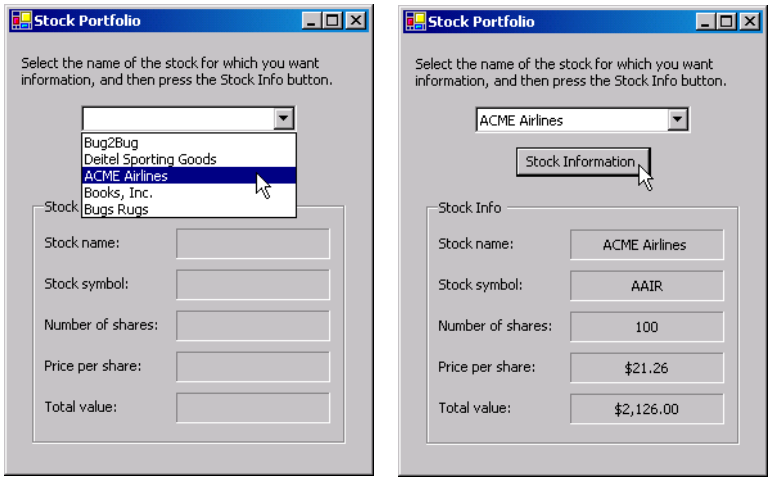


Figure 25.37 **Stock Portfolio** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial25\Exercises\StockPortfolio` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `StockPortfolio.sln` in the `StockPortfolio` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `stocks.mdb` database from your `C:\Examples\Tutorial25\Exercises\Databases` directory to your `C:\SimplyVB\StockPortfolio` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `stock.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the `Form`.*** Add two command objects to the `Form`, and set both their `Connection` properties to the database connection object. Name the command objects `objSelectStockNameCommand` and `objSelectStockInformationCommand`. The first object will be used to retrieve the name of a stock and the second item will be used to retrieve all of a stock's information, based on the name of the stock.

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectStockNameCommand` object and click the ellipses `Button` that appears to the right of the `CommandText` property in the **Properties** window. In the **Query Builder**, select **stockName** from the **stocks** table and click **OK**. Select the `objSelectStockInformationCommand` and open the **Query Builder** as you did for `objSelectStockNameCommand`. This time, select the **stockSymbol**, **shares** and **price** items from the **stocks** table. Then, select the **stockName** item and provide it with the **=?** criteria value. Finally, uncheck the **stockName** item from the **stocks** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Add a `Load` event handler for the `Form`. Add code to this event handler that opens a connection to the database. Use the `objSelectStockNameCommand` to retrieve the `StockNames`, and add them to the `ComboBox`.

h) ***Adding a `Click` event handler for the `btnStockInformation` `Button`.*** Add a `Click` event handler for the **Stock Information** `Button`. Add code to the event handler that passes the `SelectedItem` to the `StockData` method as a `String`. Then close the connection.

i) ***Defining the `StockData` method.*** Create a `StockData` method that takes a `String` representing the name of the stock as an argument. Connect to the database, and retrieve the information for the stock passed as an argument (using `objSelectStockInformationCommand`). Display the information in the corresponding `Labels` and close the connection to the database. Call the `ComputeTotalValueString` method, which you define in the next step, to calculate the total value.

j) ***Defining the `ComputeTotalValueString` method.*** Create the `ComputeTotalValueString` method to compute the total value by multiplying the number of shares by the price per share.

k) ***Running the application.*** Select **Debug > Start** to run your application. Select a company from the `ComboBox` and click the **Stock Information** `Button`. Verify that the information displayed in the **Stock Info** `GroupBox` is correct, based on the information stored in `stock.mdb`. Repeat this process for the other companies.

l) ***Closing the application.*** Close your running application by clicking its close box.

m) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 25.11 Solution
' StockPortfolio.vb

Imports System.Data.OleDb

Public Class FrmStockPortfolio
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' called when Form is loaded
   Private Sub FrmStockPortfolio_Load(ByVal sender _
      As System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      objOleDbConnection.Open() ' open connection

      ' create data reader to read from database
      Dim objReader As OleDbDataReader

      objReader = objSelectStockNameCommand.ExecuteReader()

      Do While objReader.Read

         ' add all stock names in database to the ComboBox
         cboStockNames.Items.Add(objReader("stockName"))
      Loop

      objOleDbConnection.Close() ' close database connection
   End Sub ' FrmStockPortfolio_Load

   ' handles click event for btnStockInformation Button
   Private Sub btnStockInformation_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnStockInformation.Click

      ' String representing stock selected
      Dim strSelection As String = _
         Convert.ToString(cboStockNames.SelectedItem)

      ' display stock information
      StockData(strSelection)
```

```
    End Sub ' btnStockInformation_Click

    ' retrieve stock information from database
    ' and set corresponding output Labels with data
    Private Sub StockData(ByVal strStock As String)

       objOleDbConnection.Open() ' open connection

       ' specify which stock's information to retrieve
       objSelectStockInformationCommand.Parameters( _
          "stockName").Value = strStock

       ' create data reader to read from database
       Dim objReader As OleDbDataReader

       objReader = objSelectStockInformationCommand.ExecuteReader()
       objReader.Read() ' read from the database

       ' set all output Labels
       ' with coresponding stock information
       lblNameOutput.Text = strStock
       lblSymbolOutput.Text = _
          Convert.ToString(objReader("stockSymbol"))

       lblShareNumberOutput.Text = _
          Convert.ToString(objReader("shares"))

       Dim decSharePrice As Decimal = _
          Convert.ToDecimal(objReader("price"))

       lblSharePriceOutput.Text = _
          String.Format("{0:C}", decSharePrice)

       objReader.Close() ' close OleDbDataReader

       lblTotalOutput.Text = ComputeTotalValueString( _
          lblShareNumberOutput.Text, Convert.ToString(decSharePrice))

       objOleDbConnection.Close() ' close database connection
    End Sub ' StockData

    ' helper procedure to compute total value of stock
    ' and return it as a String
    Private Function ComputeTotalValueString(ByVal strShareNumber _
       As String, ByVal strSharePrice As String) As String

       Dim intShareNumber As Integer = _
          Convert.ToInt32(strShareNumber)
       Dim decSharePrice As Decimal = _
          Convert.ToDecimal(strSharePrice)

       ' return the total value formatted as a currency
       Return String.Format("{0:C}", _
          (intShareNumber * decSharePrice))

    End Function ' ComputeTotalValueString

End Class ' FrmStockPortfolio
```

25.12 ***(Restaurant Bill Calculator Application)*** A restaurant wants you to develop an application that calculates a table's bill (Fig. 25.38). The application should display all the menu items from the restaurant's database in four `ComboBoxes`. Each `ComboBox` should contain a category of food offered by the restaurant (**Beverage**, **Appetizer**, **Main course** and **Dessert**). The user can choose from one of these `ComboBoxes` to add an item to a table's bill. When the table is finished ordering, the user can click the **Calculate Bill** `Button` to display the **Subtotal:**, **Tax:** and **Total:** for the table.

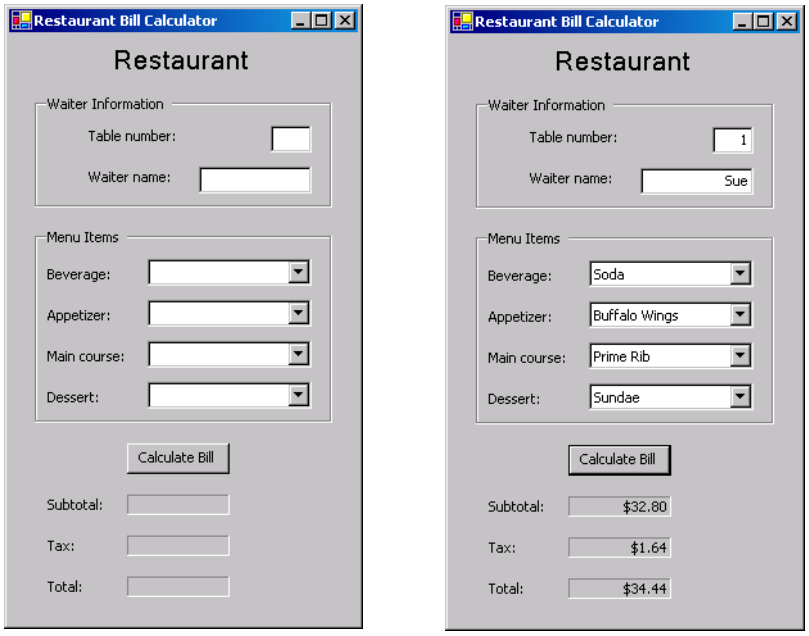


Figure 25.38 **Restaurant Bill Calculator** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial25\Exercises\RestaurantBillCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `RestaurantBillCalculator.sln` in the `RestaurantBillCalculator` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `menu.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyVB\RestaurantBillCalculator` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `menu.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the `Form`.*** Add two command objects to the `Form`, and set both their `Connection` properties to the database connection object. Name the command objects `objSelectNameCommand` and `objSelectPriceCommand`. The first object will be used to retrieve the name of a menu item, based on category (for example, appetizer). The second command object will be used to retrieve a menu item's price, based on the item's name.

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectNameCommand` object and open the **Query Builder**. Add the **menu** table and select the **name** and **category** items. Provide the **category** item with the **=?** criteria value, then deselect **category** in the **menu** table. Click **OK**. Select the `objSelectPriceCommand` and open the **Query Builder**. This time, select the items marked **price** and **name** from the **menu** table. Provide the **name** item with the **=?** criteria value. Finally, uncheck the **name** item in the **menu** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Create the `Load` event handler for the `Form`. Add code to the event handler that opens a connection to the database. Call the `LoadCategory` method four times, each time passing a different category and `ComboBox` as arguments. Close the connection to the database.

h) ***Coding the `LoadCategory` method.*** Create a method `LoadCategory` that takes a `String` representing the `Category` to load and the name of the `ComboBox` to add items to as arguments. Because the `Form`'s `Load` event handler is calling this method before it closes the connection to the database, the connection should still be open. Create a data reader to read all the items from the database for the specified `Category`, using `objSelectNameCommand`. Close the reader before exiting the method, so that a new reader can be created when the method is invoked again.

i) ***Adding `SelectedIndexChanged` event handler for the `ComboBoxes`.*** Add a `SelectedIndexChanged` event handler for all the `ComboBoxes`. Add code to the event handler that adds the `String` representation of the `SelectedItem` to the `ArrayList`.

j) ***Adding a `Click` event handler for the `btnCalculateBill` `Button`.*** Add a `Click` event handler for the **Calculate Bill** `Button`. Add code to the event handler that ensures that a table number and waiter name have been entered. If one of these fields is empty, display a `MessageBox` informing the user that both fields must contain information. The event handler should then call the `CalculateSubtotal` method to calculate the subtotal of the bill. Display the subtotal, tax and total of the bill in the appropriate `Labels`.

k) ***Coding the `CalculateSubtotal` method.*** The `CalculateSubtotal` method should open a connection to the database and retrieve the `Price` field for all the menu items in the `m_objBillItems` `ArrayList` (using `objSelectPriceCommand`). This method should then calculate the total price of all the items in the `ArrayList` and return this value as a `Decimal`. Remember to close the connection to the database.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter a table number and waiter name, and select different menu items from the `ComboBoxes`. Click the **Calculate Bill** `Button` and verify that the subtotal, tax and total values are correct. Select more items from the `ComboBoxes` and again click the **Calculate** `Button`. Verify that the price of the new items has been added to the bill.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 25.12 Solution
' RestaurantBillCalculator.vb

Imports System.Data.OleDb

Public Class FrmRestaurantBillCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' hold all items on running bill
   Dim m_objBillItems As ArrayList = New ArrayList()

   ' invoked when application is loaded
   Private Sub FrmRestaurantBillCalculator_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      objOleDbConnection.Open() ' open connection to the database

      ' load all ComboBoxes with appropriate items
      LoadCategory("Beverage", cboBeverage)
      LoadCategory("Appetizer", cboAppetizer)
      LoadCategory("Main Course", cboMainCourse)
      LoadCategory("Dessert", cboDessert)

      objOleDbConnection.Close() ' close connection to the database
   End Sub ' FrmRestaurantBillCalculator_Load
```

```

    ' loads the specified category of menu items in
    ' their corresponding ComboBox
    Private Sub LoadCategory(ByVal strCategory As String, _
       ByVal cboCategory As ComboBox)

       ' specify category parameter
       objSelectNameCommand.Parameters( _
          "category").Value = strCategory

       ' declare a reader for the database
       Dim objMenuReader As OleDbDataReader

       objMenuReader = _
          objSelectNameCommand.ExecuteReader()

       Do While objMenuReader.Read()

          ' retrieve names of items in the specified category
          ' from database, then add to specified ComboBox
          cboCategory.Items.Add(objMenuReader("Name"))
       Loop

       objMenuReader.Close() ' close the reader
    End Sub ' LoadCategory

    ' handles SelectedIndexChanged event for cboBeverage ComboBox
    Private Sub cboBeverage_SelectedIndexChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles cboBeverage.SelectedIndexChanged

       ' add selected Beverage to m_objBillItems ArrayList
       m_objBillItems.Add(cboBeverage.SelectedItem)
    End Sub ' cboBeverage_SelectedIndexChanged

    ' handles SelectedIndexChanged event for cboAppetizer ComboBox
    Private Sub cboAppetizer_SelectedIndexChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles cboAppetizer.SelectedIndexChanged

       ' add selected Appetizer to m_objBillItems ArrayList
       m_objBillItems.Add(cboAppetizer.SelectedItem)
    End Sub ' cboAppetizer_SelectedIndexChanged

    ' handles SelectedIndexChanged event for cboMainCourse ComboBox
    Private Sub cboMainCourse_SelectedIndexChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles cboMainCourse.SelectedIndexChanged

       ' add selected Main Course to m_objBillItems ArrayList
       m_objBillItems.Add(cboMainCourse.SelectedItem)
    End Sub ' cboMainCourse_SelectedIndexChanged

    ' handles SelectedIndexChanged event for cboDessert ComboBox
    Private Sub cboDessert_SelectedIndexChanged(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles cboDessert.SelectedIndexChanged

       ' add selected Dessert to m_objBillItems ArrayList
       m_objBillItems.Add(cboDessert.SelectedItem)
    End Sub ' cboDessert_SelectedIndexChanged
```

```

    ' handles click event for btnCalculateBill Button
    Private Sub btnCalculateBill_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
       Handles btnCalculateBill.Click

       ' must enter waiter name and table number
       If txtWaiterName.Text = "" OrElse _
          txtTableNumber.Text = "" Then

          MessageBox.Show( _
             "Table Number and Waiter Name must be entered", _
             "Empty Field", MessageBoxButtons.OK, _
             MessageBoxIcon.Exclamation)

       Else ' calculate the bill

          ' calculate the Subtotal
          Dim decSubtotal As Decimal = CalculateSubtotal()

          ' display the Subtotal in Label
          lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

          ' calculate tax and display in Label
          Dim decTax As Decimal = Convert.ToDecimal( _
             decSubtotal * 0.05)

          lblTaxResult.Text = String.Format("{0:C}", decTax)

          ' calculate total and display in Label
          lblTotalResult.Text = _
             String.Format("{0:C}", (decSubtotal + decTax))

       End If

    End Sub ' btnCalculateBill_Click

    ' calculate the subtotal of the bill
    Private Function CalculateSubtotal() As Decimal

       ' open connection to the database
       objOleDbConnection.Open()

       ' declare a reader for the database
       Dim objMenuReader As OleDbDataReader

       Dim intCounter As Integer
       Dim decSubtotal As Decimal = 0

       For intCounter = 0 To (m_objBillItems.Count - 1)

          ' specify name of item to retrieve
          objSelectPriceCommand.Parameters("name").Value = _
             Convert.ToString(m_objBillItems(intCounter))

          objMenuReader = _
             objSelectPriceCommand.ExecuteReader()

          objMenuReader.Read() ' read from the database

          ' retrieve price of items with specified name
```

```
            ' and add price to dblSubtotal
            decSubtotal += Convert.ToDecimal(objMenuReader("Price"))

            objMenuReader.Close() ' close the reader
        Next

        ' close connection to the database
        objOleDbConnection.Close()

        Return decSubtotal
    End Function ' CalculateSubtotal

End Class ' FrmRestaurantBillCalculator
```

25.13 ***(Airline Reservation Application)*** An airline company wants you to develop an application that displays flight information (Fig. 25.39). The database contains two tables, one containing information about the flights, the other containing passenger information. The user should be able to choose a flight number from a `ComboBox`. When the **View Flight Information** `Button` is clicked, the application should display the date of the flight, the flight's departure and arrival cities and the names of the passengers schedule to take the flight.

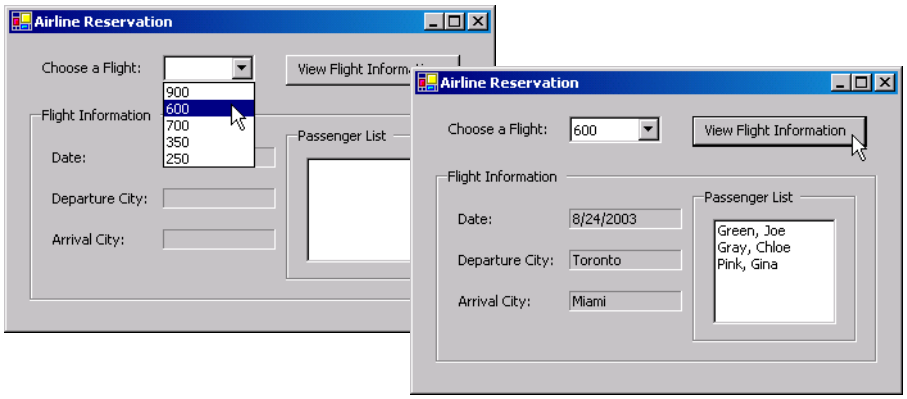


Figure 25.39 Airline Reservation application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial25\Exercises\AirlineReservation` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AirlineReservation.sln` in the `AirlineReservation` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `reservations.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyVB\AirlineReservation` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `reservations.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the Form.*** Add three command objects to the `Form`, and set all their `Connection` properties to the database connection object. Name the command objects `objSelectFlightNumberCommand` (used to retrieve flight numbers), `objSelectFlightInformationCommand` (used to retrieve information about a flight based on the flight's number) and `objSelectPassengerInformationCommand` (used to retrieve information about a flight's passengers based on the flight's number).

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectFlightNumberCommand` object and open the **Query Builder**. Select **FlightNumber** from the **flights** table and click **OK**. Select the `objSelectFlightInformationCommand` and open the **Query Builder**. This time, select the **Date**, **DepartureCity** and

ArrivalCity items from the **flights** table. This action causes all items from the table to be returned. Then, select the **FlightNumber** item and provide it with the **=?** criteria value. Finally, uncheck the **FlightNumber** item from the **flights** table. Click **OK** to dismiss the **Query Builder**. Select the `objSelectPassengerInformationCommand` and open the **Query Builder**. Select the **LastName** and **FirstName** items from the **reservations** table. Then, select the **FlightNumber** item and provide it with the **=?** criteria value. Finally, uncheck the **FlightNumber** item from the **reservations** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Create a `Load` event handler for the `Form` that opens a connection to the database. Retrieve all the `FlightNumbers` from the `Flights` table in the `reservations.mdb` database (using `objSelectFlightNumberCommand`), and add those `FlightNumbers` to the `ComboBox`.

h) ***Adding a `Click` event handler for the `btnViewFlightInformation` Button.*** Add a `Click` event handler for the **View Flight Information** `Button`. Add code to the event handler to pass the `SelectedItem` to the `DisplayFlightInformation` method.

i) ***Defining the `DisplayFlightInformation` method.*** The `DisplayFlightInformation` method should take as an argument a `String` representing the flight number chosen. You will need to define two readers in this method, to read from the two tables in the database. Once you open the connection to the database, create a reader that reads the specified flight information from the `flights` table (using `objSelectFlightInformationCommand`). Display the flight information in the correct `Label`. Close this reader, and create a second reader that reads passenger information from the `reservations` table (using `objPassengerInformationCommand`). Retrieve from the table all the passengers scheduled to take the specified flight. Clear any old items from the `ListBox`, and display passengers' names in the `ListBox`.

j) ***Running the application.*** Select **Debug > Start** to run your application. Select a flight and click the **View Flight Information** `Button`. Verify that the flight information is correct. Repeat this process for the other flights.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 25.13 Solution
' AirlineReservation.vb

Imports System.Data.OleDb

Public Class FrmAirlineReservation
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' invoked when Form is loaded
   Private Sub FrmAirlineReservation_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      objOleDbConnection.Open() ' open database connection

      ' create reader to read information from database
      Dim objReader As OleDbDataReader

      objReader = objSelectFlightNumberCommand.ExecuteReader()

      Do While objReader.Read()

         ' retrieve flight number from database
         ' and add it to cboChooseAFlight
         cboChooseAFlight.Items.Add( _
```

```
            Convert.ToString(objReader("FlightNumber")))

         Loop

         objOleDbConnection.Close() ' close database connection
      End Sub ' FrmAirlineReservation_Load

      ' handles click event for btnViewFlightInformation Button
      Private Sub btnViewFlightInformation_Click(ByVal sender _
         As System.Object, ByVal e As System.EventArgs) _
         Handles btnViewFlightInformation.Click

         ' retrieve selected index
         Dim strFlightNumber As String = _
            Convert.ToString(cboChooseAFlight.SelectedItem)

         ' display new flight information
         DisplayFlightInformation(strFlightNumber)

      End Sub ' btnFlightView_Click

      ' display flight information in GroupBox's Labels
      Private Sub DisplayFlightInformation( _
         ByVal strFlightNumber As String)

         objOleDbConnection.Open() ' open database connection

         objSelectFlightInformationCommand.Parameters( _
            "FlightNumber").Value = strFlightNumber

         ' create data reader to read information from database
         Dim objFlightReader As OleDbDataReader

         objFlightReader = _
            objSelectFlightInformationCommand.ExecuteReader()

         Do While objFlightReader.Read()

            ' retrieve date, departure city, arrival city
            lblDateOutput.Text = _
               Convert.ToString(objFlightReader("Date"))
            lblDepartureOutput.Text = _
               Convert.ToString(objFlightReader("DepartureCity"))
            lblArrivalOutput.Text = _
               Convert.ToString(objFlightReader("ArrivalCity"))
         Loop

         objFlightReader.Close() ' close the data reader

         ' specify flight number to retrieve passenger information
         objSelectPassengerInformationCommand.Parameters( _
            "flightNumber").Value = strFlightNumber

         ' create data reader to read information from database
         Dim objPassengerReader As OleDbDataReader

         objPassengerReader = _
            objSelectPassengerInformationCommand.ExecuteReader()

         Dim strFirstName As String
         Dim strLastName As String
```

```
89
90          lstDisplay.Items.Clear() ' clear previous passenger list
91
92          Do While objPassengerReader.Read()
93
94             strLastName = _
95                Convert.ToString(objPassengerReader("LastName"))
96             strFirstName = _
97                Convert.ToString(objPassengerReader("FirstName"))
98
99             lstDisplay.Items.Add(strLastName & ", " & strFirstName)
100         Loop
101
102         objPassengerReader.Close() ' close the data reader
103
104         objOleDbConnection.Close() ' close database connection
105
106      End Sub ' DisplayFlighInformation
107
108   End Class ' FrmAirlineReservation
```

What does this code do?

25.14 What does the following code do?

