• Use text boxes, masked text boxes, rich text boxes, group boxes, check boxes, radio buttons, and picture boxes effectively
• Set the BorderStyle property to make controls appear flat or three-dimensional
• Select multiple controls and move, align, and set common properties for them
Introducing More Controls
Text Box

- Allows for user input
  - The user can use Cut, Copy, and Paste while entering text
- Text property
  - What is entered or displayed in the text box
- TextAlign property
  - Determines alignment of text within the control
    - HorizontalAlignment.Left
    - HorizontalAlignment.Right
    - HorizontalAlignment.Center
Text Boxes

• Allows for user input
  – Cut, copy, and paste are available for use
• Text property
  – What is entered or displayed in the text box
    • nameLabel.Text = nameTextBox.Text;
  – Literals can be assigned to a text box’s Text property
    • messageTextBox.Text = “Watson, come here.”;
• TextAlign property
  – Determines alignment of text within the text box
    • HorizontalAlignment.Left
    • HorizontalAlignment.Right
    • HorizontalAlignment.Center
Masked Text Boxes

• A specialized form of the Textbox control
• Specify the format (the Mask property) of user entered data
  – Example names
    • dateMaskedTextBox
    • phoneMaskedTextBox
Rich Text Boxes

• A rich text box allows specialized formatting
  – Character and paragraph formatting, similar to a word processor
  – Displays a URL as a link if DetectUrl property is set to true
  – Load formatted text into a rich text box from a .rtf file using the rich text box’s LoadFile method
    • sampleRichTextBox.LoadFile(“Rich Text Boxes.rtf”);
Displaying Text on Multiple Lines

- By default, a value too long for a text box does not wrap to a second line.
- Set two properties of a text box or rich text box to cause a long value to wrap:
  1. Set `WordWrap` to true
  2. Set `Multiline` to true
- The text box height property must be set tall enough to display the lines.
Group Boxes

- Container for other controls, such as radio buttons and check boxes
- Separates controls into logical groups
- Words in a group box Text property appear on the top edge of the box
- Change group box name only if referred to in code
  - In code, set Enable property to false to disable all controls inside the box
Check Boxes

• Allows the user to select or deselect an option
• In any group of check boxes, any number can be selected
• \textit{Checked property}
  – checked = true
  – unchecked = false
• Write an event handler for the \textit{CheckedChanged} event
  – Executes when the user clicks in the box
• Use the \text{Text} property of a check box for the text that appears next to the box
Radio Buttons

- Only one button of a group may be selected
- Radio buttons operate in groups
  - Create a group box and place a group of buttons inside
  - Any buttons directly on the form operate as one group
- Checked property
  - selected = true
  - unselected = false
- Write an event handler for the CheckedChanged event
  - Executes when the user selects a button
- Use the Text property of a radio button for the text that appears next to the button
• *PictureBox control* holds an image
• Set the *Image property* to a graphic file that has been added to the project's resources
• Graphic file extensions
  – .bmp, .gif, .jpg, .jpeg, .png, .ico, .emf, .wmf
• Place a PictureBox control on a form
  – Select its Image property in the Properties window
  – Click the Properties button
• The Select Resource dialog box
  – Click the Import button to add an image to the list

Click Import and navigate to image files
Picture Boxes – 4 (Properties)

- **SizeMode**
  - *StretchImage* resizes the graphic to fill the control

- **Visible**
  - Code to make a picture box invisible at run time:
    - `logoPictureBox.Visible = false;`

- **Image**
  - If assigned at run time, refer to project name, Resources folder, and name of graphic resource
  - Clearing a Picture Box
    - Set Image property to *null*
**Picture Boxes - 5**

- **Adding and Removing Resources**
  - Add, remove, and rename resources using the Visual Studio *Project Designer*.

- **Project/ProjectName Properties**

  ![Project Designer](image)

  - Add a new resource to the project resources.
  - Remove a resource from the project resources.
Smart Tags

• Smart tags help set the most common properties of controls
  – Click arrow in upper-right corner of control
  – Shortcut displays a few properties
Using Images for Forms and Controls

• An image can be set as the background of a form or control
  – Set form’s BackgroundImage property to a graphic resource
  – Set form’s BackgroundImageLayout property to *Tile, Center, Stretch, or Zoom*
  – Buttons, check boxes and radio buttons
    • Set control’s Image property to a graphic resource
Setting a Border and Style

