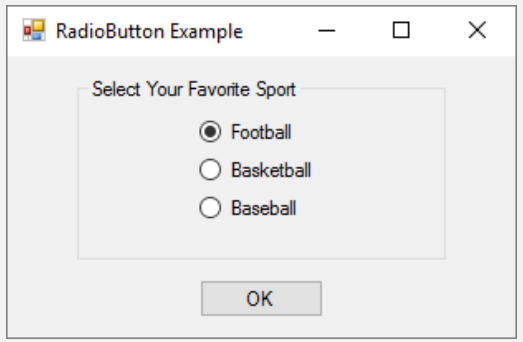
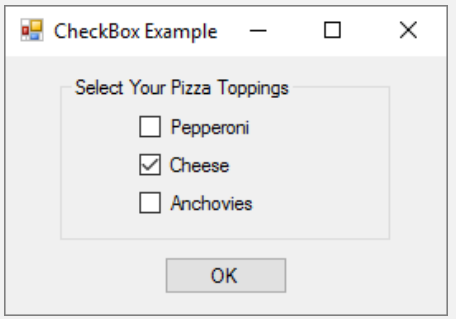
4.9 Radio Buttons and Check Boxes

- **Radio buttons** allow users to select one choice from several possible choices
  - Clicking on a radio button, the program automatically deselects any others. This is known as mutually exclusive selection
- **Check boxes** allows users to select multiple options





AUBURN UNIVERSITY

RadioButton Control

- The .NET Framework provides the RadioButton Control for you to create a group of radio buttons
- Common properties are:
  - **Text**: holds the text that is displayed next to the radio button
  - **Checked**: a Boolean property that determines whether the control is selected or deselected

RadioButton Control

- When you want to create a group of radio buttons on a form, you use the **RadioButton control**

- **RadioButton controls** are normally grouped in one of the following ways:
  - You place them inside a **GroupBox control**. All RadioButton controls that are inside a GroupBox are members of the same group.
  - You place them inside a **Panel control**. All RadioButton controls that are inside a Panel are members of the same group.
  - You place them on a form but not inside a GroupBox or a Panel. All RadioButton controls that are on a form but not inside a GroupBox or Panel are members of the same group.

Working with Radio Buttons in Code

- In code, use a decision structure to determine whether a RadioButton is selected. For example,

```
If (choiceRadioButton.Checked) { } else { }
```

- You do not need to use the == operator, although the following is equivalent to the above

```
If (choiceRadioButton.Checked == true) { } else { }
```

CheckBox Control

- The .NET Framework provide the CheckBox Control for you to give the user an option, such as true/false or yes/no
- Common properties are:
  - **Text**: holds the text that is displayed next to the radio button
  - **Checked**: a Boolean property that determines whether the control is selected or deselected
- In code, use a decision structure to determine whether a CheckBox is selected. For example,

```
If (option1CheckBox.Checked) { } else { }
```

The CheckedChanged Event

- Anything a RadioButton or a CheckBox control's Checked property changes, a CheckedChanged event is raised
- You can create a CheckedChanged event handler and write codes to respond to the event
- Double click the control in the Designer to create an empty CheckedChanged event handler similar to:

```
private void yellowRadioButton_CheckedChanged(object sender, EventArgs e)
{
}
```

5.1 More About ListBoxes

- ListBox controls have various methods and properties that you can use in code to manipulate the ListBox's contents
- The **`Items.Add`** method allows you to add an item to the ListBox control

```
ListBoxName.Items.Add(Item);
```

  - where *`ListBoxName`* is the name of the ListBox control; *Item* is the value to be added to the Items property
- The **`Items.Clear`** method can erase all the items in the Items proeprty

```
employeeListBox.Items.Clear();
```

- The **Count** property reports the number of items stored in the ListBox

Sample Codes

- You can add string literals

```
private void addButton_Click(object sender, EventArgs e)
{
   namesListBox.Items.Add("Chris");
   namesListBox.Items.Add("Alicia");
}
```