```
1   objSelectAgeData.Parameters("Name").Value = "Bob"
2
3   objOleDbConnection.Open()
4
5   Dim objReader As OleDbDataReader = _
6      objSelectAgeData.ExecuteReader
7
8   objReader.Read()
9
10  m_intAge = Convert.ToInteger(objReader("Age"))
11
12  objReader.Close()
13  objOleDbConnection.Close()
```

Answer: This code segment connects to a database and retrieves the `Age` data for the person with the name `Bob`. This data is then stored in `m_intAge`. The code then closes both the data reader and the connection to the database.

What's wrong with this code?

25.15 Find the error(s) in the following code. This method should modify the `Age` field of `strUserName`.

```
1   objUpdateAge("Age").Value = _
2      intAge
3
4   objUpdateBalance.Parameters("Original_NAME").Value = _
5      strUserName
6
7   objUpdateAge.ExecuteNonQuery()
8
9   objOleDbConnection.Close()
```

Answer: Line 1 does not properly set the `Age` parameter—the `Parameters` property of `objUpdateAge` must be used. Also, no connection is made to the database before the `UPDATE` is attempted.

```
objUpdateAge.Parameters("Age").Value = _
   intAge

objUpdateBalance.Parameters("Original_NAME").Value = _
   strUserName

objOleDbConnection.Open()

objUpdateAge.ExecuteNonQuery()

objOleDbConnection.Close()
```

Programming Challenge

25.16 ***(Enhanced Restaurant Bill Calculator)*** Modify the application you developed in Exercise 25.12 to keep track of multiple table bills at the same time. The user should be able to calculate a bill for a table and save that table's subtotal and waiter's name. The user should also be able to retrieve that information at a later time. [*Hint*: This database contains two tables, one for the menu items, as before, and another for all the tables in the restaurant.] Sample outputs are shown in Fig. 25.40.

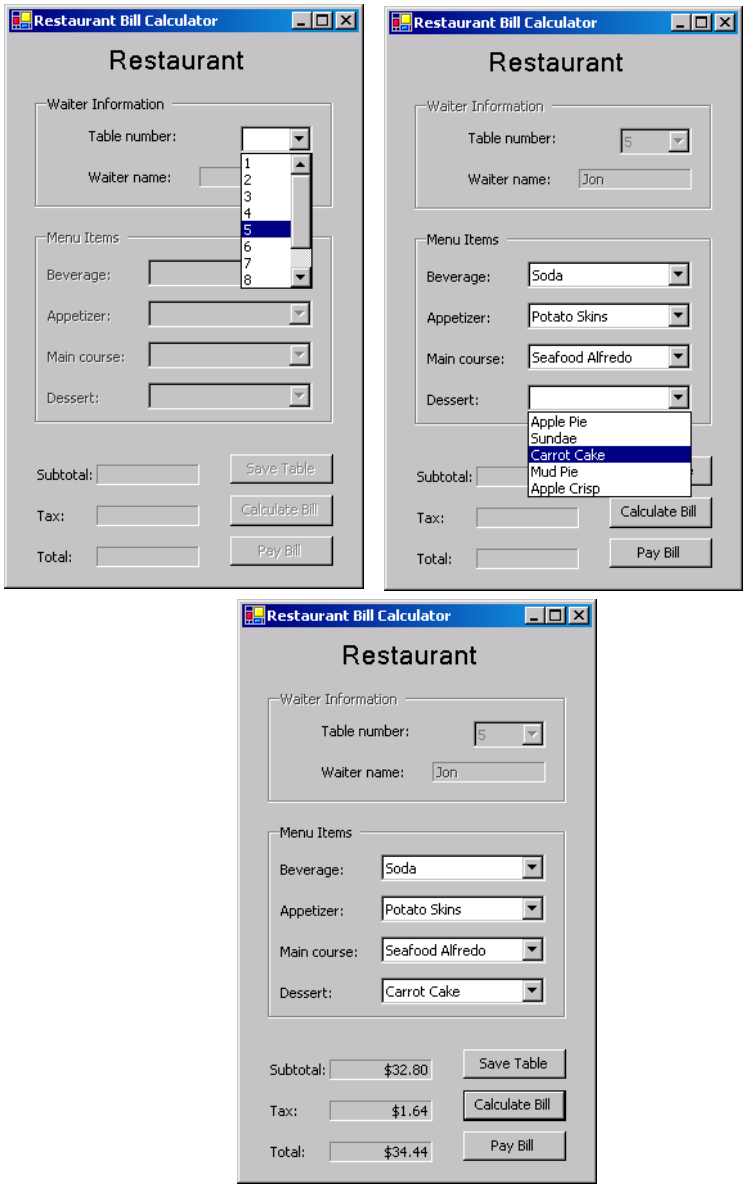


Figure 25.40 Enhanced **Restaurant Bill Calculator** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial25\Exercises\RestaurantBillCalculatorEnhanced` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `RestaurantBillCalculator.sln` in the `RestaurantBillCalculatorEnhanced` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `menu2.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyVB\RestaurantBillCalculatorEnhanced` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `menu2.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the Form.*** Add five command objects to the `Form`, and set their `Connection` properties to the database connection object. Name the command objects `objSelectNameCommand`, `objSelectPriceCommand`, `objSelectTableNumberCommand`, `objSelectTableInfoCommand` and `objUpdateSubtotalCommand`.

f) ***Setting the command objects' `CommandText` properties.*** Set the `CommandText` properties of `objSelectNameCommand` and `objSelectPriceCommand` as you did in Exercise 25.12. Set `objSelectTableNumberCommand` to retrieve table numbers from the **tables** table. Set `objSelectTableInfoCommand` to retrieve the name of the waiter and the subtotal of a table, based on that table's number. Set `objUpdateSubtotalCommand` to modify the subtotal for a table, also based on that table's number. [*Note*: For the last command object, you will need to change the type of query in the **Query Builder**, as you did earlier in this tutorial.]

g) ***Copying your existing code.*** Copy the code for the application you created in Exercise 25.12 into the template application for this exercise. Place this code before method `ResetForm`. Disregard any syntax errors that may appear in the **Task List** at this point.

h) ***Adding an instance variable.*** Add an instance variable (after the declaration of `m_objBillItems`) called `m_decSubtotal`, that will hold the subtotal for each table when it is loaded in the application.

i) ***Modifying methods `btnCalculateBill_Click` and `CalculateSubtotal`.*** Remove the portion of the `btnCalculateBill_Click` event handler that checked for a table number and waiter name—this information will be displayed shortly. Modify the `CalculateSubtotal` method to update the table's subtotal based on the table's previous subtotal and any new items selected.

j) ***Creating a method.*** Create method `LoadTables` that reads the table numbers from the database and adds them to the **Table number**: `ComboBox`. This method should be called in `FrmRestaurantBillCalculator_Load` directly after the connection to the database is opened.

k) ***Adding an event handler.*** Add an event handler for the **Table number**: `ComboBox`. When a table is selected from the `ComboBox`, that table's data should be loaded from the database.

l) ***Creating an event handler for the Save Table `Button`.*** Create an event handler for the **Save Table** `Button`. This event handler should calculate the subtotal for the selected table. The event handler should then call method `UpdateTable`, passing the subtotal and table number as arguments. Finally, call the `ResetForm` method to reset the data displayed in the GUI.

m) ***Creating an event handler for the Pay Bill `Button`.*** Create an event handler for the **Pay Bill** `Button`. This event handler should retrieve the current table number then call method `UpdateTable`, passing a subtotal of `0` (for new customers) and table number as arguments. Finally, call the `ResetForm` method to reset the data displayed in the GUI.

n) ***Creating method `UpdateTable`.*** Create a method `UpdateTable` that takes the subtotal and table number as arguments. This method should save the table data in the database.

o) ***Running the application.*** Select **Debug > Start** to run your application. Select a table number and various menu items from the `ComboBoxes`. Click the **Calculate Bill** `Button` and verify that the subtotal, tax and total values are correct. Select more items from the `ComboBoxes` and again click the **Calculate Bill** `Button`. Verify that the price of the new items has been added to the bill. Click the **Save Table** `Button`. Select a different table and various menu items. Click the **Calculate Bill** `Button` and verify that the price of the new items has been added to the bill. Click the **Save Table** `Button`. Select the first table and verify that the subtotal is the same as it was when the table was saved. Select various menu items and Click the **Calculate Bill** `Button`. Verify that the subtotal, tax and total values are correct (and now include prices of the new menu items). Click the **Pay Bill** `Button` and verify that the subtotal is not reset to **$0.00**.

p) ***Closing the application.*** Close your running application by clicking its close box.

q) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 25.15 Solution
' RestaurantBillCalculator.vb

Imports System.Data.OleDb

Public Class FrmRestaurantBillCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' hold all items on running bill
   Dim m_objBillItems As ArrayList = New ArrayList

   Dim m_decSubtotal As Decimal = 0 ' current table subtotal

   ' invoked when application is loaded
   Private Sub FrmRestaurantBillCalculator_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      objOleDbConnection.Open() ' open connection to the database

      ' load cboTables ComboBox with table numbers
      LoadTables()

      ' load all ComboBoxes with appropriate items
      LoadCategory("Beverage", cboBeverage)
      LoadCategory("Appetizer", cboAppetizer)
      LoadCategory("Main Course", cboMainCourse)
      LoadCategory("Dessert", cboDessert)

      objOleDbConnection.Close() ' close connection to the database
   End Sub ' FrmRestaurantBillCalculator_Load

   ' loads the specified category of menu items in
   ' their corresponding ComboBox
   Private Sub LoadCategory(ByVal strCategory As String, _
      ByVal cboCategory As ComboBox)

      ' specify category parameter
      objSelectNameCommand.Parameters( _
         "category").Value = strCategory

      ' declare a reader for the database
      Dim objMenuReader As OleDbDataReader

      objMenuReader = _
         objSelectNameCommand.ExecuteReader()

      Do While objMenuReader.Read()

         ' retrieve names of items in the specified category
         ' from database, then add to specified ComboBox
         cboCategory.Items.Add(objMenuReader("Name"))
      Loop

      objMenuReader.Close()
   End Sub ' LoadCategory

   ' handles SelectedIndexChanged event for cboBeverage ComboBox
```

```
   Private Sub cboBeverage_SelectedIndexChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles cboBeverage.SelectedIndexChanged

      ' add selected Beverage to m_objBillItems ArrayList
      m_objBillItems.Add(cboBeverage.SelectedItem)
   End Sub ' cboBeverage_SelectedIndexChanged

   ' handles SelectedIndexChanged event for cboAppetizer ComboBox
   Private Sub cboAppetizer_SelectedIndexChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles cboAppetizer.SelectedIndexChanged

      ' add selected Appetizer to m_objBillItems ArrayList
      m_objBillItems.Add(cboAppetizer.SelectedItem)
   End Sub ' cboAppetizer_SelectedIndexChanged

   ' handles SelectedIndexChanged event for cboMainCourse ComboBox
   Private Sub cboMainCourse_SelectedIndexChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles cboMainCourse.SelectedIndexChanged

      ' add selected Main Course to m_objBillItems ArrayList
      m_objBillItems.Add(cboMainCourse.SelectedItem)
   End Sub ' cboMainCourse_SelectedIndexChanged

   ' handles SelectedIndexChanged event for cboDessert ComboBox
   Private Sub cboDessert_SelectedIndexChanged(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles cboDessert.SelectedIndexChanged

      ' add selected Dessert to objBillItems ArrayList
      m_objBillItems.Add(cboDessert.SelectedItem)
   End Sub ' cboDessert_SelectedIndexChanged

   ' handles click event for btnCalculateBill Button
   Private Sub btnCalculateBill_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculateBill.Click

      ' calculate the Subtotal
      Dim decSubtotal As Decimal = CalculateSubtotal()

      ' display the Subtotal in Label
      lblSubtotalResult.Text = String.Format("{0:C}", decSubtotal)

      ' calculate tax and display in Label
      Dim decTax As Decimal = Convert.ToDecimal(decSubtotal * 0.05)

      lblTaxResult.Text = String.Format("{0:C}", decTax)

      ' calculate total and display in Label
      lblTotalResult.Text = _
         String.Format("{0:C}", (decSubtotal + decTax))

   End Sub ' btnCalculateBill_Click

   ' calculate the subtotal of the bill
   Private Function CalculateSubtotal() As Decimal

      ' open connection to the database
```

```
      objOleDbConnection.Open()

      ' declare a reader for the database
      Dim objMenuReader As OleDbDataReader

      Dim intCounter As Integer
      Dim decSubtotal As Decimal = m_decSubtotal

      For intCounter = 0 To (m_objBillItems.Count - 1)

         ' specify name of item to retrieve
         objSelectPriceCommand.Parameters("name").Value = _
            Convert.ToString(m_objBillItems(intCounter))

         objMenuReader = _
            objSelectPriceCommand.ExecuteReader()

         ' read from the database
         Do While objMenuReader.Read()

            ' retrieve price of items with specified name
            ' and add price to decSubtotal
            decSubtotal += Convert.ToDecimal(objMenuReader("Price"))
         Loop

         objMenuReader.Close() ' close the reader
      Next

      ' close database connection
      objOleDbConnection.Close()

      Return decSubtotal
   End Function ' CalculateSubtotal

   ' reset all controls on the Form
   Private Sub ResetForm()

      ' clear all Labels
      lblSubtotalResult.Text = ""
      lblTaxResult.Text = ""
      lblTotalResult.Text = ""
      lblWaiterNameOutput.Text = ""

      ' set Selected index to -1 so
      ' no item is selected in ComboBoxes
      cboBeverage.SelectedIndex = -1
      cboAppetizer.SelectedIndex = -1
      cboMainCourse.SelectedIndex = -1
      cboDessert.SelectedIndex = -1
      cboTables.SelectedIndex = -1

      fraMenuItems.Enabled = False ' disable Menu Items GroupBox

      ' enable Waiter Information GroupBox
      fraWaiterInformation.Enabled = True

      ' disable all Buttons
      btnSaveTable.Enabled = False
      btnCalculateBill.Enabled = False
      btnPayBill.Enabled = False

```

```
      m_decSubtotal = 0 ' clear subtotal

      ' reinitialize m_objBillItems ArrayList
      m_objBillItems = New ArrayList
   End Sub ' ResetForm

   ' load tables from database
   Private Sub LoadTables()

      ' declare a reader for the database
      Dim objTableReader As OleDbDataReader

      objTableReader = _
         objSelectTableNumberCommand.ExecuteReader()

      Do While objTableReader.Read()

         ' retrieve names of items in the specified category
         ' from database, then add to specified ComboBox
         cboTables.Items.Add(objTableReader("TableNumber"))

      Loop

      objTableReader.Close()
   End Sub ' LoadTables

   ' handles SelectedIndexChanged event for cboTables ComboBox
   Private Sub cboTables_SelectedIndexChanged(ByVal sender _
      As System.Object, ByVal e As System.EventArgs) _
      Handles cboTables.SelectedIndexChanged

      fraMenuItems.Enabled = True ' enable all Menu Items GroupBox

      ' enable all Buttons
      btnSaveTable.Enabled = True
      btnCalculateBill.Enabled = True
      btnPayBill.Enabled = True

      ' clear Tax and Total output Labels
      lblTaxResult.Text = ""
      lblTotalResult.Text = ""

      objOleDbConnection.Open() ' open connection to the database

      ' SELECT statement used to retrieve item names
      objSelectTableInfoCommand.Parameters( _
         "tableNumber").Value = _
         Convert.ToInt32(cboTables.SelectedItem)

      ' declare a reader for the database
      Dim objTableReader As OleDbDataReader

      objTableReader = _
         objSelectTableInfoCommand.ExecuteReader()

      Do While objTableReader.Read()

         ' retrieve waiter name and subtotal from database
         lblWaiterNameOutput.Text = _
            Convert.ToString(objTableReader("WaiterName"))

```

```
         m_decSubtotal = _
            Convert.ToDecimal(objTableReader("Subtotal"))
         lblSubtotalResult.Text = _
            String.Format("{0:C}", m_decSubtotal)
      Loop

      objTableReader.Close() ' close the reader

      ' close connection to the database
      objOleDbConnection.Close()

      ' disable Waiter Information GroupBox
      fraWaiterInformation.Enabled = False
   End Sub ' cboTables_SelectedIndexChanged

   ' handles click event for btnSaveTable Button
   Private Sub btnSaveTable_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnSaveTable.Click

      Dim decSubtotal As Decimal = CalculateSubtotal()
      Dim intTableNumber As Integer = _
         Convert.ToInt32(cboTables.SelectedItem)

      ' update table
      UpdateTable(decSubtotal, intTableNumber)

      ResetForm() ' reset all controls
   End Sub ' btnSaveTable_Click

   ' handles click event for btnSaveTable Button
   Private Sub btnPayBill_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPayBill.Click

      Dim intTableNumber As Integer = _
         Convert.ToInt32(cboTables.SelectedItem)

      UpdateTable(0, intTableNumber) ' update table
      ResetForm()  ' reset all controls
   End Sub ' btnPayBill_Click

   ' update table information in database
   Private Sub UpdateTable(ByVal decSubtotal As Decimal, _
      ByVal intTableNumber As Integer)

      objOleDbConnection.Open() ' open database connection

      ' specify parameters for UPDATE statement
      objUpdateSubtotalCommand.Parameters( _
         "subtotal").Value = decSubtotal
      objUpdateSubtotalCommand.Parameters( _
         "Original_TableNumber").Value = _
         Convert.ToString(intTableNumber)

      ' execute update statement
      objUpdateSubtotalCommand.ExecuteNonQuery()

      objOleDbConnection.Close() ' close database connection
   End Sub ' UpdateTable
```

```

End Class ' FrmRestaurantBillCalculator
```

TUTORIAL 26

CheckWriter Application

Introducing Graphics and Printing

Solutions

Instructor's Manual Exercise Solutions Tutorial 26

MULTIPLE-CHOICE QUESTIONS

26.1 The RGB value (0, 0, 255) represents _____.

a) `Color.Red` b) `Color.Green`
c) `Color.Blue` d) `Color.Yellow`

26.2 The _____ property of the `PrintPreviewDialog` object makes text appear smoother.

a) `AntiAlias` b) `UseAntiAlias`
c) `Alias` d) `UseAlias`

26.3 Use a _____ object to allow the users to preview a document before it is printed.

a) `PrintPreviewDialog` b) `PrintDocument`
c) `Print` d) `PrintPreviewControl`

26.4 The _____ event handler specifies what will be printed.

a) `OnPaint` b) `Print`
c) `Document` d) `PrintPage`

26.5 To display the preview dialog of the _____ object, call method `ShowDialog`.

a) `PrintPreviewDialog` b) `PrintDocument`
c) `PrintDialog` d) Both a and b.

26.6 Set the _____ property to `False` to indicate that there are no more pages to print.

a) `Document` b) `HasMorePages`
c) `TerminatePrint` d) Both a and b.

26.7 The `Print` method sends a _____ object to the printer for printing.

a) `Graphics` b) `PrintDocument`
c) `PrintPreviewDialog` d) `Brush`

26.8 Keyword _____ references the current object.

a) `This` b) `Class`
c) `Me` d) `Property`

26.9 Opacity is the _____ value of a color.

a) red b) transparency
c) dithering d) blue

26.10 Design units are used to specify the _____ of a `Font`.

a) `Size` b) `Name`
c) `FontFamily` d) `Style`

Answers: 26.1) c. 26.2) b. 26.3) a. 26.4) d. 26.5) a. 26.6) b. 26.7) a. 26.8) c. 26.9) b. 26.10) a.

EXERCISES

26.11 (*CheckWriter Modified to Print Background Images*) Modify the **CheckWriter** application to display and print a background for the check. The GUI should look similar to Fig. 26.31. Users can select a background image. The image should appear in the **Print preview** dialog box and also should print as a background to the check.