- Most controls can appear to be three-dimensional or flat
- Labels, text boxes, and picture boxes have a BorderStyle property
  - None
  - FixedSingle
  - Fixed3D
- Text boxes default to Fixed3D; Labels and picture boxes default to None
Drawing a Line

• Draw a line to create a logo or to divide the screen
• Use a Label control to draw a line
  – Properties
    • AutoSize = false
    • Text = blank
    • BorderStyle = None
    • BackColor = desired color for line
    • Size - Width and Height = desired size
• Use the LineShape control to draw a line
• Use the Graphics methods to draw a line
  – Covered in Chapter 13
Working with Multiple Controls - 1

• Select more than one control at a time
  – Move controls as a group
  – Set similar properties for the group
  – Align the controls
Selecting Multiple Controls - 2

• Use the mouse to drag a selection box around multiple controls

• When multiple controls are selected, each has resizing handles (if resizable)
• Selecting Multiple Controls
  – Drag a selection box around the controls
  – Click on a control, hold down the Ctrl or Shift key and click more controls
  – Ctrl-click (or Shift-click) a control to deselect, without changing rest of group
  – *Edit/Select All*
    • Keyboard shortcut: Ctrl + A
• Deselect a group of controls by clicking on the form (not on a control)
• Move controls as a group
  – Drag entire group to a new location
• Set properties for groups of controls
  – Property window displays shared properties, all properties in group can be changed at once
Working with Multiple Controls - 4

- Aligning controls
  - Use the *Format* menu or Layout toolbar to
    - Make controls equal size (width, height, both)
    - Align controls (tops, bottoms, centers)
    - Create equal spacing between controls (horizontal and/or vertical)
Designing Applications for User Convenience

• Create programs that are easy to use
  – Interface clear and consistent
  – Programs look and behave like other Windows programs
  • Controls operate in standard way
  • Keyboard access keys
  • Accept button
  • Correct tab function
  • Define ToolTips
Designing the User Interface

• Follow industry standards for color, size, and placement of controls
  – Use a neutral color such as gray for the form, which looks professional
    • By default the BackColor property of forms and controls is set to Control
  – White background for controls requiring user input
  – Gray background for controls that the user cannot change
  – Group controls to aid user
    • Create group boxes to hold related items
  – Use sans serif fonts; not boldface or large
Defining Keyboard Access Keys

- Many people prefer to use the keyboard rather than a mouse
- Also called hot keys
- User presses Alt + underlined letter
- Defined by adding an ampersand to Text property of control
  &OK = OK
  E&xit = Exit
- Use Windows standard keys
- Do not give two controls the same access key
Setting the Accept and Cancel Buttons

• Accept button
  – Set the form's `AcceptButton property`
  – Identified visually on the form by a thicker border
  – Responds to the Enter key

• Cancel button
  – Responds to the Esc key
  – Set the form's `CancelButton property`
Setting the Tab Order for Controls

- One control on a form always has the focus or active control
- Not all controls can receive the focus
  - Focus
    - Text boxes, buttons, check boxes, radio buttons
  - No focus
    - Labels, picture boxes
The Tab Order - 1

- Controls capable of receiving focus have a *TabStop property*
  - True by default
  - Set to false to not have focus stop on a control when the user presses the Tab key
- *TabIndex property* determines the order the focus moves as the Tab key is pressed
The Tab Order - 2

- Set the TabIndex for a text box to one higher than its label
  - Press the access key for the label, focus goes to the first control following the label (labels do not receive focus)
The Tab Order - 3

- **TabIndex properties** for controls are assigned in sequence beginning with zero as controls are created on a form.
- When a form loads, the control with the lowest TabIndex receives the focus.
- Use a control's keyboard access key to transfer focus to that control.
- Radio buttons behave differently:
  - Tab to the group and use Up and Down arrow keys to select correct button.
Setting the Tab Order

• Set each control’s TabIndex property in the Properties window
• Set TabIndexes automatically
  – Design window active
  – View/Tab Order or click Tab Order button on Layout toolbar
  – Small numbers appear in upper-left corner of each control
  – Click on each control, in Tab order
  – Press Esc key when all boxes change to blue
Setting the Form’s Location on the Screen

• Set the form's screen position by setting the `StartPosition` property.