- You can add values of other types as well

```
private void addButton_Click(object sender, EventArgs e)
{
   namesListBox.Items.Add(10);
   namesListBox.Items.Add(20);
   namesListBox.Items.Add(17.5);
}
```

9.6 Creating Multiple Forms in a Project

- Every project that has a form in a Visual C# has a class.
  - The default name of the form is **Form1** and its class is also named **`Form1`** which is stored in the **Form1.cs** file.
- A Visual C# project can have multiple forms.
  - Each form has its own class that can be instantiated and displayed on the screen.
- When you add additional forms to a project, you add additional classe(s), which are stored in their own files.
- When you create event handler for a specific form's controls, you write them as methods in the form's class.

Displaying a Form

- To display a form in your application, you need to create an instance of the form's class. E.g.

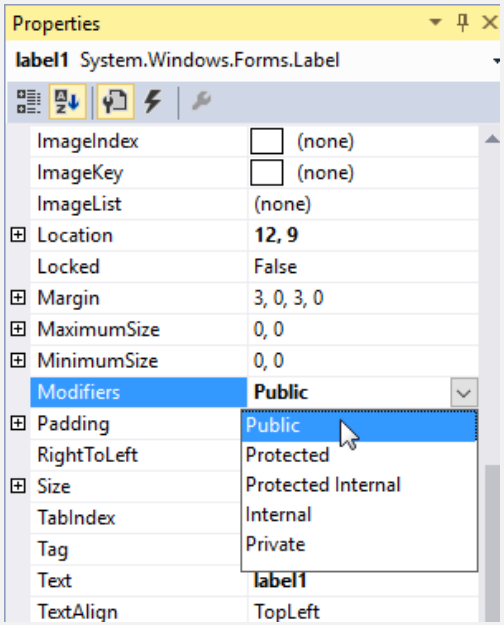
```
ErrorForm myErrorForm = new ErrorForm();
```

- The above code:
  - declares a reference variable named `myErrorForm`;
  - creates an object of the `ErrorForm` class in memory; and
  - assigns a reference to the object to the `myErroForm` variable.
- After creating the instance, use **`ShowDialog`** method to display the form. E.g.

```
myErrorForm.ShowDialog();
```

Accessing Control on a Different Form

- Controls provided by the .NET Framework have private access by default.
- This means that a control that exists on a particular form can be accessed only by code that is written in that form's class.
- You can change the access of a control by changing its **Modifiers** property in the Properties window.
- If you change a control's Modifiers property to Public, the control can be accessed by code outside the form's class.



AUBURN UNIVERSITY

Accessing Control on a Different Form

- For example, suppose `GreetingsForm` has a Label control named `messageLabel`, and the `messageLabel` control's Modifiers property has been set to Public.
- The following code shows how you can create an instance of `GreetingsForm`, assign a value to the `messageLabel` control's Text property, and then display the form:

```
GreetingsForm greetingsForm = new GreetingsForm();
greetingsForm.messageLabel.Text = "Good day!";
greetingsForm.ShowDialog();
```

Modal and Modeless Forms

- A **modal form** or dialog box must be closed or hidden before you can continue working with the rest of the application.
  - When a **modal form** is displayed, no other form in the application can receive the focus until the modal form is closed.
- A **modeless form** allows the user to switch focus to another form while it is displayed.
  - The user does not have to close a modeless form to switch focus to another form.