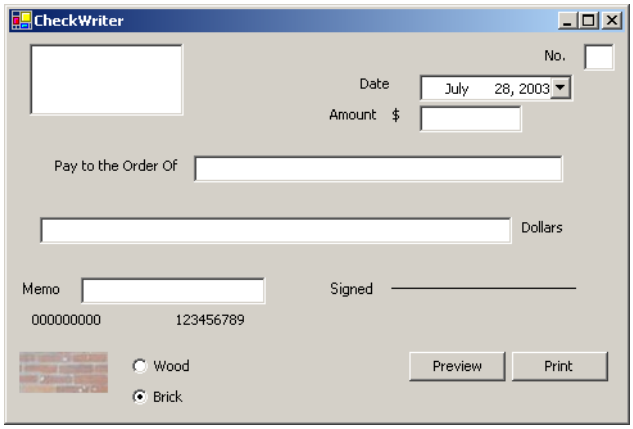


Figure 26.31 Modified **CheckWriter** GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial26\Exercises\ModifiedCheckWriter` to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CheckWriter.sln` in the `CheckWriter` directory to open the application.

c) ***Create the `CheckedChanged` event handler.*** Double click the **Wood** `RadioButton` to create its `CheckedChanged` event handler.

d) ***Defining the `CheckedChanged` event handler.*** Define the `RadioButton`'s `CheckedChanged` event handler to notify the application when users have made a background selection. If the **Wood** `RadioButton` is selected, then a preview of the wooden background should display in the `picPreview PictureBox`. Otherwise, if the **Brick** `RadioButton` is selected, then a preview of the brick background should display in the `picPreview PictureBox`.

e) ***Modifying the `objPrintDocument_PrintPage` event handler.*** Modify the `objPrintDocument_PrintPage` event handler to print the background image. [*Hint*: Use the `DrawImage` method to display the background image to print. `DrawImage` takes five arguments: The image file, the *x*-coordinate, the *y*-coordinate, the width and the height.] To print the image in the background, the `DrawImage` method must be the first method called on the `Graphics` object.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter data into the input fields and select either the **Wood** or **Brick** `RadioButton`. Verify that the appropriate image is displayed to the left of the `RadioButtons`. Click the **Preview** `Button` and verify that the check is displayed with the proper background. Close the preview and repeat this process selecting the background you had not selected before.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 26.11 Solution
' CheckWriter.vb

Imports System.Drawing.Printing

Public Class FrmCheckWriter
   Inherits System.Windows.Forms.Form

   Private m_objFont As Font ' instance variable to store font
   Private m_strPath As String

   ' Windows Form Designer generated code

```

```
    ' PrintPage event raised for each page to be printed
    Private Sub objPrintDocument_PrintPage(ByVal sender _
       As System.Object, ByVal e As PrintPageEventArgs)

       Dim sngYPosition As Single
       Dim sngXPosition As Single

       ' represent left margin of page
       Dim sngLeftMargin As Single = e.MarginBounds.Left

       ' represent top margin of page
       Dim sngTopMargin As Single = e.MarginBounds.Top

       Dim strLine As String = Nothing
       Dim objControl As Control

       ' if m_strPath has value, display
       ' specified image on Image control
       If m_strPath <> "" Then
          Dim objImage As Image
          objImage = Image.FromFile(m_strPath)

          ' print image so it is the rear-most object
          e.Graphics.DrawImage(Image.FromFile(m_strPath), _
              sngLeftMargin, sngTopMargin, Me.Width, _
              Me.Height - 60)

       End If

       ' iterate over the form, printing each control
       For Each objControl In Me.Controls

          ' we do not want to print Buttons or RadioButtons
          If objControl.GetType.Name <> "Button" AndAlso _
              objControl.GetType.Name <> "RadioButton" Then
             strLine = objControl.Text

             Select Case objControl.Name

                ' underline the date
                Case "dtpDate"
                   m_objFont = New Font("Tahoma", 8.25, _
                      FontStyle.Underline)

                   ' draw a box around amount
                Case "txtAmount"
                   e.Graphics.DrawRectangle(Pens.Black, _
                      txtAmount.Location.X + sngLeftMargin, _
                      txtAmount.Location.Y + sngTopMargin - 4, _
                      txtAmount.Width, txtAmount.Height)

                   m_objFont = objControl.Font ' default font

                Case Else
                   m_objFont = objControl.Font ' default font

             End Select

             ' set string positions relative to page margins
             sngXPosition = sngLeftMargin + _
                 objControl.Location.X
```

```

            sngYPosition = sngTopMargin + _
                objControl.Location.Y

            ' draw text in graphics object
            e.Graphics.DrawString(strLine, m_objFont, _
                Brushes.Black, sngXPosition, sngYPosition)

         End If

      Next ' control

      ' draw box around check
      e.Graphics.DrawRectangle(Pens.Black, sngLeftMargin, _
          sngTopMargin, Me.Width, Me.Height - 60)

      ' indicate that there are no more pages to print
      e.HasMorePages = False

   End Sub ' objPrintDocument_PrintPage

   ' print document
   Private Sub btnPrint_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPrint.Click

      ' create new object to assist in printing
      Dim objPrintDocument As New PrintDocument

      ' tell PrintDocument where to find PrintPage event handler
      AddHandler objPrintDocument.PrintPage, _
         AddressOf objPrintDocument_PrintPage

      ' if no printers installed, display error message
      If PrinterSettings.InstalledPrinters.Count = 0 Then
         ErrorMessage()
         Return ' exit event handler
      End If

      ' print the document
      objPrintDocument.Print()

   End Sub ' btnPrint_Click

   ' display document in print preview dialog
   Private Sub btnPreview_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPreview.Click

      ' create new object to assist in previewing
      Dim objPrintDocument As New PrintDocument

      ' tell PrintDocument where to find PrintPage event handler
      AddHandler objPrintDocument.PrintPage, _
          AddressOf objPrintDocument_PrintPage

      ' if no printers installed, display error message
      If PrinterSettings.InstalledPrinters.Count = 0 Then
         ErrorMessage()
         Return ' exit event handler
      End If
```

```
136
137         objPreview.Document = objPrintDocument ' specify document
138         objPreview.ShowDialog() ' show print preview
139
140      End Sub ' btnPreview_Click
141
142      ' display an error message to the user
143      Sub ErrorMessage()
144
145         MessageBox.Show("No printers installed. You must " & _
146            "have a printer installed to preview or print " & _
147            "the document.", "Print Error", _
148            MessageBoxButtons.OK, MessageBoxIcon.Information)
149
150      End Sub ' ErrorMessage
151
152      ' display selected image in Image control
153      Private Sub radWood_CheckedChanged(ByVal sender As _
154          System.Object, ByVal e As System.EventArgs) _
155          Handles radWood.CheckedChanged
156
157         ' if Wood RadioButton is selected
158         ' then set m_strPath to wood image
159         If radWood.Checked = True Then
160            m_strPath = "wood.jpg"
161
162            ' otherwise set m_strPath to brick image
163         Else
164            m_strPath = "bricks.jpg"
165         End If
166
167         ' set PictureBox image
168         picPreview.Image = Image.FromFile(m_strPath)
169
170      End Sub ' radWood_CheckedChanged
171
172   End Class ' FrmCheckWriter
```

26.12 (*Company Logo Designer Application*) Develop a **Company Logo** application that allows users to design a company logo (Fig. 26.32). The application should provide the user with `RadioButtons` to allow the selection of the next shape to draw. `TextBoxes` should be provided to allow the user to enter the dimensions of the shapes.

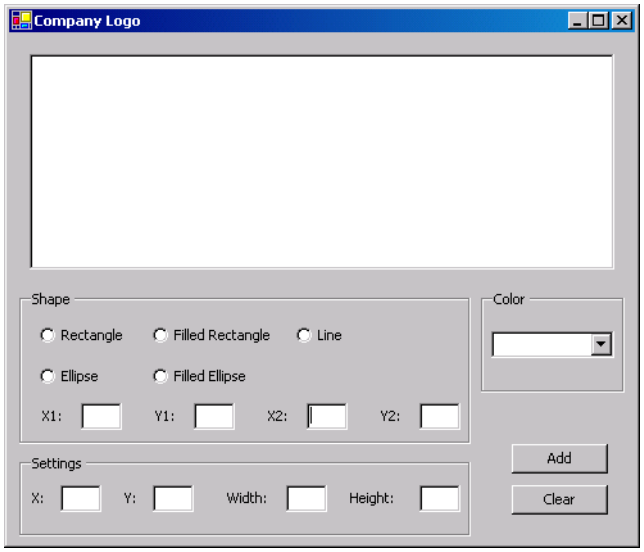


Figure 26.32 Company Logo GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial26\Exercises\CompanyLogo` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CompanyLogo.sln` in the `CompanyLogo` directory to open the application.

c) ***Defining the `Add` Button's `Click` event handler.*** Create the **Add** `Button`'s `Click` event handler. Define the event handler so that the shape that users specify is drawn on the `PictureBox`. Use the `CreateGraphics` method on the `PictureBox` to retrieve the `Graphics` object used to draw on the `PictureBox`.

d) ***Defining the `Clear` Button's `Click` event handler.*** Create the **Clear** `Button`'s `Click` event handler, and define it so that the `PictureBox` is cleared. [*Hint*: To clear the entire `PictureBox`, use the `PictureBox`'s `Invalidate` method. The `Invalidate` method is often used to refresh (update) graphics of a control. By using the `Invalidate` method without specifying a graphic to draw, the `PictureBox` clears.] Also ensure that all `TextBoxes` are cleared when the **Clear** `Button` is clicked.

e) ***Running the application.*** Select **Debug > Start** to run your application. Use the `RadioButtons` and `TextBoxes` to display at least one of each type of shape. Use different colors for the different shapes. Click the **Clear** `Button` to clear the shapes.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 26.12 Solution
' CompanyLogo.vb

Public Class FrmLogo
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' adds shapes specified by users to the PictureBox
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' create a graphics object to draw shapes
      Dim objGraphics As Graphics = picImage.CreateGraphics

      ' create brush object used for solid shapes
```

```
17         Dim objBrush As SolidBrush = New SolidBrush( _
18            Color.FromName(Convert.ToString(cboColor.SelectedItem)))
19
20         ' create pen object used for unfilled shapes
21         Dim objPen As Pen = New Pen(objBrush)
22
23         ' create rectangle
24         If radRectangle.Checked = True Then
25
26            ' determine if all values are provided
27            If (txtX1.Text <> "" AndAlso txtY1.Text <> "" AndAlso _
28               txtX2.Text <> "" AndAlso txtY2.Text <> "") Then
29
30               objgraphics.DrawRectangle(objPen, Convert.ToInt32( _
31                  txtXPosition.Text), Convert.ToInt32( _
32                  txtYPosition.Text), Convert.ToInt32(txtWidth.Text), _
33                  Convert.ToInt32(txtHeight.Text))
34            Else
35
36               MessageBox.Show("You did not provide all the setting" & _
37                  " values", "Missing setting values", _
38                  MessageBoxButtons.OK, MessageBoxIcon.Information)
39
40            End If
41
42         ElseIf radFilledRectangle.Checked = True Then
43
44            ' determine if all values are provided
45            If (txtX1.Text <> "" AndAlso txtY1.Text <> "" AndAlso _
46               txtX2.Text <> "" AndAlso txtY2.Text <> "") Then
47
48               ' create filled rectangle
49               objgraphics.FillRectangle(objBrush, Convert.ToInt32( _
50                  txtXPosition.Text), Convert.ToInt32( _
51                  txtYPosition.Text), Convert.ToInt32(txtWidth.Text), _
52                  Convert.ToInt32(txtHeight.Text))
53            Else
54
55               MessageBox.Show("You did not provide all the setting" & _
56                  " values", "Missing setting values", _
57                  MessageBoxButtons.OK, MessageBoxIcon.Information)
58
59            End If
60
61         ElseIf radEllipse.Checked = True Then
62
63            ' determine if all values are provided
64            If (txtX1.Text <> "" AndAlso txtY1.Text <> "" AndAlso _
65               txtX2.Text <> "" AndAlso txtY2.Text <> "") Then
66
67               ' draw ellipse
68               objgraphics.DrawEllipse(objPen, Convert.ToInt32( _
69                  txtXPosition.Text), Convert.ToInt32( _
70                  txtYPosition.Text), Convert.ToInt32(txtWidth.Text), _
71                  Convert.ToInt32(txtHeight.Text))
72            Else
73
74               MessageBox.Show("You did not provide all the setting" & _
75                  " values", "Missing setting values", _
76                  MessageBoxButtons.OK, MessageBoxIcon.Information)
77
```

```
         End If

      ElseIf radFilledEllipse.Checked = True Then

         ' determine if all values are provided
         If (txtX1.Text <> "" AndAlso txtY1.Text <> "" AndAlso _
            txtX2.Text <> "" AndAlso txtY2.Text <> "") Then

            ' create filled ellipse
            objgraphics.FillEllipse(objBrush, Convert.ToInt32( _
               txtXPosition.Text), Convert.ToInt32( _
               txtYPosition.Text), Convert.ToInt32(txtWidth.Text), _
               Convert.ToInt32(txtHeight.Text))
         Else

            MessageBox.Show("You did not provide all the setting" & _
               " values", "Missing setting values", _
               MessageBoxButtons.OK, MessageBoxIcon.Information)

         End If

      ElseIf radLine.Checked = True Then

         ' determine if all values are provided
         If (txtX1.Text <> "" AndAlso txtY1.Text <> "" AndAlso _
            txtX2.Text <> "" AndAlso txtY2.Text <> "") Then

         ' draw line
         objGraphics.DrawLine(objPen, Convert.ToInt32(txtX1.Text), _
            Convert.ToInt32(txtY1.Text), Convert.ToInt32( _
            txtX2.Text), Convert.ToInt32(txtY2.Text))
         Else

            MessageBox.Show("You did not provide all the X and Y" & _
               " values", "Missing X Y values", _
               MessageBoxButtons.OK, MessageBoxIcon.Information)

         End If

      End If

   End Sub ' btnAdd_Click

   ' clear the application GUI
   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      ' clear all fields
      txtX1.Text = ""
      txtX2.Text = ""
      txtY1.Text = ""
      txtY2.Text = ""
      txtXPosition.Text = ""
      txtYPosition.Text = ""
      txtWidth.Text = ""
      txtHeight.Text = ""

      ' clear PictureBox
      picImage.Invalidate()

   End Sub ' btnClear_Click
```

```
End Class ' FrmLogo
```

26.13 (*Letter Head Designer Application*) Create a **LetterHead** application that allows users to design stationery for company documents (Fig. 26.33). Allow users to specify the image that will serve as the letterhead.

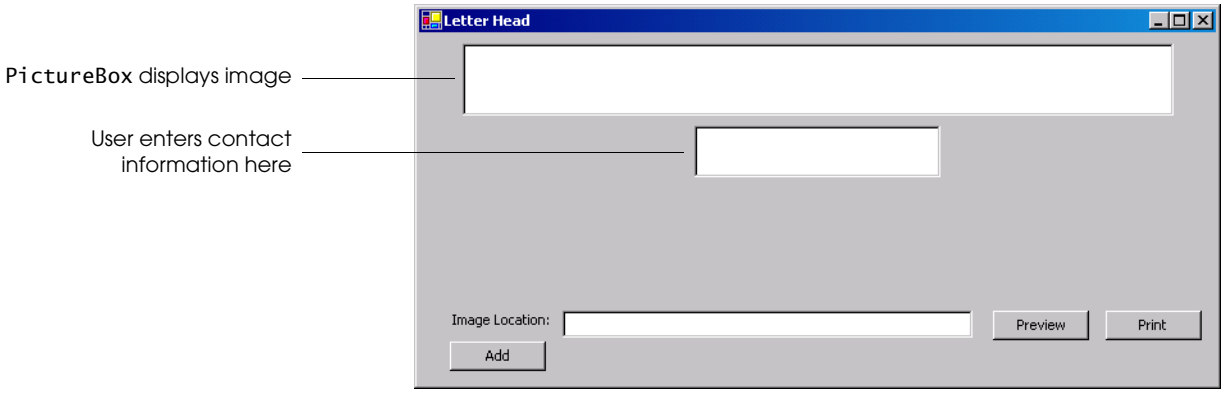


Figure 26.33 **Letter Head** GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial26\Exercises\LetterHead` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `LetterHead.sln` in the `LetterHead` directory to open the application.

c) ***Creating a `PrintPreviewDialog` control.*** Add a `PrintPreviewDialog` control to allow users to preview the letterhead before it is printed.

d) ***Defining the `PrintPage` event handler.*** Allow users to print the document by defining the `PrintPage` event handler as you did in the **CheckWriter** application.

e) ***Defining the `btnPrint_Click` event handler.*** The `btnPrint_Click` event handler should tell the `PrintDocument` where to find the `PrintPage` event handler, as in the **CheckWriter** application, and print the document.

f) ***Defining the `btnPreview_Click` event handler.*** The `btnPreview_Click` event handler should tell the `PrintDocument` where to find the `PrintPage` event handler, as in the **CheckWriter** application, and then show the preview dialog.

g) ***Testing the application.*** The `Letterhead.png` image file, located in `C:\Examples\Tutorial26\Exercises\Images` has been provided for you to test the application's letter head image capability.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter your contact information and specify the location of an image. [Note: An image has been supplied in an `Images` directory, located in your `C:\Examples\Tutorial26\Exercises` directory. The image should be displayed in the `PictureBox` at the top of the `Form`. Click the **Preview** `Button` and verify that the image and contact information is displayed in the preview. Finally, click the **Print** `Button` to verify that the letterhead prints with the appropriate image and contact information.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 26.13 Solution
' LetterHead.vb

Imports System.Drawing.Printing

Public Class FrmLetterHead
```

```
    Inherits System.Windows.Forms.Form

    ' create font object
    Private m_objFont As Font

    ' Windows Form Designer generated code

    ' PrintPage event raised for each page to be printed.
    Private Sub objPrintDocument_PrintPage(ByVal sender _
       As Object, ByVal e As PrintPageEventArgs)

       Dim sngYPosition As Single
       Dim sngXPosition As Single

       Dim sngLeftMargin As Single = e.MarginBounds.Left
       Dim sngTopMargin As Single = e.MarginBounds.Top

       Dim strPath As String

       ' get location of image
       strPath = txtImage.Text

       ' make sure image location was provided
       If strPath <> "" Then
          Dim objImage As Image
          objImage = Image.FromFile(strPath)

          ' print image so it is on top of page
          e.Graphics.DrawImage(Image.FromFile(strPath), _
             sngLeftMargin + picImage.Location.X, _
             sngTopMargin + picImage.Location.Y, _
             picImage.Size.Width, picImage.Size.Height)

       End If

       ' if contact information is provided, print data
       If txtInformation.Text <> "" Then

          ' specifies font of text
          m_objFont = New Font("Tahoma", 12, _
             FontStyle.Bold)

          sngXPosition = sngLeftMargin + _
             txtInformation.Location.X

          sngYPosition = sngTopMargin + _
             txtInformation.Location.Y

          ' print information
          e.Graphics.DrawString(txtInformation.Text, m_objFont, _
             Brushes.Black, sngXPosition, sngYPosition)
       End If

       ' indicate there are no more pages to print
       e.HasMorePages = False

    End Sub ' objPrintDocument_PrintPage

    ' print the document
    Private Sub btnPrint_Click(ByVal sender As _
       System.Object, ByVal e As System.EventArgs) _
```

```
      Handles btnPrint.Click

      ' create new object to assist in printing
      Dim objPrintDocument As New PrintDocument

      ' tell printer where to find PrintPage event handler
      AddHandler objPrintDocument.PrintPage, _
         AddressOf objPrintDocument_PrintPage

      ' print the document
      objPrintDocument.Print()

   End Sub ' btnPrint_Click

   ' display document in print preview dialog
   Private Sub btnPreview_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnPreview.Click

      Dim objPrintDocument As PrintDocument = _
         New PrintDocument

      AddHandler objPrintDocument.PrintPage, _
         AddressOf objPrintDocument_PrintPage

      objPreview.Document = objPrintDocument
      objPreview.ShowDialog()

   End Sub ' btnPreview_Click

   ' add the specified image to the Image control
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' import the image specified
      If txtImage.Text <> "" Then

         picImage.Image = Image.FromFile(txtImage.Text)

      Else

         MessageBox.Show("You must enter a path for your image.", _
            "Input Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

      EndIf

   End Sub ' btnAdd_Click

End Class ' FrmLetterHead
```

What does this code do?

26.14 What is the result of the following code? Assume that `objOutput_PrintPage` is defined.

```
Private Sub btnPrint_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnPrint.Click

   Dim objOutput As New PrintDocument

```

```
    AddHandler objOutput.PrintPage, _
       AddressOf objOutput_PrintPage

    objPrintOutput.Print()

End Sub ' btnPrint_Click
```

Answer: The code indicates that the `PrintPage` event for `objOutput` should invoke the `objOutput_PrintPage` event handler.

What's wrong with this code?

26.15 Find the error(s) in the following code. This is the definition for a `Click` event handler for a `Button`. This event handler should draw a rectangle on a `PictureBox` control.

```
Private Sub btnDrawImage_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnDrawImage.Click

   ' create an orange colored brush
   Dim objBrush As SolidBrush = New SolidBrush(Orange)

   ' create a Graphics object to draw on the PictureBox
   Dim objGraphics As Graphics = picPictureBox.AcquireGraphics

   ' draw a filled rectangle
   objGraphics.FillRectangle(objBrush, 2, 3, 40, 30)

End Sub ' btnDrawImage_Click
```

Answer: When specifying a color for the `SolidBrush` you must precede the color name with "`Color.`" You cannot just write the color name. Also, to retrieve the `Graphics` object, the `CreateGraphics` method must be used, not `AcquireGraphics`. The corrected code is shown below.

```
Private Sub btnDrawImage_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnDrawImage.Click

   ' create an orange colored brush
   Dim objBrush As SolidBrush = New SolidBrush(Color.Orange)

   ' create a Graphics object to draw on the PictureBox
   Dim objGraphics As Graphics = picPictureBox.CreateGraphics

   ' draw a filled rectangle
   objGraphics.FillRectangle(objBrush, 2, 3, 40, 30)

End Sub ' btnDrawImage_Click
```

Programming Challenge

26.16 (***Screen Saver Simulator Application***) Develop an application that simulates a screen saver. This application should add random-colored, random-sized, solid and hollow shapes at different positions of the screen (Fig. 26.34). Copy the `C:\Exercises\Tutorial26\ScreenSaver` directory, and place it in your `C:\SimplyVB` directory. The design of the `Form` has been created, which consists of a black `Form` and a `Timer` control. In the `ScreenSaver.vb` code view, the `DisplayShape` method has been provided and the `Timer`'s tick event handler has already been defined for you.

You must write the rest of the `DisplayShape` method code. Create the `Graphics` object from the `Form` using the `Form`'s `CreateGraphics` method, and specify random colors, sizes and positions for the filled and hollow shapes that will be displayed on the screen. The width and height of the shapes should be no larger than 100 pixels.

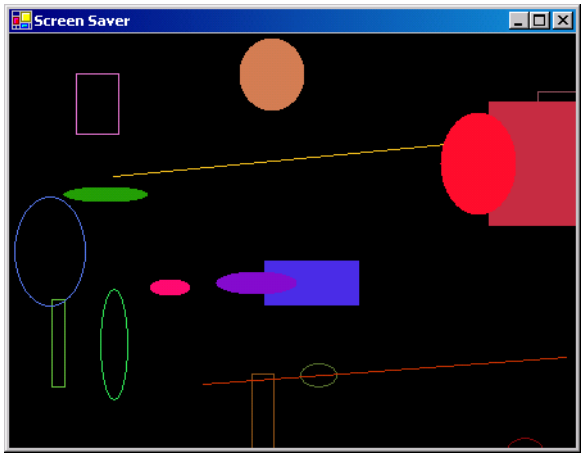


Figure 26.34 Screen Saver running.

Answer:

```
' Exercise 26.16 Solution
' ScreenSaver.vb

Public Class FrmShapeChanger
   Inherits System.Windows.Forms.Form

   Private m_dblCount As Double = 0.0

   ' Windows Form Designer generated code

   ' create a specified graphic on the form
   Private Sub DisplayShape()

      ' create a Graphics object
      Dim objGraphicsObject As Graphics = Me.CreateGraphics

      ' create random object for random number generation
      Dim objRandom As Random = New Random

      ' create random color
      Dim objColor As Color = Color.FromArgb( _
         objRandom.Next(0, 255), objRandom.Next(0, 255), _
         objRandom.Next(0, 255))

      ' create brush object used for solid shapes
      Dim objBrush As SolidBrush = New SolidBrush( _
          objColor)

      ' create pen object used for unfilled shapes
      Dim objPen As Pen = New Pen(objBrush)

      ' create random number used to create random shape
      Dim intShape As Integer = objRandom.Next(0, 5)

      ' set to width of form
      Dim intWidth As Integer = Me.Size.Width

      ' set to height of form
      Dim intHeight As Integer = Me.Size.Height

      ' decide which shape to draw
```

```
      Select Case intShape
         Case 0

            ' create filled rectangle
            objGraphicsObject.FillRectangle(objBrush, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
               objRandom.Next(10, 100), _
               objRandom.Next(10, 100))

         Case 1

            ' create filled ellipse
            objGraphicsObject.FillEllipse(objBrush, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
               objRandom.Next(10, 100), _
               objRandom.Next(10, 100))