• To center a form on the user's screen, set the Start Position property to `CenterScreen`.
Creating Tool Tips - 1

• ToolTips are small popup labels that appear when the mouse pointer is paused over a control

• Add a ToolTip component to a form
  – The form and each control have a new property
    • ToolTip on toolTip1

• The ToolTip control appears in the component tray at the bottom of the Form Designer

• Set the ToolTip on toolTip1 property to the text that will appear when the user pauses the mouse pointer over the control.
Creating Tool Tips - 2

• Set properties of the ToolTip component to modify a ToolTip’s appearance
  – Select ToolTip component in component tray
  – Change properties
    • BackColor, ForeColor, IsBalloon, ToolTipIcon
    • Properties apply to all ToolTips with that component

• Create multiple ToolTip components to have a variety of ToolTip appearances
Creating ToolTips - 2

Hover the mouse pointer over a control to pop up the ToolTip.

ToolTip component in the component tray.
Coding for the Controls

- Clear out contents of text boxes and labels
- Reset the focus (the active control)
- Change the color of text
- Change the text in a ToolTip
Clearing Text Boxes and Labels

• Set the Text property of a Text Box or a Label to an *empty string*
  – Two quotes with no space between: ""
  – string.Empty

• A Text box (not a Label) can use the *Clear* method
  – dataTextBox.Clear();
Writing Code for the Controls

Clearing Text Boxes and Labels

- Can set the Text property of a Text Box or a Label to an empty string
  - Use two quotes with no space between: \\
  - Or, use string.Empty

- For a Text Box (not a Label) can use the Clear method

```csharp
nameTextBox.Text = " ";
messageLabel.Text = " ";
dataTextBox.Clear();
messageLabel.Text = string.Empty;
```
Resetting the Focus

• In code reset the focus to the control that should receive the input
  – Places the insertion point in a text box
  – Makes the control the active control
  – Cannot set focus to a disabled control

• Use the Focus method of any control that can receive the focus

  //Make the insertion point appear in this text box.
  nameTextBox.Focus();
Setting the Checked Property of Radio Buttons and Check Boxes

• Select or deselect radio buttons and check boxes at design time or at run time

• Checked property
  – Selected = true
  – Deselected = false

• In a group of radio buttons, setting one button to true sets all others to false
  – Only one radio button in a group can be selected at one time

```csharp
//Make button selected.
redRadioButton.Checked = true;

//Make box unchecked.
displayCheckBox.Checked = false;

//Make box checked.
displayCheckBox.Checked = true;
```
• Set the visibility of a control at run time

    //Make label invisible.
    messageLabel.Visible = false;

• Make visibility of a control depend on selection of a radio button or check box

    //Make the visibility of the label match the setting in the check box.
    messageLabel.Visible = displayCheckBox.Checked;
Disabling Controls

• Enabled property determines whether a control is available or grayed out (disabled)
  – Default setting is true

• Enabled property of a control can be set at design time or in code at run time
  – displayButton.Enabled = true;

• Usually best to disable rather than hide controls
Setting Properties Based on User Actions

- Change Enabled or Visible property of a control based on a user action
- Example: User signs in and clicks the signInButton

```csharp
private void signInButton_Click(object sender, EventArgs e)
{
    // Set visibility and enable controls.

    welcomeRichTextBox.Visible = true;
    clothingRadioButton.Enabled = true;
    equipmentRadioButton.Enabled = true;
    juiceBarRadioButton.Enabled = true;
    membershipRadioButton.Enabled = true;
    personalTrainingRadioButton.Enabled = true;
}
```
Changing the Color of Text

• Most controls have ForeColor and BackColor properties
  – ForeColor = Color of text
  – BackColor = Color around text

• The Color Constants
  – *Color constants* in Color class specify a color
    • Type "Color." in the editor to see a full list of colors
      ```csharp
      nameTextBox.ForeColor = Color.Red;
      messageLabel.ForeColor = Color.White;
      ```
Concatenating Text

- Concatenating joins text strings
  - "Tack" one string on the end of another
  - i.e. join a literal and a property
- Use a plus sign (+) between the two strings

```csharp
messageLabel.Text = "Your name is: " + nameTextBox.Text;
nameAndAddressLabel.Text = nameTextBox.Text + "  " + addressTextBox.Text;

welcomeRichTextBox.Text = "Welcome Member #" + memberIDMaskedTextBox.Text + Environment.NewLine + nameTextBox.Text;
```
Printing a Form

- Add PrintForm component to form
- Add *Print* button to form
- Use PrintForm’s *Print* method to send form to printer
- Use PrintForm’s *PrintAction* property to send form to the Print Preview window

```csharp
//Print the form on the printer.
printForm1.Print();

//Print to the Print Preview window.
printForm1.Print();
```