Flow of Executions

- In a modal form, statement next to the `ShowDialog` method will not execute until the modal form is closed. e.g.

```
statement;
statement;
messageForm.ShowDialog();
statement;
statement;
statement;
```
// not executed

- In a modeless form, statement next to the `Show` method will execute immediately after the modeless form is displayed. e.g.

```
statement;
statement;
messageForm.ShowDialog();
statement;
statement;
statement;
```
// executed

5.6 Using File for Data Storage

AUBURN UNIVERSITY

- When a program needs to save data for later use, it writes the data in a file
- There are always three steps:
  - Open the file: create a connection between the file and the program
  - Process the file: either write to or read from the file
  - Close the file: disconnect the file and the program
- In general, there are two types of files:
  - Text file: contains data that has been encoded as text using a scheme such as Unicode
  - Binary file: contains data that has not been converted to text. You cannot view the contents of binary files with a text editor.
- This chapter only works with text files

File Accessing

- A file object is an object that is associated with a specific file and provides a way for the program to work with that file
- The .NET Framework provides two classes to create file objects through the **`System.IO`** namespace
  - **`StreamWriter`**: for writing data to a text file
  - **`StreamReader`**: for reading data from a text file
- You need to write the following directive at the top of your program:

```
Using System.IO;
```

Writing Data to a File

- Start with creating a `StreamWriter` object

```
StreamWriter outputFile;
```

- Use one of the File methods to open the file to which you will be writing data. Sample File methods are:
  - `File.CreateText`
  - `File.AppendText`
- Use the **`Write`** or **`WriteLine`** method to write items of data to the file
- Close the connection.

Sample Code

```
StreamWriter outputFile;
outputFile = File.CreateText("courses.txt");
outputFile.WriteLine("Introduction to Computer Science");
outputFile.WriteLine("English Composition");
outputFile.Write("Calculus I");
outputFile.Close();
```

- The **`WriteLine`** method writes an item of data to a file and then writes a newline characters which specifies the end of a line
- The **`Write`** method writes an item to a file without a newline character

CreateText vs. AppendText

- The previous code uses the `File.CreateText` method for the following reasons:
  - It creates a text file with the name specified by the argument. If the file already exists, its contents are erased
  - It creates a `StreamWriter` object in memory, associated with the file
  - It returns a reference to the `StreamWriter` object
- When there is a need not to erase the contents of an existing file, use the `AppendText` method

```
StreamWriter outputFile;
outputFile = File.AppendText("Names.txt");
outputFile.WriteLine("Lynn");
outputFile.WriteLine("Steve");
outputFile.Close();
```

AUBURN UNIVERSITY

Specifying the Location of an Output File

- If you want to open a file in a different location, you can specify a path as well as filename in the argument
- Be sure to prefix the string with the @ character
- `StreamWriter outputFile;`

```
outputFile =
File.CreateText(@”C:\Users\chris\Documents\Names.txt”);
```

Reading Data from a File

- Start with creating a `StreamReader` object

```
StreamReader inputFile;
```

- Use the **`File.OpenText`** method to open the file from which you will be reading data

```
inputFile = File.OpenText("students.txt");
```

- Use the **`Read`** or **`ReadLine`** method to read items of data from the file

  - `StreamReader.ReadLine`: Reads a line of characters from the current stream and returns the data as a string.
  - `StreamReader.Read`: Reads the next character or next set of characters from the input stream.

- Close the connection

Reading a File with a Loop

- `StreamReader` objects have a Boolean property named EndOfStream that signals whether or not the end of file has been reached
- You can write a loop to detect the end of the file.

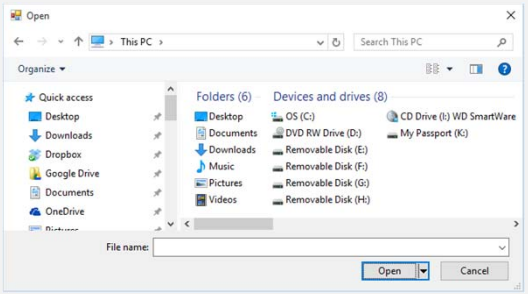
```
while (inputFile.EndOfStream == false) { }
```