         Case 2

            ' draw ellipse
            objGraphicsObject.DrawEllipse(objPen, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
               objRandom.Next(10, 100), _
               objRandom.Next(10, 100))

         Case 3

            ' draw rectangle
            objGraphicsObject.DrawRectangle(objPen, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
               objRandom.Next(10, 100), _
               objRandom.Next(10, 100))

         Case 4

            ' draw line
            objGraphicsObject.DrawLine(objPen, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
               objRandom.Next(10, Width), _
               objRandom.Next(10, Height))
      End Select

   End Sub ' DisplayShape

   ' invoked each time tmrScreenSaver ticks
   Private Sub tmrScreenSaver_Tick(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles tmrScreenSaver.Tick

      m_dblCount += 0.25

      ' draw shape every half second
      If m_dblCount Mod 2.5 = 0 Then
         DisplayShape() ' draw another shape
      End If

   End Sub ' tmrScreenSaver_Tick
```

```

End Class ' FrmShapeChanger
```

26.17 (*Screen Saver Simulator Enhancement Application*) Enhance the **Screen Saver Simulator** application from Exercise 26.16 by modifying the `Timer` control's `Tick` event handler. Add code to this event handler so that after a specified amount of time, the screen should clear the displayed shapes. After the screen clears, random shapes should continue to display. Also, modify the code so that you can specify random opacity (alpha values) for the colors by using `Color` structure's `FromArgb` method. You should pass four arguments to this method. The first argument is the alpha value, the second is the red value, the third is the green value and the fourth is the blue value.

Answer:

```
' Exercise 26.17 Solution
' ScreenSaver.vb

Public Class FrmShapeChanger
   Inherits System.Windows.Forms.Form

   Private m_dblCount As Double = 0.0

   ' Windows Form Designer generated code

   ' create a specified graphic on the form
   Private Sub DisplayShape()

      ' create a Graphics object
      Dim objGraphicsObject As Graphics = Me.CreateGraphics

      ' create random object for random number generation
      Dim objRandom As Random = New Random

      ' create random color with random opacity
      Dim objColor As Color = Color.FromArgb( _
         objRandom.Next(0, 255), objRandom.Next(0, 255), _
         objRandom.Next(0, 255), objRandom.Next(0, 255))

      ' create brush object used for solid shapes
      Dim objBrush As SolidBrush = New SolidBrush(objColor)

      ' create pen object used for unfilled shapes
      Dim objPen As Pen = New Pen(objBrush)

      ' create random number used to create random shape
      Dim intShape As Integer = objRandom.Next(0, 5)

      ' set to width of form
      Dim intWidth As Integer = Me.Size.Width

      ' set to height of form
      Dim intHeight As Integer = Me.Size.Height

      ' decide which shape to draw
      Select Case intShape
         Case 0

            ' create filled rectangle
            objGraphicsObject.FillRectangle(objBrush, _
               objRandom.Next(0, Width), _
               objRandom.Next(0, Height), _
```

```
                 objRandom.Next(10, 100), _
                 objRandom.Next(10, 100))

           Case 1

              ' create filled ellipse
              objGraphicsObject.FillEllipse(objBrush, _
                 objRandom.Next(0, Width), _
                 objRandom.Next(0, Height), _
                 objRandom.Next(10, 100), _
                 objRandom.Next(10, 100))

           Case 2

              ' draw ellipse
              objGraphicsObject.DrawEllipse(objPen, _
                 objRandom.Next(0, Width), _
                 objRandom.Next(0, Height), _
                 objRandom.Next(10, 100), _
                 objRandom.Next(10, 100))

           Case 3

              ' draw rectangle
              objGraphicsObject.DrawRectangle(objPen, _
                 objRandom.Next(0, Width), _
                 objRandom.Next(0, Height), _
                 objRandom.Next(10, 100), _
                 objRandom.Next(10, 100))

           Case 4

              ' draw line
              objGraphicsObject.DrawLine(objPen, _
                 objRandom.Next(0, Width), _
                 objRandom.Next(0, Height), _
                 objRandom.Next(10, Width), _
                 objRandom.Next(10, Height))
        End Select

     End Sub ' DisplayShape

     ' invoked each time tmrScreenSaver ticks
     Private Sub tmrScreenSaver_Tick(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles tmrScreenSaver.Tick

        m_dblCount += 0.25

        ' draw shape every half second
        If m_dblCount Mod 2.5 = 0 Then
           DisplayShape() ' draw another shape
        End If

        If m_dblCount Mod 200 = 0 Then
           Me.Invalidate() ' clear screen
        End If

     End Sub ' tmrScreenSaver_Tick

  End Class ' FrmShapeChanger
```

TUTORIAL 27

Phone Book Application

Introducing Multimedia Using Microsoft Agent

Solutions

Instructor's Manual Exercise Solutions Tutorial 27

MULTIPLE-CHOICE QUESTIONS

27.1 The _____ method is used to specify what the Microsoft Agent will say.

a) `Speak` b) `Say`
c) `Command` d) `Voice`

27.2 The _____ method is used to activate a Microsoft Agent character's animation.

a) `Show` b) `Play`
c) `Speak` d) `Appear`

27.3 Method `MoveTo` takes two arguments. What do these arguments represent?

a) The direction in which the Agent should move (left, right, up, down).
b) The name of the character and its position.
c) The *x*-coordinate and *y*-coordinate of the location to which the Agent object should move.
d) The name of the character and the direction of movement.

27.4 Which method of `IAgentCtlCharacter` displays the Microsoft Agent character on the screen?

a) `Play` b) `Show`
c) `Speak` d) `Appear`

27.5 Use the _____ event handler to execute code when users click **Hide** the Agent character context menu.

a) `Hide` b) `HideEvent`
c) `Command` d) `Disappear`

27.6 The `Add` method of the `Commands` property _____.

a) adds a new command to the command list.
b) joins two commands together.
c) displays the `Commands` pop-up window.
d) both a and c

27.7 The _____ event handler controls what occurs when users speak to the Agent.

a) `Command` b) `ClickEvent`
c) `Click` d) `SelectedIndexChanged`

27.8 _____ specifies the *x*-coordinate of the mouse cursor on the screen.

a) `Cursor.Location.X` b) `Cursor.Position.X`
c) `Mouse.Location.X` d) `Mouse.Position.X`

27.9 Specifying _____ as a parameter to Peedy's `Play` method causes him to smile.

a) `"Think"` b) `"Smile"`
c) `"Pleased"` d) `"Happy"`

27.10 Specifying _____ as a parameter to Peedy's `Play` method causes him to rest.

a) `"RestPose"` b) `"Rest"`
c) `"Think"` d) `"Pose"`

Answers: 27.1) a. 27.2) b. 27.3) c. 27.4) b. 27.5) b. 27.6) a. 27.7) a. 27.8) b. 27.9) c. 27.10) a.

EXERCISES

27.11 (*Appointment Book Application Using Microsoft Agent*) Write an application that allows users to add appointments to an appointment book that uses Microsoft Agent. When users speak a person's name, Merlin returns the time and date of the appointment that users have with that person. If users say "Today", Merlin returns a list of the users' appointments for the day.

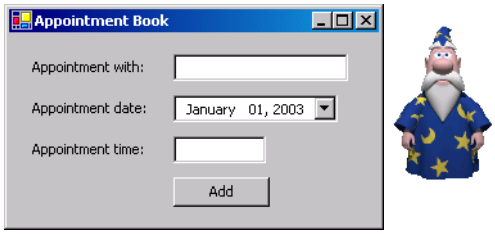


Figure 27.32 Appointment Book GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\AppointmentBook` to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `AppointmentBook.sln` in the `AppointmentBook` directory to open the application.

c) ***Downloading the Merlin Microsoft Agent.*** Download the `Merlin.acs` character file from the Microsoft Web site.

d) ***Adding the Agent Control to the `Form`.*** Add the Microsoft Agent control to the `Form`.

e) ***Creating module-level variables.*** Create three module-level variables of type `ArrayList` to store the date, time and person with which the user has an appointment. Create a module-level variable of type `AgentObjects.IAgentCtlCharacter` (as you did in the **Phone Book** application).

f) ***Defining the `FrmAppointments_Load` event handler.*** Load Merlin's character file, display him on the screen and add the "Today" command to the command list.

g) ***Defining the `btnAdd_Click` event handler.*** Define this event handler so that the information provided by the user is added to its corresponding `ArrayList`. The **Appointment With:** `TextBox` input should be added to the `ArrayList` containing the names of people with whom the user has an appointment. The input for the appointment date and time should also be added to their respective `ArrayLists`. Display an error message if the user leaves the **Appointment With:** or the **Appointment Time:** `TextBox` empty.

h) ***Adding voice-enabled commands.*** Within the `btnAdd_Click` event handler, add a voice-enabled command that allows a user to speak the name of the person with whom the user has an appointment to the command list. This allows a user to check for whether there is an appointment with someone by speaking the person's name. The command should also appear in the `Commands` context menu.

i) ***Defining the Agent's `Command` event handler.*** As you did in the **Phone Book** application, define what occurs when a user speaks or selects a command. If the user specifies the `Today` command, Merlin should tell the user the names of all the people with whom the user has an appointment today. If the user specifies a specific name, Merlin should state the time and date at which the user has an appointment with this person. If the user did not schedule any appointments, then Merlin should inform the user that no appointments were scheduled.

j) ***Running the application.*** Select **Debug > Start** to run your application. Enter various appointments, where at least two of the appointments are scheduled for the current day. Input the name of the person you are meeting at one of the appointments by either speaking the name into your microphone, or right-clicking the agent and selecting that person's name. Verify that the agent repeats back correct information about that appointment. Input the value "Today" by either speaking it into the microphone or right-clicking the agent and selecting **Today**. Verify that the agent repeats back all the appointments for the current day.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 27.11 Solution
' AppointmentBook.vb

Public Class FrmAppointments
   Inherits System.Windows.Forms.Form

   ' create three Arraylists
   Private m_objPerson As ArrayList = New ArrayList
   Private m_objDate As ArrayList = New ArrayList
   Private m_objTime As ArrayList = New ArrayList

   ' represent current Agent
   Private m_objMSpeaker As AgentObjects.IAgentCtlCharacter

   ' Windows Form Designer generated code

   ' invoked when Form is loaded
   Private Sub FrmAppointments_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      ' load agent character file
      objMainAgent.Characters.Load("Merlin", "Merlin.acs")

      ' specify current agent
      m_objMSpeaker = objMainAgent.Characters("Merlin")

      ' show Merlin on screen
      m_objMSpeaker.Show(0)

      ' add voice enabled command to Agent object
      m_objMSpeaker.Commands.Add("Today", "Today", _
          "Today", True, True)
   End Sub 'FrmAppointments_Load

   ' add appointments to ArrayLists
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      If txtWith.Text = "" OrElse txtTime.Text = "" Then

         MessageBox.Show( _
            "You left one or more fields empty above", _
            "Empty Field", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)
      Else
         ' add information to ArrayLists
         m_objPerson.Add(txtWith.Text)
         m_objDate.Add(dtpAddDate.Value.ToLongDateString)
         m_objTime.Add(txtTime.Text)

         m_objMSpeaker.Commands.Add(txtWith.Text, _
            txtWith.Text, txtWith.Text, True, True)

         ' clear TextBoxes
         txtWith.Clear()
         txtTime.Clear()
```

```
         End If

      End Sub ' btnAdd_Click

      ' determine what Agent does when it hears command
      Private Sub objMainAgent_Command(ByVal sender As Object, _
         ByVal e As AxAgentObjects._AgentEvents_CommandEvent) _
         Handles objMainAgent.Command

         ' get UserInput object
         Dim objCommand As AgentObjects.IAgentCtlUserInput = _
             CType(e.userInput, AgentObjects.IAgentCtlUserInput)

         Dim intCount As Integer

         ' boolean to determine if user has appointments
         Dim blnAppointments As Boolean = False
         Dim strAppointments As String = _
            "Today you have an appointment with:"

         ' determine if user spoke Today command
         If objCommand.Name = "Today" Then

            ' search objDate arraylist for appointments
            ' set for today
            For intCount = 0 To m_objDate.Count - 1

               If Convert.ToString(m_objDate(intCount)) = _
                  Today.Date.ToLongDateString Then

                  ' add the appointment to string Agent speaks
                  strAppointments = strAppointments & _
                      Convert.ToString(m_objPerson(intCount)) _
                      & " at " & _
                      Convert.ToString(m_objTime(intCount)) _
                      & ", "

                  blnAppointments = True
               End If

            Next

         Else
            ' specify string Agent speaks
            strAppointments = "You have to " & _
                "meet with " & _
                Convert.ToString(objCommand.Name) & _
                " on: "

            ' determine if user made command using
            ' person's name
            For intCount = 0 To m_objPerson.Count - 1

               If objCommand.Name = _
                  Convert.ToString(m_objPerson(intCount)) Then

                  ' add date and time to string Agent speaks
                  strAppointments = strAppointments & _
                      Convert.ToString(m_objDate(intCount)) _
                      & " at " & _
                      Convert.ToString(m_objTime(intCount)) _
```

```
                    & ", "

                 blnAppointments = True
              End If

           Next

        End If

        ' if user has no appointments, specify in strAppointments
        If blnAppointments = False Then
           strAppointments = "You have no scheduled " _
              & "appointments."
        End If

        ' Agent speaks strAppointments string
        m_objMSpeaker.Speak(strAppointments)
     End Sub ' objMainAgent_Command

End Class ' FrmAppointments
```

27.12 *(Craps Game Application Using Microsoft Agent)* Modify the **Craps Game** application from Tutorial 16 to include a Microsoft Agent character.

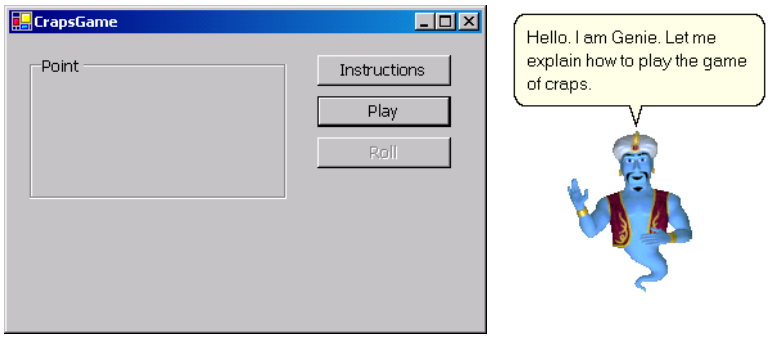


Figure 27.33 Modified **Craps Game** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\CrapsGameEnhancement` to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `CrapsGame.sln` in the `CrapsGameEnhancement` directory to open the application.

c) ***Downloading the Genie Microsoft Agent.*** Download the `Genie.acs` character file from the Microsoft Web site.

d) ***Adding the Agent control to the `Form`.*** Add the Microsoft Agent control to the `Form`.

e) ***Creating a module-level variable.*** Create a module-level variable of type `AgentObjects.IAgentCtlCharacter` (as you did in the **Phone Book** application).

f) ***Defining the `FrmCrapsGame_Load` event handler.*** Load Genie's character file, and display him on the screen.

g) ***Modifying the `btnPlay_Click` event handler.*** Add code to the `btnPlay_Click` event handler to control the Agent. When the user wins the game, Genie should play his `Pleased` animation and congratulate the user. If the user loses, Genie should play his `Confused` animation and say that the user lost. If the user neither wins nor loses, Genie should tell the user to roll again. Make sure to reset him to his `RestPose` after he plays any animation.

h) ***Defining the `btnRoll_Click` event handler.*** Add code to the `btnRoll_Click` event handler to control the Agent. If users "make their point," Genie should play his `Pleased` animation and state that the user won. If the user rolls a 7, Genie should

play his `Confused` animation and say that the user lost. Otherwise, Genie should tell the user to roll again.

i) ***Defining the `btnInstructions_Click` event handler.*** Define the `btnInstructions_Click` event handler to make Genie introduce himself to the user. Genie should then explain the rules to the game of craps.

j) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Instructions** `Button` and allow the agent character to tell you the rules of the game. Use the **Play** and **Roll** `Buttons` to play a few games of craps. When you need to roll again, verify that the agent tells you to roll again. Also, verify that the agents informs you whether you won or lost at the end of each game.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 27.12 Solution
' CrapsGame.vb

Imports System.IO

Public Class FrmCrapsGame
   Inherits System.Windows.Forms.Form

   ' die-roll constants
   Enum DiceNames
      SNAKE_EYES = 2
      TREY = 3
      CRAPS = 7
      LUCKY_SEVEN = 7
      YO_LEVEN = 11
      BOX_CARS = 12
   End Enum

   ' filename and directory constants
   Const m_strFILE_PREFIX As String = "/images/die"
   Const m_strFILE_SUFFIX As String = ".png"

   ' instance variables
   Private m_intMyPoint As Integer = 0
   Private m_objRandom As Random = New Random

   ' create object that represents current agent
   Private m_objMSpeaker As AgentObjects.IAgentCtlCharacter

   ' Windows Form Designer generated code

   ' begin new game and determine point
   Private Sub btnPlay_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnPlay.Click

      ' initialize variables for new game
      m_intMyPoint = 0
      fraPointDiceGroup.Text = "Point"
      lblStatus.Text = ""

      ' remove point-die images
      picPointDie1.Image = Nothing
      picPointDie2.Image = Nothing

      Dim intSum As Integer = RollDice() ' roll dice
```

```

        ' check die roll
        Select Case intSum

           Case DiceNames.LUCKY_SEVEN, _
              DiceNames.YO_LEVEN ' win on first roll

              btnRoll.Enabled = False ' disable roll button
              lblStatus.Text = "You win!!!"

              ' play Agent's Pleased animation
              m_objMSpeaker.Play("Pleased")

              ' make Agent speak
              m_objMSpeaker.Speak("Congratulations, you won!")

              m_objMSpeaker.Play("RestPose")

           Case DiceNames.SNAKE_EYES, _
              DiceNames.TREY, _
              DiceNames.BOX_CARS ' lose on first roll

              btnRoll.Enabled = False
              lblStatus.Text = "Sorry. You lose."

              ' play Agent's Confused animation
              m_objMSpeaker.Play("Confused")

              ' make Agent speak
              m_objMSpeaker.Speak("Sorry, you lost!")

              m_objMSpeaker.Play("RestPose")

           Case Else ' player must match point
              m_intMyPoint = intSum
              fraPointDiceGroup.Text = "Point is " & intSum
              lblStatus.Text = "Roll Again!"
              m_objMSpeaker.Speak("Please roll again.")
              picPointDie1.Image = picDie1.Image
              picPointDie2.Image = picDie2.Image
              DisplayDie(picPointDie2, m_intMyDie2)
              btnPlay.Enabled = False ' disable Play Button
              btnRoll.Enabled = True  ' enable Roll Button
        End Select

     End Sub ' btnPlay_Click

     ' determine outcome of next roll
     Private Sub btnRoll_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnRoll.Click

        Dim intSum As Integer = RollDice()

        ' determine outcome of roll
        If intSum = m_intMyPoint Then  ' player matches point
           lblStatus.Text = "You win!!!"
           m_objMSpeaker.Play("Pleased")
           m_objMSpeaker.Speak("Congratulations, you won!")
           m_objMSpeaker.Play("RestPose")

           btnRoll.Enabled = False
```

```
            btnPlay.Enabled = True
         ElseIf intSum = DiceNames.CRAPS Then ' player loses
            lblStatus.Text = "Sorry. You lose."
            m_objMSpeaker.Play("Confused")
            m_objMSpeaker.Speak("Sorry, you lost!")
            m_objMSpeaker.Play("RestPose")

            btnRoll.Enabled = False
            btnPlay.Enabled = True

         Else
            m_objMSpeaker.Speak("Please roll again.")
         End If

      End Sub ' btnRoll_Click

      ' generate random die rolls
      Private Function RollDice() As Integer

         ' roll the dice
         Dim intDie1 As Integer = m_objRandom.Next(1, 7)
         Dim intDie2 As Integer = m_objRandom.Next(1, 7)

         ' display image corresponding to each die
         DisplayDie(picDie1, intDie1)
         DisplayDie(picDie2, intDie2)

         Return (intDie1 + intDie2) ' return sum of dice values

      End Function ' RollDice

      ' display die image
      Private Sub DisplayDie(ByVal picDie As PictureBox, _
         ByVal intFace As Integer)

         ' assign die images to PictureBoxes
         picDie.Image = _
            Image.FromFile(Directory.GetCurrentDirectory & _
            m_strFILE_PREFIX & intFace & m_strFILE_SUFFIX)

      End Sub ' DisplayDie

      ' invoked when Form is loaded
      Private Sub FrmCrapsGame_Load(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles MyBase.Load

         ' load Genie character file
         objMainAgent.Characters.Load("Genie", "Genie.acs")

         m_objMSpeaker = objMainAgent.Characters("Genie")

         ' show Genie on the screen
         m_objMSpeaker.Show(0)
      End Sub ' FrmCrapsGame_Load

      ' Agent explains instructions of the game
      Private Sub btnInstructions_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnInstructions.Click

         m_objMSpeaker.Play("Wave")

```

```
        m_objMSpeaker.Speak("Hello. I am Genie. Let me " & _
            "explain how to play the game of craps.")

        m_objMSpeaker.Speak("Click the Play button to begin " & _
            "the game.")

        m_objMSpeaker.Speak("Clicking Play causes you to roll" _
            & " two dice.")

        m_objMSpeaker.Speak("If the sum of your dice roll is " _
            & "7 or 11 on your first throw, then you win.")

        m_objMSpeaker.Speak("However, if the sum is 2, 3, or " _
          & "12 on your first throw, then you lose.")

        m_objMSpeaker.Speak("If the sum is 4, 5, 6, 7, 8, 9 " _
            & "or 10 on your first throw, then that sum " _
            & "becomes your 'point'")

        m_objMSpeaker.Speak("To win, you must continue rolling" _
            & " the dice until you roll the point value again.")

        m_objMSpeaker.Speak("You lose if you roll a 7 before " _
            & "reaching your point value.")

        m_objMSpeaker.Play("RestPose")
    End Sub ' btnInstructions_Click

End Class ' FrmCrapsGame
```

27.13 (*Security Panel Application Using Microsoft Agent*) Modify the **Security Panel** application from Tutorial 12 to include Microsoft Agent.

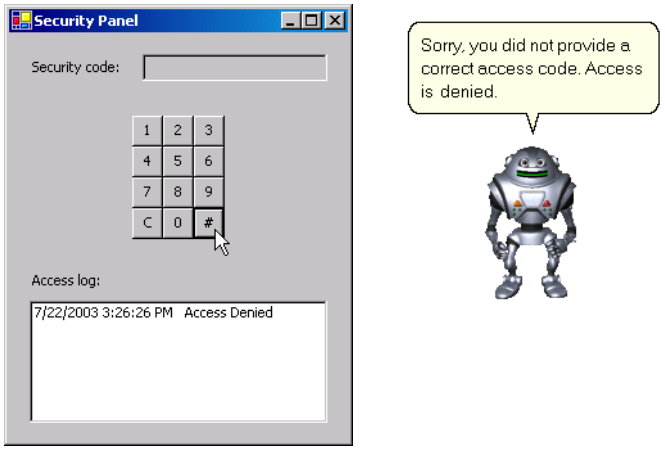


Figure 27.34 Robby from modified **Security Panel** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\SecurityPanelEnhancement` to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `SecurityPanel.sln` in the `SecurityPanelEnhancement` directory to open the application.

c) ***Downloading the Robby Microsoft Agent.*** Download the `Robby.acs` character file from the Microsoft Web site.

d) ***Adding the Agent control to the Form.*** Add the Microsoft Agent control to the `Form`.

e) ***Creating a module-level variable.*** Create a module-level variable of type `AgentObjects.IAgentCtlCharacter` (as you did in the **Phone Book** application).

f) ***Defining the `FrmSecurityPanel_Load` event handler.*** Load Robby's character file, and display him on the screen. Command Robby to tell users to input their access codes.

g) ***Modifying the `btnEnter_Click` event handler.*** Add code to the `btnEnter_Click` event handler to use the Microsoft Agent. If the user enters a valid access code, Robby should welcome the user and state the type of employee that the access code represents. If the access code is invalid, then Robby should state that an invalid code was provided and that access is denied.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter various access codes. For correct access codes, verify that the agent tells you what type of employee the access code represents. For incorrect access codes, verify that the agents tells you that access is denied.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 27.13 Solution
' SecurityPanel.vb

Public Class FrmSecurityPanel
   Inherits System.Windows.Forms.Form

   Private m_objMSpeaker As AgentObjects.IAgentCtlCharacter

   ' Windows Form Designer generated code

   Private Sub btnEnter_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles btnEnter.Click

      Dim intAccessCode As Integer ' stores access code entered
      Dim strMessage As String  ' displays access status of users

      intAccessCode = Convert.ToInt32(txtSecurityCode.Text)
      txtSecurityCode.Clear()

      Select Case intAccessCode ' check access code input

         ' access code less than 10
         Case Is < 10
            strMessage = "Restricted Access"
            m_objMSpeaker.Speak("Welcome. You have " _
                & "restricted access.")

         ' access code between 1645 and 1689
         Case 1645 To 1689
            strMessage = "Technicians"
            m_objMSpeaker.Speak("Welcome. Your access" _
                & " code indicates that you are " _
                & "part of the Technician Personnel.")

         ' access code equal to 8345
         Case 8345
            strMessage = "Custodians"
            m_objMSpeaker.Speak("Welcome. Your access" _
                & " code indicates that you are " _
                & "part of Custodial Services.")

```

```
            ' access code equal to 9998 or between
            ' 1006 and 1008
            Case 9998, 1006 To 1008
               strMessage = "Scientists"
               m_objMSpeaker.Speak("Welcome. Your access" _
                   & " code indicates that you are " _
                   & "part of the Scientific Personnel.")

            ' if no other Case is True
            Case Else
               strMessage = "Access Denied"
               m_objMSpeaker.Speak("Sorry, you did not " _
                   & "provide a correct access code." _
                   & " Access is denied.")

         End Select

         ' display time and message in ListBox
         lstLogEntry.Items.Add(Date.Now & "    " & strMessage)
      End Sub ' btnEnter_Click

      Private Sub btnZero_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnZero.Click

         txtSecurityCode.Text &= "0"  ' concatenate "0" to display
      End Sub ' btnZero_Click

      Private Sub btnOne_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnOne.Click

         txtSecurityCode.Text &= "1"  ' concatenate "1" to display
      End Sub ' btnOne_Click

      Private Sub btnTwo_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnTwo.Click

         txtSecurityCode.Text &= "2"  ' concatenate "2" to display
      End Sub ' btnTwo_Click

      Private Sub btnThree_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnThree.Click

         txtSecurityCode.Text &= "3"  ' concatenate "3" to display
      End Sub ' btnThree_Click

      Private Sub btnFour_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnFour.Click

         txtSecurityCode.Text &= "4"  ' concatenate "4" to display
      End Sub ' btnFour_Click

      Private Sub btnFive_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnFive.Click

         txtSecurityCode.Text &= "5"  ' concatenate "5" to display
      End Sub ' btnFive_Click

      Private Sub btnSix_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnSix.Click

         txtSecurityCode.Text &= "6"  ' concatenate "6" to display
```

```
   End Sub ' btnSix_Click

   Private Sub btnSeven_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnSeven.Click

      txtSecurityCode.Text &= "7"  ' concatenate "7" to display
   End Sub ' btnSeven_Click

   Private Sub btnEight_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEight.Click

      txtSecurityCode.Text &= "8"  ' concatenate "8" to display
   End Sub ' btnEight_Click

   Private Sub btnNine_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnNine.Click

      txtSecurityCode.Text &= "9"  ' concatenate "9" to display
   End Sub ' btnNine_Click

   Private Sub btnClear_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnClear.Click

      txtSecurityCode.Clear() ' clear text from TextBox
   End Sub ' btnClear_Click

   ' invoked when the Form is loaded
   Private Sub FrmSecurityPanel_Load(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles MyBase.Load

      ' load Agent character file
      objMainAgent.Characters.Load("Robby", "Robby.acs")

      ' specify current agent
      m_objMSpeaker = objMainAgent.Characters("Robby")

      ' show Robby on screen
      m_objMSpeaker.Show(0)

      ' Robby speaks
      m_objMSpeaker.Speak("Please enter your security " _
          & "access code.")
   End Sub ' FrmSecurityPanel_Load

End Class ' FrmSecurityPanel
```

What does this code do?

27.14 After the user clicks the **Call** `Button`, what does the following event handler do?

```
Private Sub btnCall_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnCall.Click

   objMainAgent.Characters.Load("Genie", "Genie.acs")

   objMSpeaker = objMainAgent.Characters("Genie")

   objMSpeaker.Show(0)

   objMSpeaker.Speak("Hello, I'm Genie the special agent!")
```

```
End Sub
```

Answer: The agent object is loaded as "Genie." Genie appears and says, "Hello, I'm Genie the special agent!"

What's wrong with this code?

27.15 Find the error(s) in the following code. The event handler should have an agent object appear and say, "Hello, my name is Merlin". This should happen when the user clicks the **Call** `Button`.

```
Private Sub btnCall_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnCall.Click

   objMainAgent.Characters.Load("Merlin", "Merlin.acs")

   objMSpeaker = objMainAgent.Characters("Merlin")

   Dim intNumber As Integer = 10

   objMSpeaker.Show(intNumber)

   objMSpeaker.Play("Hello, my name is Merlin")

End Sub
```

Answer: The `Play` method is called with an invalid argument. To have the agent say "Hello, my name is Merlin", you must call the `Speak` method.

```
Private Sub btnCall_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnCall.Click

   objMainAgent.Characters.Load("Merlin", "Merlin.acs")

   objMSpeaker = objMainAgent.Characters("Merlin")

   Dim intNumber As Integer = 10

   objMSpeaker.Show(intNumber)

   objMSpeaker.Speak("Hello, my name is Merlin")

End Sub
```

Programming Challenge

27.16 ***Car Payment Application Using Microsoft Agent***) Enhance the **Car Payment Calculator** application from Tutorial 9 to use the Microsoft Agent, Robby. When the application is executed, Robby should appear on the screen and wave to users. He should then explain the purpose of the application. After the user enters information into each field of the **Car Payment Calculator** and clicks the **Calculate** `Button`, Robby should speak the calculated payment amounts and the period (number of months) over which they were calculated. The `C:\Examples\Tutorial27\Exercises\CarPaymentCalculatorEnhancement` directory contains the template application for this exercise. Copy it to your working directory and open the application to begin the exercise.

Answer:

```
' Exercise 27.16 Solution
' CarPayment.vb

```

```
Public Class FrmCarPayment
   Inherits System.Windows.Forms.Form

   Private m_objMSpeaker As AgentObjects.IAgentCtlCharacter

   ' Windows Form Designer generated code

   ' handles Calculate Button's Click event
   Private Sub btnCalculate_Click( ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnEnter.Click

      Dim intYears As Integer = 2          ' repetition counter
      Dim intMonths As Integer = 0         ' payment period
      Dim intPrice As Integer = 0          ' car price
      Dim intDownPayment As Integer = 0    ' down payment
      Dim dblInterest As Double = 0        ' interest rate
      Dim decMonthlyPayment As Decimal = 0 ' monthly payment
      Dim intLoanAmount As Integer = 0     ' cost after down payment
      Dim dblMonthlyInterest As Double = 0 ' monthly interest rate
      Dim strAgentSpeak As String = "You will have to pay: "

      ' remove text displayed in ListBox
      lstPayments.Items.Clear()

      ' add header to ListBox
      lstPayments.Items.Add("Months" & ControlChars.Tab & _
         ControlChars.Tab & "Monthly Payments")

      ' retrieve user input and assign values
      ' to their respective variables
      intDownPayment = Convert.ToInt32(Val(txtDownPayment.Text))
      intPrice = Convert.ToInt32(Val(txtStickerPrice.Text))
      dblInterest = Convert.ToDouble(Val(txtInterest.Text)) / 100

      ' determine amount borrowed and monthly interest rate
      intLoanAmount = intPrice - intDownPayment
      dblMonthlyInterest = dblInterest / 12

      ' loop four times
      Do While intYears <= 5

         ' calculate payment period
         intMonths = 12 * intYears

         ' calculate monthly payment using Pmt
         decMonthlyPayment = Convert.ToDecimal( _
              Pmt(dblMonthlyInterest, intMonths, -intLoanAmount))

         ' display payment value
         lstPayments.Items.Add(intMonths & ControlChars.Tab & _
            ControlChars.Tab & String.Format("{0:C}", _
            decMonthlyPayment))

         If intYears < 5 Then

            ' specify format of payment values for first
            ' four years
            strAgentSpeak = strAgentSpeak & String.Format("{0:C}", _
               decMonthlyPayment) & " per month, over " & intMonths _
               & " months, "
         Else
```

```
            ' specify format for payment value of fifth year
            strAgentSpeak = strAgentSpeak & "or " & _
               String.Format("{0:C}", decMonthlyPayment) _
               & " per month, over " & intMonths _
               & " months."
         End If

         intYears += 1  ' increment counter
      Loop

      ' Robby speaks strAgentSpeak string
      m_objMSpeaker.Speak(strAgentSpeak)
   End Sub ' btnCalculate_Click

   ' invoked when Form is loaded
   Private Sub FrmCarPayment_Load( ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      ' load Robby character into agent object
      objMainAgent.Characters.Load("Robby", "Robby.acs")

      m_objMSpeaker = objMainAgent.Characters("Robby")

      ' show Robby on the screen
      m_objMSpeaker.Show(0)

      ' play Wave animation
      m_objMSpeaker.Play("Wave")

      ' make Robby speak instructions for application
      m_objMSpeaker.Speak("Hello, I will be your assistant.")

      m_objMSpeaker.Play("RestPose")

      m_objMSpeaker.Speak( _
         "You need to enter the price of a car," _
         & "  the down-payment amount and " _
         & " the annual interest rate of the loan.")

      m_objMSpeaker.Speak( _
         "After you input this information, " _
         & "I will calculate the monthly payments you will " _
         & "need to make for two-, three-, four- and " _
         & "five-year loans.")
   End Sub ' FrmCarPayment_Load

End Class ' FrmCarPayment
```

TUTORIAL 28

Bookstore Application: Web Applications

Introducing Internet Information Services

Solutions

Instructor's Manual Exercise Solutions Tutorial 28

MULTIPLE-CHOICE QUESTIONS

28.1 ASPX pages have the _____ extension.

a) `.html` b) `.wbform`
c) `.vbaspx` d) `.aspx`

28.2 _____ applications divide functionality into separate tiers.

a) *n*-tier b) Multi-tier
c) Both a and b. d) None of the above.

28.3 All tiers of a multi-tier application _____.

a) must be located on the same computer
b) must be located on different computers
c) can be located on the same computer or on different computers
d) must be arranged so that the client and middle tier are on the same computer and the information tier is on a different computer

28.4 The client tier interacts with the _____ tier to access information from the _____ tier.

a) middle; information b) information; middle
c) information; bottom d) bottom; information

28.5 A _____ is specialized software that responds to client requests by providing resources.

a) host b) host name
c) DNS server d) Web server

28.6 A(n) _____ can be thought of as an address that is used to direct a browser to a resource on the Web.

a) middle tier b) ASPX page
c) URL d) query string

28.7 A _____ represents a group of _____ on the Internet.

a) domain; hosts b) host; domain names
c) host name; hosts d) None of the above.

28.8 _____ is a Web server.

a) IIS b) `localhost`
c) Visual Studio .NET d) `wwwroot`

28.9 A _____ is a Web server that is located on a computer across a network such as the Internet.

a) `localhost` b) local Web server
c) remote Web server d) None of the above.

28.10 The _____ tier is the application's user interface.

a) middle b) client
c) bottom d) information

Answers: 28.1) d. 28.2) c. 28.3) c. 28.4) a. 28.5) d. 28.6) c. 28.7) a. 28.8) a. 28.9) c. 28.10) b.

EXERCISES

28.11 (*Phone Book Application*) Over the next three tutorials, you will create a **PhoneBook** application. This phone book should be a Web-based version of the **PhoneBook** appli-

cation created in Tutorial 27. [*Note*: This Web application will not use Microsoft Agent.] The **PhoneBook** application should consist of two ASPX pages, which will be named `PhoneBook` and `PhoneNumber`. The `PhoneBook` page displays a `DropDownList` (a Web control similar to a `ComboBox` Windows `Form` control) that contains the names of several people. The names are retrieved from the `db_Phone.mdb` database. When a name is selected and the **Get Number** `Button` is clicked, the client browser is redirected to the `PhoneNumber` page. The telephone number of the selected name should be retrieved from a database and displayed in a `Label` on the `PhoneNumber` page. For this exercise, you need only organize the components (`PhoneBook` and `PhoneNumber` ASPX pages, `db_Phone.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs. One page will contain a `DropDownList` control. The middle tier should contain the code used to retrieve the names and phone numbers from the database. The information tier should contain the `db_Phone.mdb` database where the phone-number information is stored.

28.12 (***US State Facts Application***) Over the next three tutorials, you will create a **USStateFacts** application. This application is designed to allow users to review their knowledge about specific U.S. states. This application should consist of two ASPX pages. The first page (named `States`) should display a `ListBox` containing 10 different state names. These state names are stored in the `db_StateFacts.mdb` database. The user should be allowed to select a state name and click a `Button` to retrieve information about the selected state from the database. The information should be displayed on a different ASPX page (named `StateFacts`). The `StateFacts` page should display an image of the state flag and list the state capital, state flower, state tree and state bird (retrieved from the database) in a `Table`. You will be provided with images of the state flags. For this exercise, you need only organize the components (`States` and `StateFacts` ASPX pages, `db_StateFacts.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs, including a `ListBox` for the different state names. The middle tier should contain the code used to retrieve the state names and information from the database. The information tier should contain the `db_StateFacts.mdb` database where the state information is stored.

28.13 (***Road Sign Review Application***) Over the next three tutorials, you will create a **RoadSignReview** application. The **RoadSignReview** application should consist of two ASPX pages. This application displays road signs for users to review and allows them to schedule a driving test. The first page (named `RoadSigns`) should display 15 road signs in a `Table`. You will be provided images of the road signs. When the mouse pointer is moved over a sign, the name of the sign will appear in a tooltip in the Web browser window. The table should display the images by retrieving their information from the `db_RoadSigns.mdb` database. This page also will contain two `TextBoxes` and a `Button` that allow users to provide their information to register for a driving test. When users click the **Register** `Button`, the second page (`RoadTestRegistered`) displays confirmation information that the user has registered for a driving test. For this exercise, you need only organize the components (`RoadSigns` and `RoadTestRegistered` ASPX pages, `db_RoadSigns.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs, one of which will contain a `Table` control for the various road signs. The middle tier should contain the code used to retrieve the road sign names from the database. The information tier should contain the `db_RoadSigns.mdb` database where the road-signs information is stored.

TUTORIAL 29

Bookstore Application: Client Tier

Introducing Web Controls
Solutions

Instructor's Manual Exercise Solutions Tutorial 29

MULTIPLE-CHOICE QUESTIONS

29.1 You change the _____ property of the ASPX page to specify the color that displays in the background of the page.

a) `BackColor` b) `bgColor`
c) `BackgroundColor` d) `Color`

29.2 `Button`, `Label` and `Table` controls for ASPX pages can be accessed from the _____ tab.

a) **Web Forms** b) **Components**
c) **Data** d) Both a and b.

29.3 The _____ attribute is used to specify the position of a Web control on an ASPX page.

a) `position` b) `location`
c) `style` d) `coordinate`

29.4 Unlike the Windows `Form` Designer, the Web Form Designer _____.

a) does not provide two viewing modes
b) provides two viewing modes
c) allows you to design the graphical user interface
d) does not allow you to design the user interface

29.5 The `BorderStyle` property of the `Image` control _____.

a) specifies the color of the border
b) specifies the type of border that displays around the `Image` control
c) specifies the width of the border d) Both a and b.

29.6 Setting the `BorderStyle` property to `OutSet` makes a control appear _____.

a) raised b) with a bold border
c) with the specified border width d) with the specified border color

29.7 Every _____ of a `Table` Web control can contain one or more _____.

a) `TableRow`; `TableColumns` b) `TableColumn`; `TableRows`
c) `TableRow`; `TableCells` d) `TableCell`; `TableRows`

29.8 For you to be able to create an **ASP**.NET **Web** application project, _____ must be running.

a) IIS b) Microsoft Access
c) Microsoft Word d) Internet Explorer

29.9 The _____ mode allows you to create the ASPX page's GUI by dragging and dropping controls on the page.

a) **HTML** b) **Design**
c) **Visual** d) **GUI**

29.10 To specify the position of a Web control, set the _____ and _____ values of the _____ attribute.

a) `X`, `Y`, `style` b) `X`, `Y`, `position`
c) `TOP`, `LEFT`, `style` d) `TOP`, `LEFT`, `position`

Answers: 29.1) b. 29.2) a. 29.3) c. 29.4) b. 29.5) b. 29.6) a. 29.7) c. 29.8) a. 29.9) b. 29.10) c.

EXERCISES

[Note: In these exercises, we may ask you to set an ASPX page as the application's start page, meaning that this page will appear first when the application is run. You can set an ASPX page as the start page by right clicking the file in the **Solution Explorer** *and selecting* **Set As Start Page***.]*

29.11 (*Phone Book Application: GUI*) Create the user interface for the **Phone Book** application. The design for the two pages for this application is displayed in Fig. 29.25.

a) ***Creating an ASP.NET Web application***. Create an ASP.NET Web application project, and name it `PhoneBook`. Rename the ASPX page to `PhoneBook.aspx`, and set `Option Strict` to `On`. Set `PhoneBook.aspx` as the start page.

b) ***Changing the background color***. Change the background color of your ASPX page (`PhoneBook.aspx`) to the light-yellow **Web Palette** color (located in the sixth column of the 12th row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` of the ASPX page to `Phone Book`.

c) ***Adding a `Label`***. Create a `Label`, set the font size to `X-Large` and change the `Text` property to `Phone Book Web Application`. Set the `LEFT:` portion of the `style` attribute value to `40px` and the `TOP:` portion to `17px`. Name the control `lblPhoneBook`.

d) ***Adding another `Label`***. Create another `Label`, and set the `Text` property to `Select a name from the list and click the Get Number Button:`. Set the `LEFT:` portion of the `style` attribute value to `30px` and the `TOP:` portion to `65px`. Name this Web control `lblInstructions`.

e) ***Adding a `DropDownList` Web control***. Create a `DropDownList` Web control by dragging and dropping it from the **Toolbox** onto the ASPX page. The `DropDownList` Web control looks similar to the `ComboBox` Windows `Form` control. Set the width to `190px`, and set the `LEFT:` portion of the `style` attribute value to `134px` and the `TOP:` portion to `108px`. Name the `DropDownList` `cboNames`.

f) ***Adding a `Button`***. Create a `Button`, set its width to `90px` and change the `Text` property to `Get Number`. Set the `LEFT:` portion of the `style` attribute value to `175px` and the `TOP:` portion to `200px`. Name the Web control `btnGet`.

g) ***Adding another ASPX page to the Phone Book application***. Add another ASPX page to the **Phone Book** application, name it `PhoneNumber.aspx` and change the background to the light-yellow color. Change the `title` property to `Phone Number`.

h) ***Adding a `Label` to the `PhoneNumber.aspx`***. Create a `Label` and name it `lblPhoneNumber`. Set the font size to `X-Large` and change the `Text` property to `Phone Number:`. Set the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `15px`.

i) ***Adding another `Label`***. Create another `Label`, set its `BorderStyle` to `Inset`, and set its height and width to `50px` and `380px`, respectively. Clear the text of the `Label`. Name the `Label` `lblNumbers`, and set the `LEFT:` portion of the `style` attribute value to `25px` and the `TOP:` portion to `80px`.

j) ***Adding a `Button` to the `PhoneNumber.aspx` page***. Create a `Button`, set its width to `115px` and change the `Text` property to `Phone Book`. Set the `LEFT:` portion of the `style` attribute value to `135px` and the `TOP:` portion to `150px`. Name the `Button` `btnPhoneBook`.

k) ***Saving the solution file***. Save the solution file to the `PhoneBook` folder located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

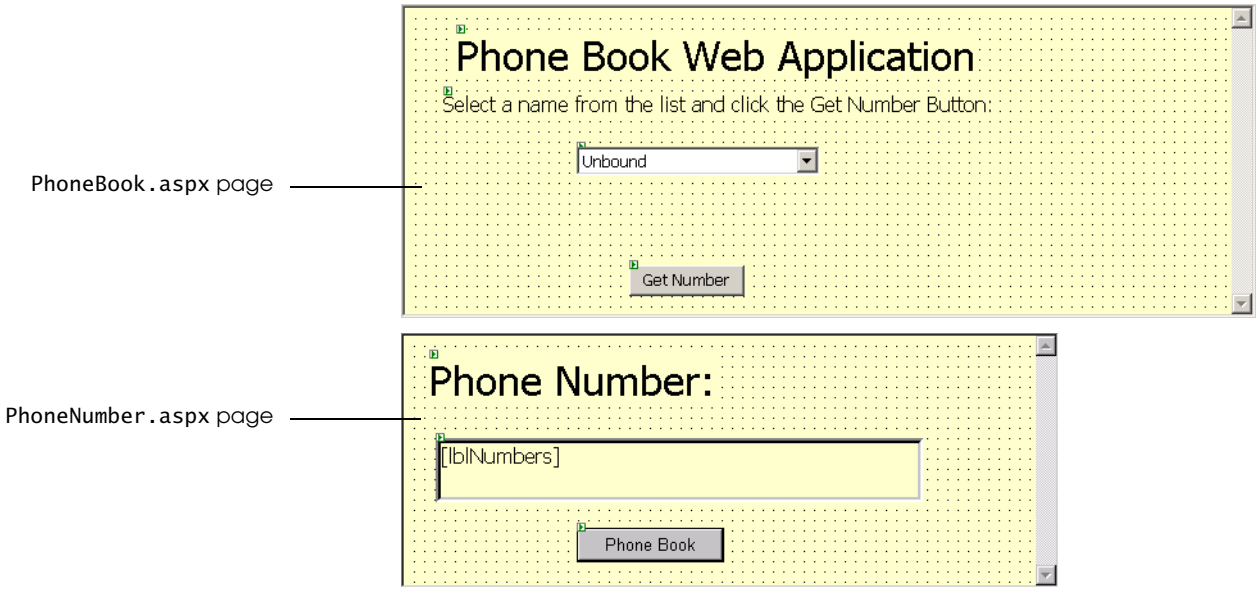


Figure 29.25 **Phone Book** application ASPX pages' design.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