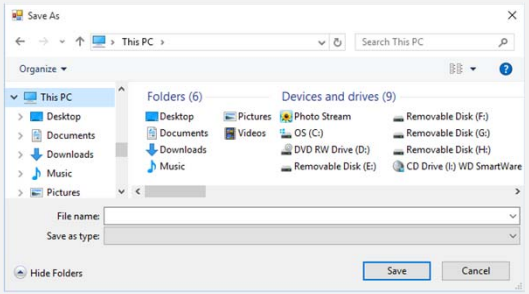
- Or

```
while (!inputFile.EndOfStream) { }
```

5.7 The OpenFileDialog and SaveFileDialog Controls

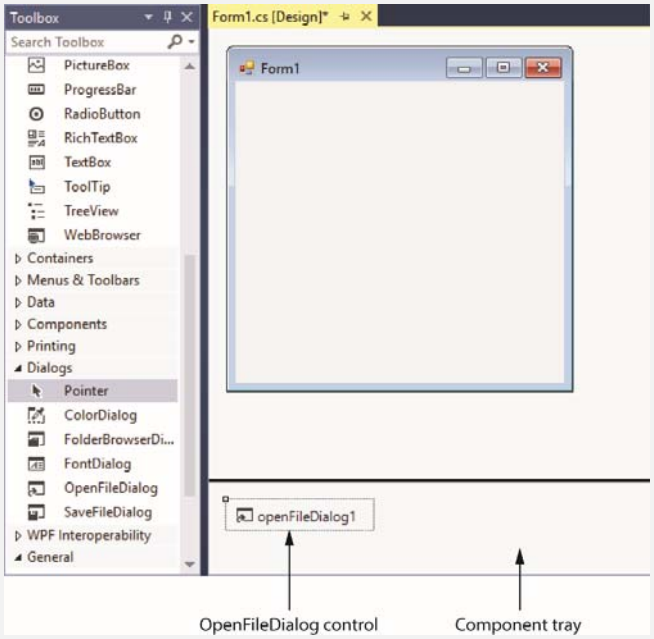
- The **OpenFileDialog** and **SaveDialog** controls allow your application to display standard Windows dialog boxes for opening and saving files
- Unlike Label, Button, and TextBox, they are invisible controls
- The OpenFileDialog control displays a standard Windows *Open* dialog box.
- The SaveDialog control displays a standard Windows *Save As* dialog box





Displaying an Open Box

- When adding an OpenFileDialog control to the form, it does not appear on the form, but in an area at the bottom of the Designer called the component tray



Displaying an Open Box

- In code, you can display an Open dialog box by calling the `ShowDialog` method:

```
private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.ShowDialog();
}
```

Detecting the User's Selection

- The **`showDialog`** method returns a value that indicates which button the user clicks to dismiss the dialog box
  - If the user clicked the Open button, the value `DialogResult.OK` is returned
  - If the user clicked the Cancel button, the value `DialogResult.Cancel` is returned
  - The following is an example that calls the ShowDialog method to determine the user's choice:

```
if (openFile.ShowDialog() == DialogResult.OK) { }
else if (openFile.ShowDialog() == DialogResult.Cancel) { }
else { }
```

The Filename and InitialDirectory Property

- When the user selects a file with the Open dialog box, the file's path and filename are stored in the control's **Filename** property
- The following is an example of how to open the selected file:

```
if (openFile.ShowDialog() == DialogResult.OK)
{
  inputFile = File.OpenText(openFile.Filename);
}
else { }
```

- You can specify a directory to be initially displayed with the InitialDirectory property. For example,

```
openFile.InitialDirectory = "C:\Data";
```

Displaying a Save As Dialog Box

- Use the following to call the SaveFileDialog control's `ShowDialog` method

```
saveFile.ShowDialog();
```

- Use the following to detect the user's choice

```
if (saveFile.ShowDialog() == DialogResult.OK) { }
```

- Use the following to open the selected file

```
if (saveFile.ShowDialog() == DialogResult.OK)
{
  outputFile = File.CreateText(openFile.Filename);
}
```