29.12 (*US State Facts Application: GUI*) Create the user interface for the **US State Facts** application. The design for the two pages of this application is displayed in Fig. 29.26.

a) ***Creating an ASP.NET Web application***. Create a new ASP.NET Web application project, and name it `USStateFacts`. Rename the first ASPX page to `States.aspx`, and set `Option Strict` to `On`. Set `States.aspx` as the start page.

b) ***Changing the background color***. Change the background color of the `States.aspx` page to the light-blue **Web Palette** color (located in the sixth column of the second row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` property of the ASPX page to `States`.

c) ***Adding a `Label` to `States.aspx`***. Create a `Label` Web control, and place it on the page. Set the font size to `XX-Large`, and change the `Text` property to `States`. Change the `LEFT:` portion of its `style` attribute value to `390px`, and set the `TOP:` portion to `15px`. Name the Web control `lblStates`.

d) ***Adding a Horizontal Rule to `States.aspx`***. Create a Horizontal Rule, place it on the ASPX page and set its `width` to `150%`. When setting its position, change the `TOP:` value to `80px`, set the `LEFT:` value to `0px` and specify the `Height:` as `4px`. Name the Horizontal Rule `hrzStates`.

e) ***Adding another `Label` to `States.aspx`***. Create another `Label`, and place it beneath the Horizontal Rule. Change the font size to `Medium`, and set the `Text` property to `Select a state from the list and click the button to view facts about that state:`. Set its height to `16px` and its width to `620px`. Change the `LEFT:` portion of its `style` attribute value to `195px`, and set the `TOP:` portion to `100px`. Name this Web control `lblInstructions`.

f) ***Adding a `ListBox` to `States.aspx`***. Create a `ListBox`, and place it on the ASPX page. Set its `Height` property to `100px` and its `Width` property to `155px`. Set the `LEFT:` portion of the `style` attribute value to `365px` and the `TOP:` portion to `150px`. Name the `ListBox` `lstStates`.

g) ***Adding a `Button` to `States.aspx`***. Create a `Button`, and place it on the page. Set its `Text` property to `Review Facts` and its `Width` property to `130px`. Change the `LEFT:` portion of the `style` attribute value to `375px` and the `TOP:` portion to `270px`. Name the `Button` `btnFacts`.

h) ***Adding another ASPX page to the US State Facts application***. Add another ASPX page to the **US State Facts** application, name it `StateFacts.aspx` and change the background color to light blue.

i) ***Adding a `Label` to `StateFacts.aspx`***. Create a `Label`, name it `lblStateName`, set its font size to `XX-Large` and change its `ForeColor` property to `Blue`. Clear the `Label`'s text. Set its position by setting the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `15px`.

j) ***Adding a Horizontal Rule***. Place the Horizontal Rule beneath the `Label` and set its `TOP:` position to `90px`, its `LEFT:` position to `0px` and its `Height:` to `4px`. Change the `width` to `150%`. Name the Horizontal Rule `hrzStateFacts`.

k) ***Adding an `Image` control to `StateFacts.aspx`***. Create an `Image` control and set its `BorderStyle` to `Outset`. Change the `BorderWidth` to `5px`. Set its height to `200px` and its width to `300px`. Set the position of the `Image` by changing the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `110px`. Name the Web control `imgFlag`.

l) ***Adding a `Table` to `StateFacts.aspx`***. Create a `Table` with four rows and two columns. Set the `BorderStyle` to `Outset`, the `BorderWidth` to `5px` and `GridLines` to `Both`. Set the height and width of each `TableCell` of the first column to `70px` and `200px`, respectively, and set the `Font` property's `Size` to `Large`. Set the `Text` property of the cells in the first column to `Capital:`, `Flower:`, `Tree:` and `Bird:`, respectively. Change the `LEFT:` portion of the `style` attribute value to `335px` and the `TOP:` portion to `110px`. Name the `Table` control `tblState`.

m) ***Adding a `Button` to `StateFacts.aspx`***. Create a `Button`, change its text to `State List`, and change the `LEFT:` portion of the `style` attribute value to `285px` and the `TOP:` portion to `425px`. Name the `Button` control `btnStateList`.

n) ***Saving the solution file***. Save the solution file to the `USStateFacts` folder located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

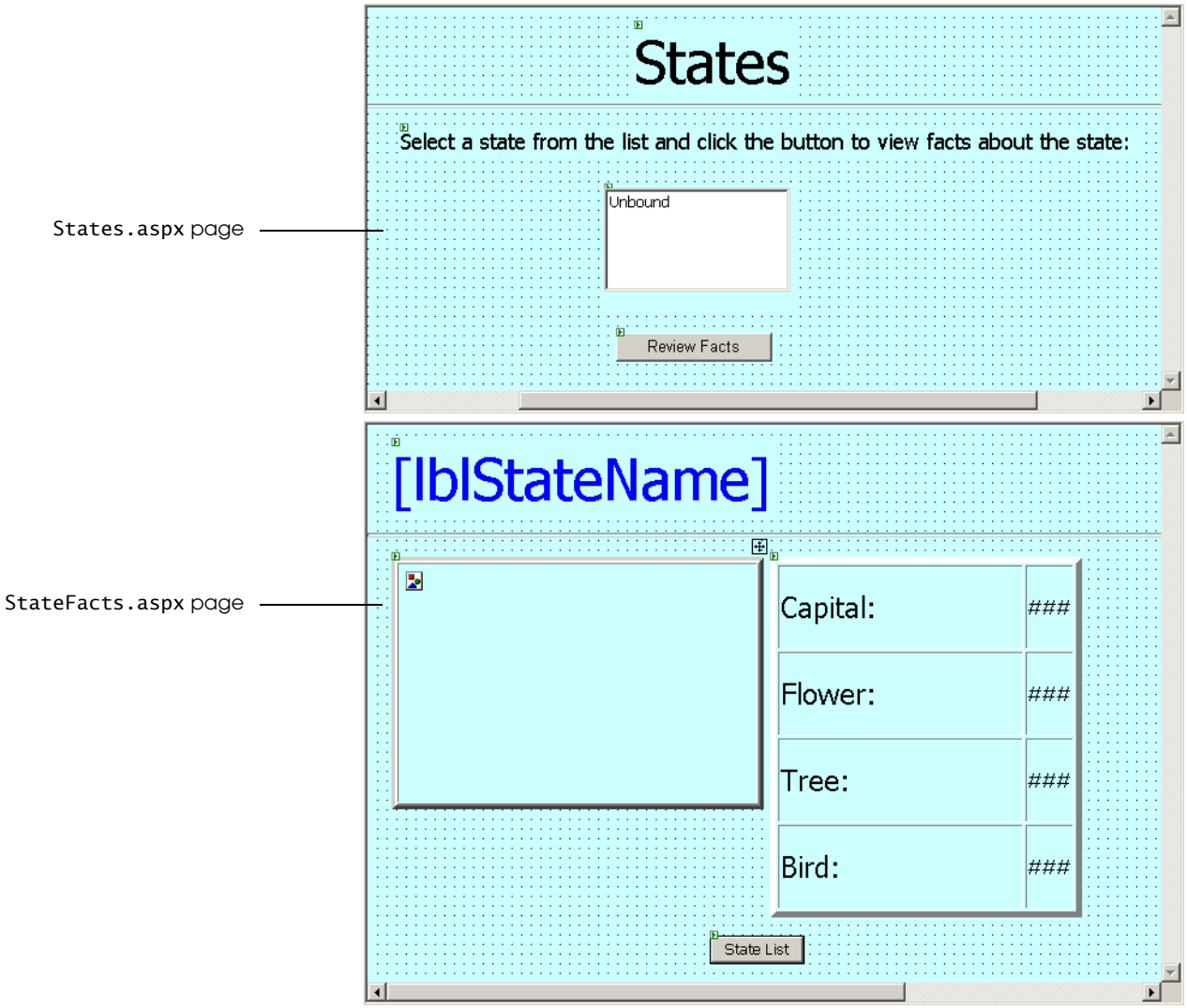


Figure 29.26 **US State Facts** application ASPX pages' design.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

29.13 (*Road Sign Review Application: GUI*) Create the user interface for the **Road Sign Review** application. The design for the two pages of this application is displayed in Fig. 29.27.

a) ***Creating an ASP.NET Web application***. Create a new ASP.NET Web application project, and name it **RoadSignReview**. Change the name of the existing ASPX page to `RoadSigns.aspx`, and set `Option Strict` to `On`. Set `RoadSigns.aspx` as the start page.

b) ***Changing the background color***. Change the background color of `RoadSigns.aspx` to the light-green **Web Palette** color (located in the sixth column of the 14th row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` of the ASPX page to `RoadSigns`.

c) ***Adding a `Label` to `RoadSigns.aspx`***. Create a `Label`, and set its font size to `XX-Large`. Change the `Text` property to `Road Signs`. Set its position by changing the `style` attribute value's `LEFT:` portion to `295px` and the `TOP:` portion to `16px`. Name the `Label` control `lblRoadSigns`.

d) ***Adding a Horizontal Rule to `RoadSigns.aspx`***. Create a Horizontal Rule. Set its `width` to `150%`, and set the `TOP:` position to `80px`, the `LEFT:` position to `0px` and the height to `4px`. Name the Horizontal Rule `hrzRoadSigns`.

e) ***Adding a `Table` to `RoadSigns.aspx`***. Create a `Table` with three rows and five columns. Set the `BorderStyle` to `Outset`, the `BorderWidth` to `5px` and the `GridLines` property to `Both`. Also, set the `Table`'s `Height` property to `279px` and `Width` property to `626px`. Set each row's height to `50px` and each `TableCell`'s width to `20px`. Change the `style` attribute value by setting `LEFT:` to `70px` and `TOP:` to `150px`. Name the `Table` control `tblRoadSigns`.

f) ***Adding a `Label` to `RoadSigns.aspx`***. Create a `Label`, and set its font size to `Large`. Change the `Text` property to `Register for Your Driving Test`. Set its position by changing the `style` attribute value's `LEFT:` portion to `70px` and `TOP:` portion to `470px`. Name the Web control `lblRegister`.

g) ***Adding a `Label` and `TextBox` to `RoadSigns.aspx`***. Create a `Label` and set its text to `Name:`. Set its font size to `Medium`, and change its position to `LEFT: 70px` and `TOP: 520px`. Name the `Label` control `lblName`. Create a `TextBox`, and place it next to the **Name:** `Label`. Set its height to `20px` and width to `115px`. Change the position to `LEFT: 135px` and `TOP: 520px`. Name the `TextBox` control `txtName`.

h) ***Adding another `Label` and `TextBox` pair to `RoadSigns.aspx`***. Create a `Label` and set its text to `Phone Number:`. Set its font size to `Medium`, and change its position to `LEFT: 275px` and `TOP: 520px`. Name the `Label` control `lblPhoneNumber`. Create a `TextBox`, and place it next to the **Phone Number:** `Label`. Set its height to `20px` and width to `115px`. Change its position to `LEFT: 410px` and `TOP: 520px`. Name the `TextBox` control `txtPhoneNumber`.

i) ***Adding a `Button` to `RoadSigns.aspx`***. Create a `Button`, set its `Text` to `Register`, and change its height and width to `30px` and `120px`, respectively. Change the position of the `Button` by setting the `LEFT:` portion of the style attribute value to `555px` and the `TOP:` portion to `520px`. Name the `Button` control `btnRegister`.

j) ***Adding another ASPX page to the Road Sign Review application***. Add another ASPX page to the application, name it `RoadTestRegistered.aspx` and change the background color to light green.

k) ***Adding a `Label` to `RoadTestRegistered.aspx`***. Create a `Label`, setting its font size to `XX-Large` and its `Text` property to `Registration Complete`. Change its position by setting the `LEFT:` portion of its `style` attribute value to `200px` and the `TOP:` portion to `15px`. Name the `Label` control `lblRegistration`.

l) ***Adding a Horizontal Rule to `RoadTestRegistered.aspx`***. Create a Horizontal Rule. Set its `width` to `150%`, the `TOP:` position to `80px`, the `LEFT:` position to `0px` and the height to `4px`. Name the Horizontal Rule `hrzRoadTestRegistered`.

m) ***Adding another `Label` to `RoadTestRegistered.aspx`***. Create a `Label`, name it `lblConfirmation` and set its font size to `Medium`. Delete the `Text` property value, and leave it blank. Change its position by setting the `LEFT:` portion of its `style` attribute value to `125px` and the `TOP:` portion to `130px`.

n) ***Saving the solution file***. Save the solution file to the `RoadSignReview` folder located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

Figure 29.27 **Road Signs** application ASPX pages' design.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

TUTORIAL 30

Bookstore Application: Information Tier

Examining the Database and Creating Database Components

Solutions

Instructor's Manual Exercise Solutions Tutorial 30

MULTIPLE-CHOICE QUESTIONS

30.1 _____ is an example of a database product.

a) Microsoft Access
b) Microsoft SQL Server
c) Oracle
d) All of the above.

30.2 An advantage of using information in a database is that _____.

a) the data can be updated in real time
b) information that changes need be updated only in one location
c) Both a and b.
d) None of the above.

30.3 When a funnel appears to the right of a field's `CheckBox` in the **Query Builder** dialog, it indicates that _____.

a) information will be updated in the specified field
b) a value will be retrieved from that field according to specified criteria
c) the field will not be included in the SQL statement
d) None of the above.

30.4 The `Parameters` property of _____ contains a collection of parameters.

a) `OleDbConnection`
b) `OleDbDataConnection`
c) `OleDbDataCommand`
d) `OleDbCommand`

30.5 The _____ can be used to create an `OleDbConnection`.

a) **Server Explorer** window
b) **Query Builder** tool
c) Both a and b.
d) None of the above.

30.6 You use the _____ object to create SQL statements for retrieving data from a database.

a) `OleDbConnection`
b) `OleDbDataReader`
c) `OleDbCommand`
d) None of the above.

30.7 The _____ is used when creating SQL statements visually for the `OleDbCommand` object's `CommandText` property.

a) **Server Explorer** window
b) **Query Builder** tool
c) Both a and b.
d) None of the above.

30.8 You use the _____ object to open a connection to the database.

a) `OleDbConnection`
b) `OleDbDataReader`
c) `OleDbCommand`
d) None of the above.

30.9 You use the _____ property of the `OleDbCommand` object to specify values for information that is not known in advance.

a) `Connection`
b) `Parameters`
c) `Field`
d) `Name`

30.10 Another name for the database tier is _____.

a) the information tier
b) the bottom tier
c) Both a and b.
d) None of the above.

Answers: 30.1) d. 30.2) c. 30.3) b. 30.4) d. 30.5) a. 30.6) c. 30.7) b. 30.8) a. 30.9) b. 30.10) c.

EXERCISES

30.11 (*Phone Book Application: Database*) Create the database connections and data command objects for the **PhoneBook** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **PhoneBook** application that you created in Tutorial 29.

b) ***Copying the `db_Phone.mdb` database to the `Databases` folder***. Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_Phone.mdb` database to the `Databases` folder in IIS's `wwwroot` folder.

c) ***Using Server Explorer to add a connection to the database***. In the **Server Explorer** window, add a connection to the `db_Phone.mdb` database. Drag and drop the connection object onto the `PhoneBook.aspx` page. Name the connection object `objOleDbConnection`.

d) ***Using Query Builder for the `PhoneBook.aspx` page.*** Add an `OleDbCommand` to the `PhoneBook.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This command should retrieve all the names of the people from the database. Name this command object `objSelectNames`.

e) ***Adding a connection to the database to the `PhoneNumber.aspx` page***. Using the **`Server Explorer`** window, drag and drop a database connection object onto the `PhoneNumber.aspx` page. Name this connection object `objOleDbConnection`.

f) ***Using Query Builder for `PhoneBook.aspx`***. Add an `OleDbCommand` to the `PhoneNumber.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This configuration should retrieve the phone number of the person whose name will be selected, from the `DropDownList` in the `PhoneBook.aspx` page, by the user. You need to set the criteria to specify which person's phone number will be retrieved from the database. Name this command object `objSelectPhoneNumber`.

g) ***Saving the solution.*** Select **File > Save All** to save the solution's files.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

30.12 (*US State Facts Application: Database*) Create the database connections and data command objects for the **USStateFacts** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **USStateFacts** application that you created in Tutorial 29.

b) ***Copying the `db_StateFacts.mdb` database to the `Databases` folder***. Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_StateFacts.mdb` database to the `Databases` folder in IIS's `wwwroot` folder.

c) ***Using Server Explorer to add a connection to the database***. In the **Server Explorer** window, add a connection to the `db_StateFacts.mdb` database. Drag-and-drop the connection object onto the `States.aspx` page. Name this connection object `objOleDbConnection`.

d) ***Using Query Builder for the `States.aspx` page.*** Add an `OleDbCommand` to the `States.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This command should retrieve the names of the states from the **name** field of the `states` table in the database. Name this command object `objSelectNames`.

e) ***Adding a connection to the database to the `StateFacts.aspx` page***. Using the **Server Explorer** window, drag-and-drop a database connection object onto the database on the `StateFacts.aspx` page. Name this connection object `objOleDbConnection`.

f) ***Using Query Builder for `StateFacts.aspx`***. Add an `OleDbCommand` to the `StateFacts.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText`

property of the `OleDbCommand`. This configuration should retrieve all the information, from the `states` table of the database, about the state that is selected by the user. You need to set the criteria to specify which state's information will be retrieved from the database. Name this command object `objSelectStateInformation`.

g) ***Saving the solution.*** Select **File > Save All** to save the solution's files.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

30.13 (*Road Sign Review Application: Database*) Create the database connections and data command objects for the **RoadSignReview** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **RoadSignReview** application that you created in Tutorial 29.

b) ***Copying the `db_RoadSigns.mdb` database to the `Databases` folder***. Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_RoadSigns.mdb` database to the `Databases` folder in IIS's `wwwroot` folder.

c) ***Using Server Explorer to add a connection to the database***. In the **Server Explorer** window add a connection to the `db_RoadSigns.mdb` database. Drag and drop the connection object onto the `RoadSigns.aspx` page. Name this command object `objOleDbConnection`.

d) ***Using Query Builder for the `RoadSigns.aspx` page***. Add an `OleDbCommand` to the `RoadSigns.aspx` page. Set the `Connection` property to the `OleDbConnection` object that you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This configuration should retrieve all the information about all the road signs from the `signs` table of the database. You will not need to specify a criterion for this exercise, because all the information from the database needs to be retrieved. Name this command object `objSelectSignInformation`.

e) ***Saving the solution.*** Select **File > Save All** to save the solution's files.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

TUTORIAL 31

Bookstore Application: Middle Tier

Introducing Code-Behind Files
Solutions

Instructor's Manual Exercise Solutions Tutorial 31

MULTIPLE-CHOICE QUESTIONS

31.1 The `Page_Load` event handler _____.

a) redirects the client browser to different Web pages
b) defines the functionality when a `Button` is clicked
c) executes any processing necessary to display a Web page
d) defines the functionality when a Web control is selected

31.2 The `Response.Redirect` method _____.

a) refreshes the current Web page
b) sends the client browser to a specified Web page
c) responds to user input
d) responds to the click of a `Button`

31.3 `Session` items are used in the **Bookstore** application because _____.

a) variables in ASP.NET Web applications must be created as `Session` items
b) values need to be shared among Web pages
c) `Session` items are simpler to create than instance variables
d) Both a and b.

31.4 Session state is used for _____ in ASP.NET.

a) tracking user-specific data
b) running an application
c) using a database
d) None of the above.

31.5 The file extension for an ASPX code-behind file is _____.

a) `.asp`
b) `.aspx`
c) `.aspx.vb`
d) `.code`

31.6 The `Response` object is a predefined ASP.NET object that _____.

a) connects to a database
b) retrieves information from a database
c) creates Web controls
d) provides methods for responding to client requests

31.7 The `Response.Redirect` method takes a(n) _____ as an argument.

a) URL
b) `Integer` value
c) `Boolean` value
d) `OleDbConnection` object

31.8 The _____ property specifies the image that an `Image` control displays.

a) `ImageGIF`
b) `ImageURL`
c) `Image`
d) `Display`

31.9 The Visual Basic .NET file that contains the ASPX page's corresponding class is called the _____.

a) ASPX file
b) code-behind file
c) class file
d) None of the above.

31.10 Information can be maintained across Web pages by adding a _____ to the `Session` object.

a) key-value pair
b) number
c) database connection object
d) None of the above.

Answers: 31.1) c. 31.2) b. 31.3) b. 31.4) a. 31.5) c. 31.6) d. 31.7) a. 31.8) b. 31.9) b. 31.10) a.

EXERCISES

31.11 (*Phone Book Application: Functionality*) Define the middle tier for the Phone Book application.

a) ***Opening the application***. Open the PhoneBook application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Importing `System.Data.OleDb` in `PhoneBook.aspx.vb`.*** Import the `System.Data.OleDb` namespace in `PhoneBook.aspx.vb`.

c) ***Defining the `Page_Load` event handler of `PhoneBook.aspx` page***. Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

d) ***Populating the `DropDownList` with names***. Add a `Do While...Loop` to `PhoneBook.aspx`'s `Page_Load` method. This loop should add to the `DropDownList` each person's name read by the data reader.

e) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

f) ***Creating the Get Number `Button`'s `Click` event handler for the `PhoneBook.aspx` page***. Double click the Get Number `Button` to create the `Click` event's event handler.

g) ***Creating a `Session` item.*** In the `Click` event handler, create a `Session` item to store the selected name.

h) ***Redirecting to the `PhoneNumber.aspx` page.*** In the `Click` event handler, use the `Response.Redirect` method to redirect the client browser to the `PhoneNumber.aspx` page.

i) ***Importing `System.Data.OleDb` in `PhoneNumber.aspx.vb`.*** Import the `System.Data.OleDb` namespace in `PhoneNumber.aspx.vb`.

j) ***Defining the `Page_Load` event handler for the `PhoneNumber.aspx` page***. Use the `Open` method to open the connection to the database. Access the `Session` item to retrieve the selected name. Specify this name as the parameter value for the `OleDbCommand` object. Create a data reader to read the information specified by the data command object.

k) ***Displaying the selected name and phone number.*** In the `Page_Load` event handler, read the desired phone number from the data reader. Display the selected name and corresponding phone number in the `lblNumbers` `Label`.

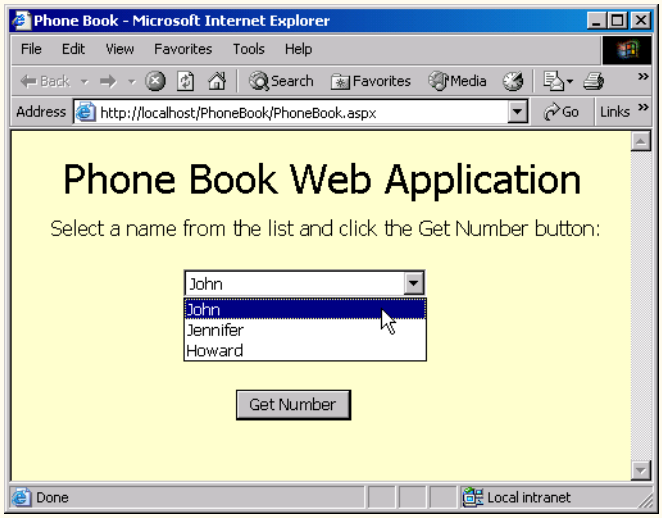
l) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

m) ***Creating the Phone Book `Button`'s `Click` event handler for the `PhoneNumber.aspx` page***. Double click the Phone Book `Button` to create the `Click` event's event handler.

n) ***Redirecting to the `PhoneBook.aspx` page.*** In the `Click` event handler, use the `Response.Redirect` method to redirect the client browser to the `PhoneBook.aspx` page.

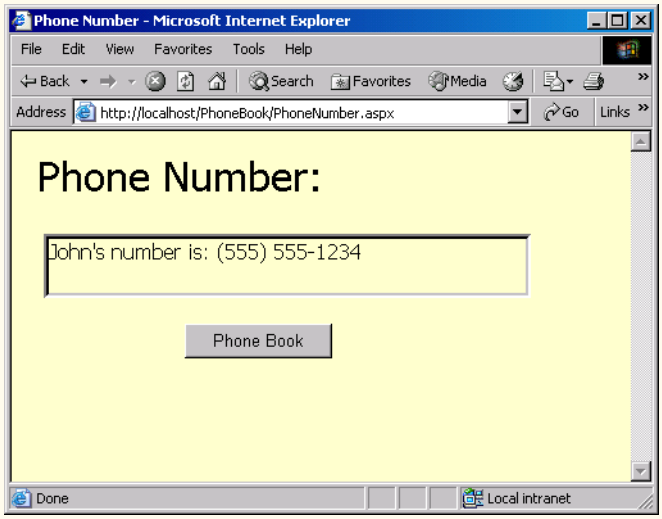
```
' Solution 31.11 Solution
' PhoneBook.aspx.vb

Imports System.Data.OleDb

Public Class PhoneBook
   Inherits System.Web.UI.Page

   ' control declarations

   ' Web Form Designer Generated Code

   ' invoked when page is loaded
   Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load
```

```

      objOleDbConnection.Open() ' open connection to the database

      ' declare reader to read from database
      Dim objReader As OleDbDataReader

      ' create reader to read from database
      objReader = objSelectNames.ExecuteReader()

      ' add names to DropDownList
      Do While objReader.Read()

         ' add item to DropDownList
         cboNames.Items.Add(Convert.ToString(objReader("Name")))
      Loop

      objReader.Close() ' close the reader

      ' close the connection to the database
      objOleDbConnection.Close()
   End Sub ' Page_Load

   ' invoke when btnGet Button is clicked
   Private Sub btnGet_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnGet.Click

      Session("Name") = Convert.ToString(cboNames.SelectedItem)

      ' redirect to another ASPX page
      Response.Redirect("PhoneNumber.aspx")
   End Sub ' btnGet_Click

End Class ' PhoneBook
```




```
' Exercise 31.11 Solution
' PhoneNumber.aspx.vb

Imports System.Data.OleDb

Public Class PhoneNumber
```

```
    Inherits System.Web.UI.Page

    ' control declarations

    ' Web Form Designer Generated Code

    ' invoked when page is loaded
    Private Sub Page_Load(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles MyBase.Load

       ' represents the name
       Dim strName As String = Convert.ToString(Session("Name"))

       objOleDbConnection.Open() ' open connection to the database

       ' specify name to retrieve phone number for
       objSelectPhoneNumber.Parameters( _
          "Name").Value = strName

       ' declare reader to read from the database
       Dim objReader As OleDbDataReader

       ' create reader to read from the database
       objReader = objSelectPhoneNumber.ExecuteReader()

       objReader.Read() ' start data reader

       ' display name and number in Label
       lblNumbers.Text = strName & "'s number is: " & _
          Convert.ToString(objReader("Phone_Numbers"))

       objReader.Close() ' close data reader

       ' close the connection to the database
       objOleDbConnection.Close()
    End Sub ' Page_Load

    ' redirects user back to PhoneBook.aspx
    Private Sub btnPhoneBook_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles btnPhoneBook.Click

       Response.Redirect("PhoneBook.aspx")
    End Sub ' btnPhoneBook_Click

End Class ' PhoneNumber
```



31.12 (*US State Facts Application: Functionality*) Define the middle tier for the **US State Facts** application.

a) ***Opening the application***. Open the **USStateFacts** application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Copying the `FlagImages` folder to your project folder.*** Copy the `C:\Examples\Tutorial31\Exercises\Images\FlagImages` folder to the `USStateFacts` folder.

c) ***Importing `System.Data.OleDb` in `States.aspx.vb`.*** Import the `System.Data.OleDb` namespace in `States.aspx.vb` before the class definition.

d) ***Defining the `Page_Load` event handler for the `States.aspx` page.*** Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

e) ***Populating the `ListBox` with state names in the `States.aspx` page.*** Add a `Do While...Loop` to `States.aspx`'s `Page_Load` method. This loop should add to the `ListBox` the name of each state read by the data reader.

f) ***Creating a `Button`'s `Click` event handler for the `States.aspx` page.*** Double click the **Review Facts** `Button` to create the `Click` event's event handler.

g) ***Creating a `Session` item***. Create a `Session` item in the `Click` event handler and assign it to the state name that the user selects from the `ListBox`.

h) ***Redirecting to the `StateFacts.aspx` page.*** In the `Click` event handler, use the `Redirect.Response` method to redirect the client browser to the `StateFacts.aspx` page.

i) ***Importing `System.Data.OleDb` in `StateFacts.aspx.vb`.*** Import the `System.Data.OleDb` namespace in `StateFacts.aspx.vb`.

j) ***Defining the `Page_Load` event handler of `StateFacts.aspx` page.*** Use the `Open` method to open the connection to the database. Access the `Session` object to retrieve the selected state name. Specify this name as a parameter value for the `OleDbCommand` object. Create a data reader to read the information specified by the data command object.

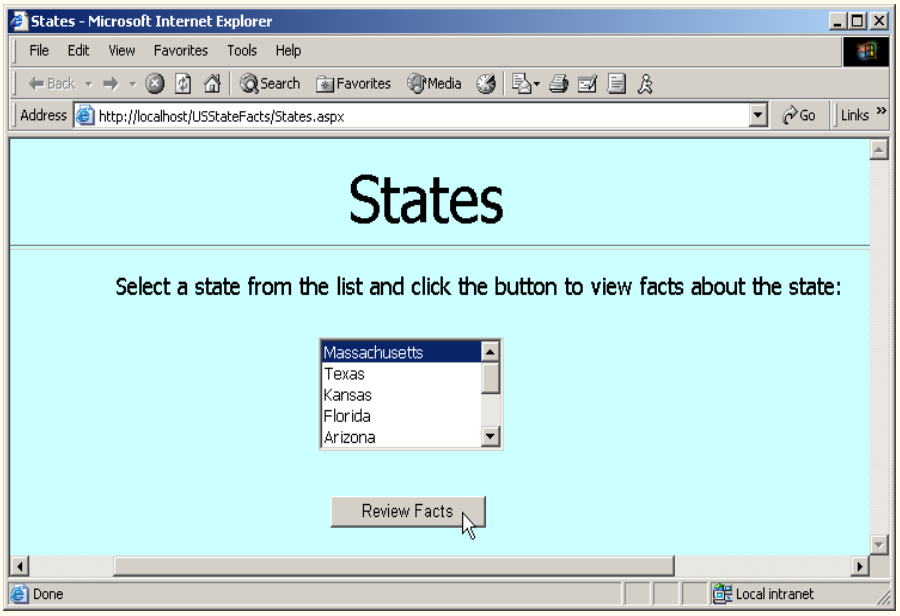
k) ***Displaying the state facts in the `Table`.*** In the `Page_Load` event handler, use the data reader to retrieve the desired state's facts. Display the selected state's name in the `lblStateName Label`. Set the `ImageURL` property of the `Image` control to the location of the selected state's flag image. Display the name of the state capital, flower, tree and bird in the `Table` on the `StateFacts.aspx` page.

l) ***Closing the connection.*** Close the connection to the database by invoking the `Close` method.

m) ***Creating the State List `Button`'s `Click` event handler for the `StateFacts.aspx` page.*** Double click the **State List** `Button` to create the `Click` event handler.

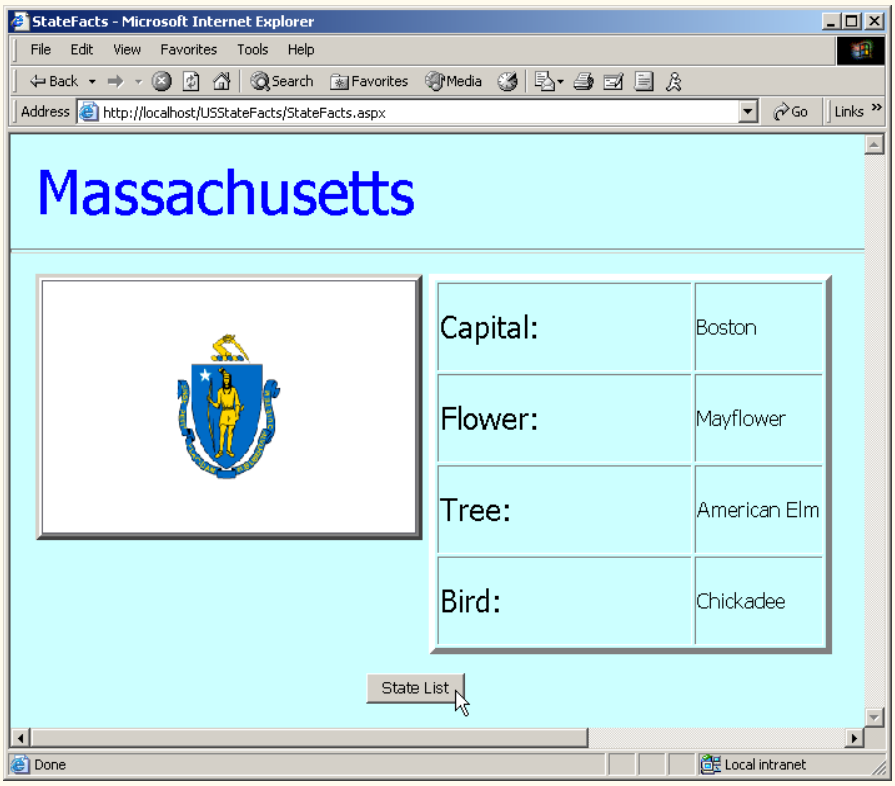
n) ***Redirecting to the `States.aspx` page.*** In the `Click` event handler use the `Redirect.Response` method to redirect the client browser to the `States.aspx` page.

```
' Exercise 31.12 Solution
' States.aspx.vb

Imports System.Data.OleDb

Public Class States
   Inherits System.Web.UI.Page

   ' control declarations

   ' Web Form Designer Generated Code

   ' invoked when page is loaded
   Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      objOleDbConnection.Open() ' open connection to the database

      ' declare reader to read from database
      Dim objReader As OleDbDataReader

      ' create reader to read from database
      objReader = objSelectNames.ExecuteReader()

      ' add names to the ListBox
      Do While objReader.Read()

         ' add item to ListBox
         lstStates.Items.Add(Convert.ToString(objReader("Name")))
      Loop

      objReader.Close() ' close the reader

      ' close the connection to the database
      objOleDbConnection.Close()
   End Sub ' Page_Load

   ' handle click event
   Private Sub btnFacts_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnFacts.Click

      ' create Session item named strName
      Session("strName") = lstStates.SelectedItem.Text

      ' redirect to another ASPX page
      Response.Redirect("StateFacts.aspx")
   End Sub ' btnFacts_Click

End Class ' States
```




```
' Solution 31.12 Solution
' StateFacts.aspx.vb

Imports System.Data.OleDb

Public Class StateFacts
   Inherits System.Web.UI.Page

   ' control declarations

   ' Web Form Designer Generated Code

   ' invoked when page is loaded
   Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      ' display name of selected book
      lblStateName.Text = Convert.ToString(Session("strName"))

      objOleDbConnection.Open() ' open connection to database

      ' specify state name to retrieve information about
      objSelectStateInformation.Parameters("Name").Value = _
         Convert.ToString(Session("strName"))

      ' declare database reader to read from database
      Dim objReader As OleDbDataReader

      ' create database reader to read from database
      objReader = objSelectStateInformation.ExecuteReader()

      ' while reader is reading database, retrieve data from
      ' specified positions and display them on page
      Do While objReader.Read()

         ' display flag for selected state
         imgFlag.ImageUrl = "FlagImages/" & _
```

```
            Convert.ToString(objReader("flag"))

         ' display information from database in Table
         tblState.Rows(0).Cells(1).Text = _
            Convert.ToString(objReader("capital"))
         tblState.Rows(1).Cells(1).Text = _
            Convert.ToString(objReader("flower"))
         tblState.Rows(2).Cells(1).Text = _
            Convert.ToString(objReader("tree"))
         tblState.Rows(3).Cells(1).Text = _
            Convert.ToString(objReader("bird"))
      Loop

      objOleDbConnection.Close() ' close connection to database
   End Sub ' Page_Load

   ' invoked when user clicks Button
   Private Sub btnStateList_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnStateList.Click

      ' redirects to States.aspx page
      Response.Redirect("States.aspx")
   End Sub ' btnStateList_Click

End Class ' StateFacts
```



31.13 (*Road Sign Review Application: Functionality*) Define the middle tier for the **Road Sign Review** application.

a) ***Opening the application***. Open the **RoadSignReview** application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Copying the `SignImages` folder to your project folder.*** Copy the `C:\Examples\Tutorial31\Exercises\Images\SignImages` folder to the `RoadSignReview` folder.

c) ***Importing `System.Data.OleDb` and `System.Web.UI.WebControls` to `RoadSigns.aspx.vb`.*** Import the `System.Data.OleDb` and `System.Web.UI.WebControls` namespaces to `RoadSigns.aspx.vb`. You need to import `System.Web.UI.WebControls` because you will be creating a Web control programmatically in this exercise.

d) ***Defining the `Page_Load` event handler for the `RoadSigns.aspx` page.*** Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

e) ***Populating the `Table` with sign images in the `RoadSigns.aspx` page.*** Add a `Do While...Loop` to `RoadSigns.aspx`'s `Page_Load` method. This loop should display an image of the sign and display the sign name in the `ToolTip` property. This property specifies the text that displays in a tooltip box when the mouse hovers over the `Image`. The sign image and name should be retrieved using the data reader. To display an `Image` in a cell of the `Table`, you need to create an `Image` control, specify a cell and use the cell's `Controls.Add` method to add an image to that cell. For example, to create an `Image` control programmatically, type `Dim imgImageName As Image = New Image()`. You then need to set the `ImageURL` property to the location of the desired image. To display an `Image` control in the first cell of the first row, you would write the line `Table.Rows(0).Cells(0).Controls.Add(imgImageName)`. Also, if you wish to specify text for a tooltip, you must set the cell's `ToolTip` property—for example, `Table.Rows(0).Cells(0).ToolTip = "This is a tooltip"`.

f) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

g) ***Creating the Register `Button`'s `Click` event handler for `RoadSigns.aspx`.*** Double click the **Register** `Button` of `RoadSigns.aspx` to create the `Click` event handler.

h) ***Creating `Session` item.*** Create two `Session` items in the `Click` event handler, and set the first one equal to the user input for the **Name:** `TextBox`. The second `Session` item should equal the user input for the **Phone Number:** `TextBox`.

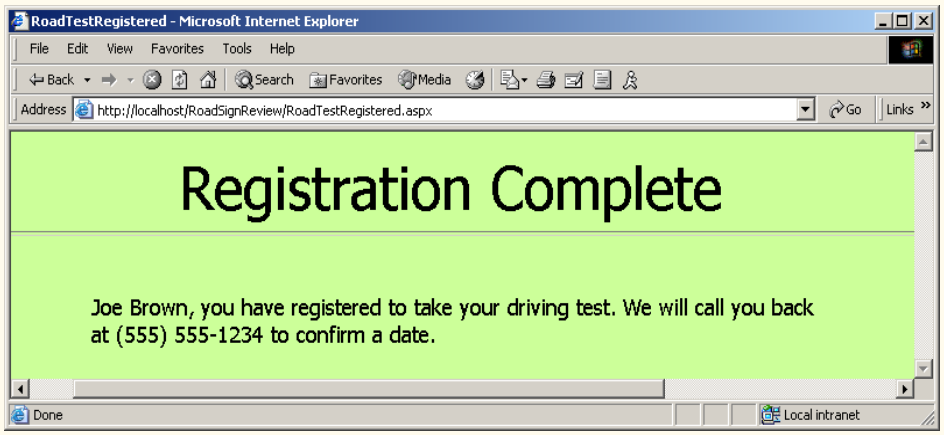
i) ***Redirecting to the `RoadTestRegistered.aspx` page.*** In the `Click` event handler, use the `Redirect.Response` method to redirect the client browser to the `RoadTestRegistered.aspx` page.

j) ***Defining the `Page_Load` method of `RoadTestRegistered.aspx` page.*** Use the `Session` items to display a confirmation to the user about the user's registration information. Display the confirmation using `Label lblConfirmation`. Display the user's name, and display text which states that the user will be contacted shortly at the phone number provided. This information should be displayed in a `Label`.

```
' Exercise 31.13 Solution
' RoadSigns.aspx.vb

Imports System.Data.OleDb
Imports System.Web.UI.WebControls

Public Class RoadSigns
   Inherits System.Web.UI.Page

   ' control declarations

   ' Web Form Designer Generated Code

   ' invoked when page is loaded
   Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      objOleDbConnection.Open() ' open connection to the database
```

```

        ' declare reader to read from database
        Dim objReader As OleDbDataReader

        ' create reader to read from database
        objReader = objSelectSignInformation.ExecuteReader()

        Dim intRow As Integer = 0
        Dim intColumn As Integer = 0

        ' display signs in Table
        Do While objReader.Read()

           ' move to next row
           If intColumn > 4 Then
              intRow += 1
              intColumn = 0
           End If

           ' create new Image control
           Dim imgCellImage As Image = New Image()

           ' set ImageURL property
           imgCellImage.ImageUrl = “SignImages/” & _
              Convert.ToString(objReader("sign"))

           ' add image to table
           tblRoadSigns.Rows((intRow)).Cells( _
              intColumn).Controls.Add(imgCellImage)

           ' add image name to ToolTip
           tblRoadSigns.Rows(intRow).Cells(intColumn).ToolTip = _
              Convert.ToString(objReader("name"))

           intColumn += 1 ' increment column location
        Loop

        objReader.Close() ' close the reader

        ' close the connection to the database
        objOleDbConnection.Close()
     End Sub ' Page_Load

     ' handles click event for btnRegister Button
     Private Sub btnRegister_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnRegister.Click

        ' create Session item
        Session("strName") = txtName.Text
        Session("strPhoneNumber") = txtPhoneNumber.Text

        ' redirect to another ASPX page
        Response.Redirect("RoadTestRegistered.aspx")
     End Sub ' btnRegister_Click

  End Class ' RoadSigns
```


```
' Exercise 31.13 Solution
' RoadTestRegistered.aspx.vb

Public Class RoadTestRegistered
   Inherits System.Web.UI.Page

   ' control declarations

   ' Web Form Designer Generated Code

   ' invoked when page is loaded
   Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

      ' display output
      lblConfirmation.Text = Convert.ToString(Session("strName")) & _
         ", you have registered to take your driving test. " & _
         "We will call you back at " & _
         Convert.ToString(Session("strPhoneNumber")) & _
         " to confirm a date."
   End Sub ' Page_Load

End Class ' RoadTestRegistered
```



32 TUTORIAL

Enhanced Car Payment Calculator Application

Introducing Exception Handling
Solutions

Instructor's Manual Exercise Solutions Tutorial 32

32.1 Dealing with exceptional situations as an application executes is called _____.

a) exception detection
b) exception handling
c) exception resolution
d) exception debugging

32.2 A(n) _____ is always followed by at least one `Catch` block or a `Finally` block.

a) `if` statement
b) event handler
c) `Try` block
d) None of the above.

32.3 The method call `Convert.ToInt32( "123.4a" )` will throw a(n) _____.

a) `FormatException`
b) `ParsingException`
c) `DivideByZeroException`
d) None of the above.

32.4 If no exceptions are thrown in a `Try` block, _____.

a) the `Catch` block(s) are skipped
b) all `Catch` blocks are executed
c) an error occurs
d) the default exception is thrown

32.5 A(n) _____ is an exception that does not have an exception handler, and therefore might cause the application to terminate execution.

a) uncaught block
b) uncaught exception
c) error handler
d) thrower

32.6 A `Try` block can have _____ associated with it.

a) only one `Catch` block
b) several `Finally` blocks
c) one or more `Catch` blocks
d) None of the above.

32.7 The _____ statement is used to rethrow an exception from inside a `Catch` block.

a) `Rethrow`
b) `Throw`
c) `Try`
d) `Catch`

32.8 _____ marks the end of a `Try` block and its corresponding `Catch` and `Finally` blocks.

a) `End Try`
b) `End Finally`
c) `End Catch`
d) `End Exception`

32.9 A `Finally` block is located _____.

a) after the `Try` block, but before each `Catch` block
b) before the `Try` block
c) after the `Try` block and the `Try` block's corresponding `Catch` blocks
d) Either (b) or (c).

32.10 A _____ is executed if an exception is thrown from a `Try` block or if no exception is thrown.

a) `Catch` block
b) `Finally` block
c) exception handler
d) All of the above.

Answers: 32.1) b. 32.2) c. 32.3) a. 32.4) a. 32.5) b. 32.6) c. 32.7) b. 32.8) a. 32.9) c. 32.10) b.

EXERCISES

32.11 (***Enhanced Miles Per Gallon Application***) Modify the **Miles Per Gallon** application (Exercise 13.13) to use exception handling to process the `FormatExceptions` that occur when converting the strings in the `TextBoxes` to `Doubles` (Fig. 32.16). The original application allowed the user to input the number of miles driven and the number of gallons used for a tank of gas, to determine the number of miles the user was able to drive on one gallon of gas.

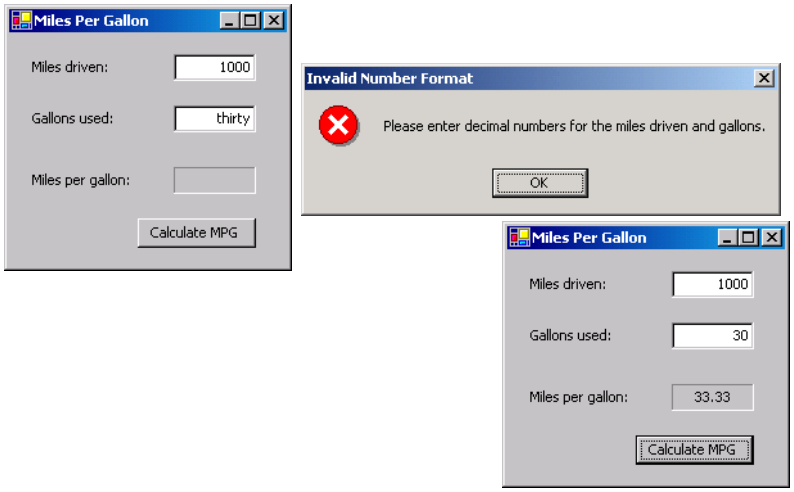


Figure 32.16 Enhanced **Miles Per Gallon** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial32\Exercises\EnhancedMilesPerGallon` to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedMilesPerGallon.sln` in the `EnhancedMilesPerGallon` directory to open the application.

c) ***Adding a `Try` block.*** Find the `btnCalculateMPG_Click` event handler. Enclose all of the code in this event handler in a `Try` block.

d) ***Adding a `Catch` block.*** After the `Try` block, add a `Catch` block to handle any `FormatException`s that may occur in the `Try` block. Inside the `Catch` block, add code to display an error message dialog.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data as shown in Fig. 32.16 and click the **Calculate MPG** `Button`. A `MessageBox` should appear asking you to enter valid input. Enter valid input and click the **Calculate MPG** `Button` again. Verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 32.11 Solution
' MilesPerGallon.vb

Public Class FrmMilesPerGallon
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' calculate and return miles per gallon
   Private Function MilesPerGallon( _
      ByVal dblMilesDriven As Double, _
      ByVal dblGallonsUsed As Double) As Double

      Return dblMilesDriven / dblGallonsUsed
   End Function ' MilesPerGallon

      ' handles Calculate Button's Click event
   Private Sub btnCalculateMPG_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculateMPG.Click

      ' retrieve user input
```

```
      Try

         ' display miles per gallon
         lblOutputValue.Text = String.Format("{0:F}", _
            MilesPerGallon(Convert.ToDouble(txtMilesDriven.Text), _
            Convert.ToDouble(txtGallonsUsed.Text)))

      ' prompt for input in correct format
      Catch formatExceptionParameter As FormatException

         MessageBox.Show( _
            "Please enter decimal numbers for " & _
            "the miles driven and gallons.", _
            "Invalid Number Format", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)

      End Try ' end Try...Catch statement

   End Sub ' btnCalculateMPG_Click

End Class ' FrmMilesPerGallon
```

32.12 (***Enhanced Prime Numbers Application***) Modify the **Prime Numbers** application (Exercise 13.17) to use exception handling to process the `FormatExceptions` that occur when converting the strings in the `TextBoxes` to `Integers` (Fig. 32.17). The original application took two numbers (representing a lower bound and an upper bound) and determined all of the prime numbers within the specified bounds, inclusive. An `Integer` greater than 1 is said to be prime if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime numbers, but 4, 6, 8 and 9 are not.

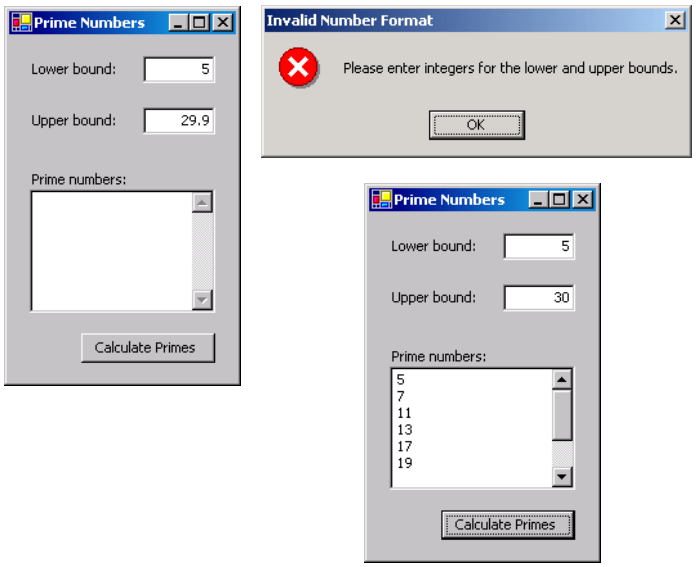


Figure 32.17 Enhanced **Prime Numbers** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial32\Exercises\EnhancedPrimeNumbers` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedPrimeNumbers.sln` in the `EnhancedPrimeNumbers` directory to open the application.

c) ***Adding a Try block.*** Find the `btnCalculatePrimes_Click` event handler. Enclose all the code following the variable declarations in a `Try` block.

d) ***Adding a Catch block***. Add a `Catch` block that catches any `FormatExceptions` that may occur in the `Try` block you added to `btnCalculatePrimes_Click` in *Step c*. Inside the `Catch` block, add code to display an error message dialog.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data as shown in Fig. 32.17 and click the **Calculate Primes** `Button`. A `MessageBox` should appear asking you to enter valid input. Enter valid input and click the **Calculate Primes** `Button` again. Verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 32.12 Solution
' PrimeNumbers.vb

Public Class FrmPrimeNumbers
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' determine if number is prime
   Private Function Prime(ByVal intNumber As Integer) As Boolean
      Dim intCount As Integer ' declare counter

      ' set square root of intNumber as limit
      Dim intLimit As Integer = Convert.ToInt32(Math.Sqrt(intNumber))

      ' loop until intCount reaches square root of intNumber
      For intCount = 2 To intLimit

         If intNumber Mod intCount = 0 Then
            Return False ' number is not prime
         End If

      Next

      Return True ' number is prime
   End Function ' Prime

   ' handles Calculate Primes Button's Click event
   Private Sub btnCalculatePrimes_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnCalculatePrimes.Click

      ' declare variables
      Dim intLowerBound As Integer
      Dim intUpperBound As Integer
      Dim intCounter As Integer
      Dim strOutput As String

      ' attempt to retrieve input from user
      Try
         intLowerBound = Convert.ToInt32(txtLowerBound.Text)
         intUpperBound = Convert.ToInt32(txtUpperBound.Text)

         If intLowerBound <= 0 OrElse intUpperBound <= 0 Then
            MessageBox.Show("Bounds must be greater than 0", _
               "Invalid Bounds", MessageBoxButtons.OK, _
               MessageBoxIcon.Exclamation)
         ElseIf intUpperBound < intLowerBound Then
            MessageBox.Show("Upper bound cannot be less than " & _
```

```
                "lower bound", "Invalid Bounds", _
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
         Else

            ' loop from lower bound to upper bound
            For intCounter = intLowerBound To intUpperBound

               ' if prime number, display in TextBox
               If Prime(intCounter) = True Then
                  strOutput &= (intCounter & ControlChars.CrLf)
               End If

            Next
         End If

         txtPrimeNumbers.Text = strOutput

      Catch formatExceptionParameter As FormatException
         MessageBox.Show( _
            "Please enter integers for the lower and upper " + _
            "bounds.", "Invalid Number Format", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)

      End Try ' end Try...Catch statement

   End Sub ' btnCalculatePrimes_Click

End Class ' FrmPrimeNumbers
```

32.13 (***Enhanced Simple Calculator Application***) Modify the **Simple Calculator** application (Exercise 6.13) to use exception handling to process the `FormatExceptions` that occur when converting the strings in the `TextBoxes` to `Integers` and the `DivideByZeroException` when performing the division (Fig. 32.18). We will define what a `DivideByZeroException` is shortly. The application should still perform simple addition, subtraction, multiplication and division.

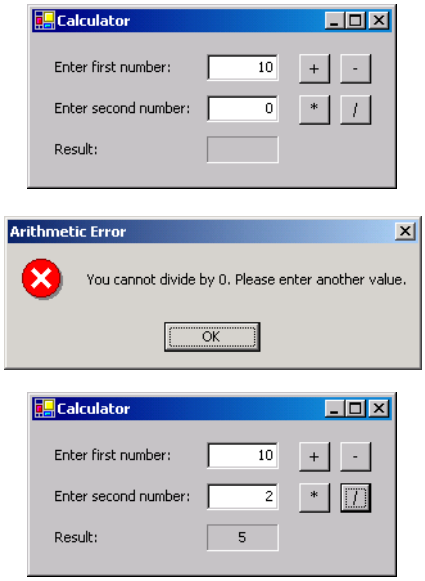


Figure 32.18 Enhanced **Simple Calculator** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial32\Exercises\EnhancedSimpleCalculator` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedSimpleCalculator.sln` in the `EnhancedSimpleCalculator` directory to open the application.

c) ***Adding a `Try` block to the `btnAdd_Click` event handler.*** Find the `btnAdd_Click` event handler. Enclose the body of `btnAdd_Click` in a `Try` block.

d) ***Adding a `Catch` block to the `btnAdd_Click` event handler.*** Add a `Catch` block that catches any `FormatException`s that may occur in the `Try` block that you added in *Step c*. Inside the `Catch` block, add code to display an error message dialog.

e) ***Adding a `Try` block to the `btnSubtract_Click` event handler.*** Find the `btnSubtract_Click` event handler, which immediately follows `btnAdd_Click`. Enclose the body of the `btnSubtract_Click` in a `Try` block.

f) ***Adding a `Catch` block to the `btnSubtract_Click` event handler.*** Add a `Catch` block that catches any `FormatException`s that may occur in the `Try` block that you added in *Step e*. Inside the `Catch` block, add code to display an error message dialog.

g) ***Adding a `Try` block to the `btnMultiply_Click` event handler.*** Find the `btnMulitply_Click` event handler, which immediately follows `btn_Subtract_Click`. Enclose the body of the `btnMultiply_Click` in a `Try` block.

h) ***Adding a `Catch` block to the `btnMultiply_Click` event handler.*** Add a `Catch` block that catches any `FormatException`s that may occur in the `Try` block that you added in *Step g*. Inside the `Catch` block, add code to display an error message dialog.

i) ***Adding a `Try` block to the `btnDivide_Click` event handler.*** Find the `btnDivide_Click` event handler, which immediately follows `btnMultiply_Click`. Enclose the body of the `btnDivide_Click` in a `Try` block.

j) ***Adding a `Catch` block to the `btnDivide_Click` event handler.*** Add a `Catch` block that catches any `FormatException`s that may occur in the `Try` block that you added in *Step i*. Inside the `Catch` block, add code to display an error message dialog.

k) ***Adding a second `Catch` block to the `btnDivide_Click` event handler.*** Immediately following the first `Catch` block inside the `btnDivide_Click` event handler, add a `Catch` block to catch any `DivideByZeroException`s. A **`DivideByZeroException`** is thrown when division by zero in integer arithmetic occurs. Inside the `Catch` block, add code to display an error message dialog.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter valid input for the first number and `0` for the second number, then click the `Button` for division. A `MessageBox` should appear asking you not to divide by `0`. Enter invalid input (such as letters) for the first and second number, then click any one of the `Buttons` provided. This time, a `MessageBox` should appear asking you to enter valid input. Enter valid input and click any one of the `Buttons` provided. Verify that the correct output is displayed.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
' Exercise 32.13 Solution
' SimpleCalculator.vb

Public Class FrmCalculator
   Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   ' handles addition Button's Click event
   Private Sub btnAdd_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnAdd.Click

      ' try to get user input
      Try

         lblResult.Text = (Convert.ToInt32(txtFirstNumber.Text) + _
```

```
            Convert.ToInt32(txtSecondNumber.Text)).ToString

         ' handle case when user enters invalid input
         Catch formatExceptionParameter As FormatException

            ' prompt user for correct input
            MessageBox.Show( _
               "Please enter two integer values.", _
               "Invalid Number Format", MessageBoxButtons.OK, _
               MessageBoxIcon.Error)

         End Try ' end Try...Catch statement

      End Sub ' btnAdd_Click

      ' handles subtraction Button's Click event
      Private Sub btnSubtract_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnSubtract.Click

         ' try to get user input
         Try

            lblResult.Text = (Convert.ToInt32(txtFirstNumber.Text) - _
               Convert.ToInt32(txtSecondNumber.Text)).ToString

         ' handle case when user enters invalid input
         Catch formatExceptionParameter As FormatException

            ' prompt user for correct input
            MessageBox.Show( _
               "Please enter two integer values.", _
               "Invalid Number Format", MessageBoxButtons.OK, _
               MessageBoxIcon.Error)

         End Try ' end Try...Catch statement

      End Sub ' btnSubtract_Click

      ' handles multiplication Button's Click event
      Private Sub btnMultiply_Click(ByVal sender As System.Object, _
         ByVal e As System.EventArgs) Handles btnMultiply.Click

         ' try to get user input
         Try

            lblResult.Text = (Convert.ToInt32(txtFirstNumber.Text) * _
               Convert.ToInt32(txtSecondNumber.Text)).ToString

         ' handle case when user enters invalid input
         Catch formatExceptionParameter As FormatException

            ' prompt user for correct input
            MessageBox.Show( _
               "Please enter two integer values.", _
               "Invalid Number Format", MessageBoxButtons.OK, _
               MessageBoxIcon.Error)

         End Try ' end Try...Catch statement

      End Sub ' btnMultiply_Click

```

```
   ' handles division Button's Click event
   Private Sub btnDivide_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles btnDivide.Click

      ' try to retrieve user input and perform division
      Try

         Dim result As Integer = _
            Convert.ToInt32(txtFirstNumber.Text) \ _
            Convert.ToInt32(txtSecondNumber.Text)

         lblResult.Text = result.ToString

      ' handle case when user enters invalid input
      Catch formatExceptionParameter As FormatException

         ' prompt user for correct input
         MessageBox.Show( _
            "Please enter two integer values.", _
            "Invalid Number Format", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)

      ' handle case when user tries to divide by zero
      Catch divideByZeroExceptionParameter As DivideByZeroException

         ' prompt user for correct input
         MessageBox.Show( _
            "You cannot divide by 0. Please enter another value.", _
            "Arithmetic Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)

      End Try ' end Try...Catch statement

   End Sub ' btnDivide_Click

   ' handles TextChanged event
   Private Sub txtFirstNumber_TextChanged(ByVal sender _
      As System.Object, ByVal e As System.EventArgs) _
      Handles txtFirstNumber.TextChanged

      lblResult.Text = ""
   End Sub ' txtFirstNumber_TextChanged

   ' handles TextChanged event
   Private Sub txtSecondNumber_TextChanged(ByVal sender _
      As System.Object, ByVal e As System.EventArgs) _
      Handles txtSecondNumber.TextChanged

      lblResult.Text = ""
   End Sub ' txtSecondNumber_TextChanged

End Class ' FrmCalculator
```

What does this code do? ▶

32.14 What does the following code do, assuming that `dblValue1` and `dblValue2` are both declared as `Doubles`?

```
Try

   dblValue1 = Convert.ToDouble(txtInput1.Text)
```

```
   dblValue2 = Convert.ToDouble(txtInput2.Text)

   txtOutput.Text = (dblValue1 * dblValue2).ToString

Catch formatExceptionParameter As FormatException

   MessageBox.Show( _
      "Please enter decimal values.", _
      "Invalid Number Format", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)

End Try
```

Answer: This code multiplies two `Doubles` if both inputs in `txtInput1` and `txtInput2` can be converted to type `Double` (that is, decimal or integer values). Otherwise it displays an error message dialog that informs the user to enter decimal values in the `TextBoxes`.

What's wrong with this code? ▶

32.15 The following code should add integers from two `TextBoxes` and display the result in `txtResult`. Assume that `intValue1` and `intValue2` are declared as `Integers`. Find the error(s) in the following code:

```
Try

   intValue1 = Convert.ToInt32(txtInput1.Text)
   intValue2 = Convert.ToInt32(txtInput2.Text)

   txtOutput.Text = (intValue1 + intValue2).ToString

End Try

Catch formatExceptionParameter As FormatException

   MessageBox.Show( _
      "Please enter valid Integers.", _
      "Invalid Number Format", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)

End Catch
```

Answer: The error in this code is that the `Catch` block appears after the `End Try` keywords. All `Catch` blocks must appear before these keywords. Also, there should be no `End Catch` following the `Catch` block.

```
Try

   intValue1 = Convert.ToInt32(txtInput1.Text)
   intValue2 = Convert.ToInt32(txtInput2.Text)

   txtOutput.Text = (intValue1 + intValue2).ToString

   Catch formatExceptionParameter As FormatException

   MessageBox.Show( _
      "Please enter valid Integers.", _
      "Invalid Number Format", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)

End Try
```

Programming Challenge ▶

32.16 (*Enhanced Vending Machine Application*) The **Vending Machine** application from Tutorial 3 has been modified to use exception handling to process the `IndexOutOfRangeExceptions` that occur when selecting items out of the range `0` through `7` (Fig. 32.19). This type of exception will be defined shortly. To get a snack, the user must type the number of the desired snack in the `TextBox`, then press the **Dispense Snack**: `Button`. The name of the snack is displayed in the output `Label`.

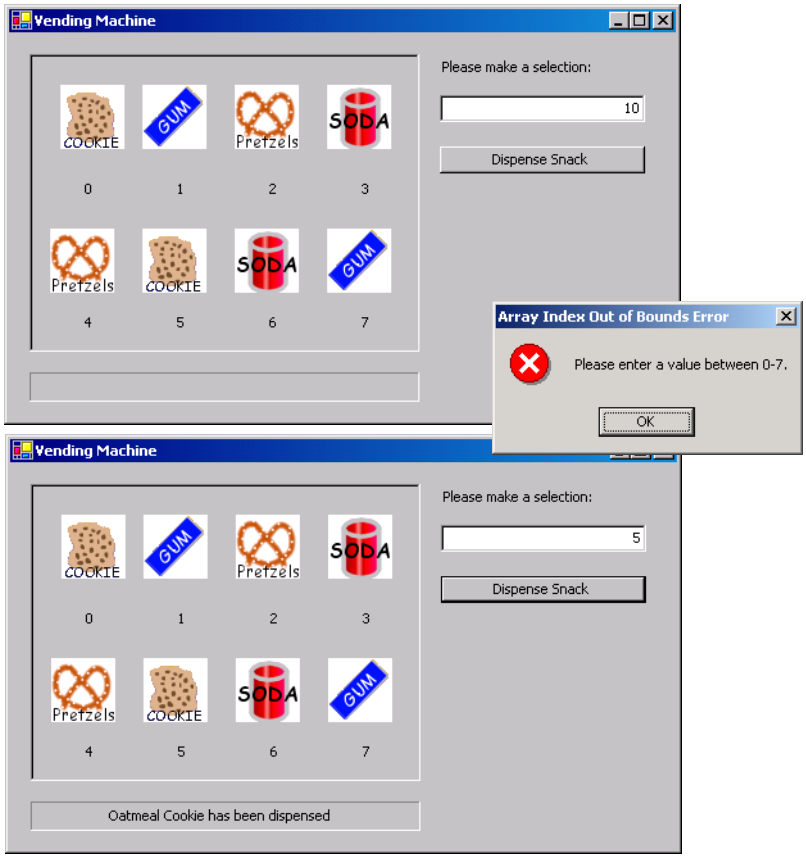


Figure 32.19 **Vending Machine** application.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial32\Exercises\EnhancedVendingMachine` directory to your `C:\SimplyVB` directory.

b) ***Opening the application's template file.*** Double click `EnhancedVendingMachine.sln` in the `EnhancedVendingMachine` directory to open the application.

c) ***Adding a `Try` block.*** Find the `btnDispense_Click` event handler. Enclose all of the code in the event handler in a `Try` block.

d) ***Adding a `Catch` block.*** Add a `Catch` block that catches any `FormatExceptions` that may occur in the `Try` block that you added to `btnDispense_Click` in *Step c*. Inside the `Catch` block, add code to display an error message dialog.

e) ***Adding a second `Catch` block.*** Immediately following the `Catch` block you added in *Step d*, add a second `Catch` block to catch any `IndexOutOfRangeExceptions` that may occur. An **`IndexOutOfRangeException`** occurs when the application attempts to access an array with an invalid index. Inside the `Catch` block, add code to display an error message dialog.

f) ***Running the application.*** Select **Debug > Start** to run your application. Make an out of range selection (for instance, `32`) and click the **Dispense Snack** `Button`. Verify that the proper `MessageBox` is displayed for the invalid input. Enter letters for a

selection and click the **Dispense Snack** `Button`. Verify that the proper `MessageBox` is displayed for the invalid input.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answers:

```
' Exercise 32.16 Solution
' VendingMachine.vb

Public Class FrmVendingMachine
    Inherits System.Windows.Forms.Form

   ' Windows Form Designer generated code

   Dim strSnacks As String() = New String() _
      {"Chocolate Chip Cookie", "Bubble Gum", _
      "Plain Pretzel", "Soda", "Salted Pretzel", _
      "Oatmeal Cookie", "Diet Soda", "Sugar-free Gum"}

   ' method to dispense snack
   Private Sub btnDispense_Click(ByVal sender As _
      System.Object, ByVal e As System.EventArgs) _
      Handles btnDispense.Click

      ' try to get user input
      Try

         ' get user input
         Dim intSelection As Integer = _
            Convert.ToInt32(txtSelection.Text)

         lblOutput.Text = strSnacks(intSelection).ToString & _
            " has been dispensed"

      ' handle case when user enters invalid input
      Catch formatExceptionParameter As FormatException

         MessageBox.Show( _
            "Please enter an integer value", _
            "Number Format Exception", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)

      ' handle case when user inputs number not in array bounds
      Catch indexExceptionParamter As IndexOutOfRangeException

         MessageBox.Show( _
            "Please enter a value between 0-7.", _
            "Array Index Out of Bounds Error", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)

      End Try ' end Try...Catch statement

   End Sub ' btnDispense_Click

End Class ' FrmVendingMachine
```