TryParse Method

- With the TryParse methods, you can determine whether a string (such as a control's Text property) contains a value that can be converted to a specific data type before it is converted to that data type.
- The TryParse methods do not throw an exception, so you can use them without a try-catch statement.
- There are several TryParse methods in the .NET Framework
  - We use the int.TryParse method to convert a string to an int.
  - We use the double.TryParse method to convert a string to a double.
  - We use the decimal.TryParse method to convert a string to a decimal.
- When you call one of the TryParse methods, you pass two arguments:
  1. the string that you want to convert
  2. the name of the variable in which you want to store the converted value.
- The general format of how the int.TryParse method is called:

int.TryParse(string, out targetVariable)

TryParse Method

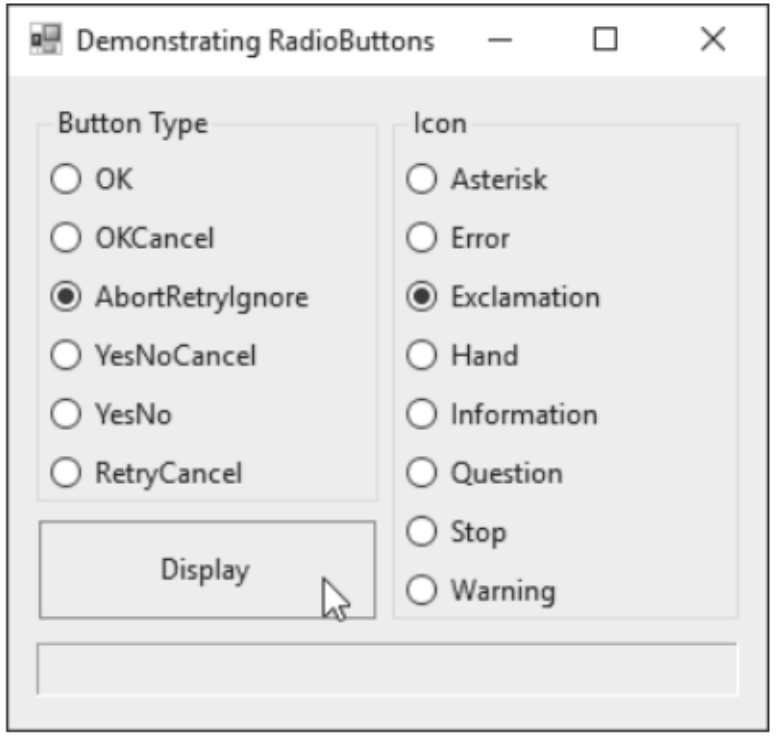
- The TryParse methods return either true or false, they are commonly called as the Boolean expression in an if statement.

```
1  int number;
2
3  if (int.TryParse(inputTextBox.Text, out number))
4  {
5      MessageBox.Show("Success!");
6  }
7  else
8  {
9      MessageBox.Show("Enter a valid integer.");
10 }
```

- If inputTextBox.Text is successfully converted to an int, the resulting value is assigned to the number variable, and the method returns true. That causes the statement in line 5 to execute.
- If inputTextBox.Text cannot be converted to an int, the value 0 is assigned to the number variable, and the method returns false. That causes the statement in line 9 (after the else clause) to execute.

Radio Buttons

- Radio buttons are useful when you want the user to select one choice from several possible choices.
- only one radio button in a group may be selected at a time.



Radio Buttons

- When you want to create a group of radio buttons on a form, you use the RadioButton control, which is found in the Common Controls section of the Toolbox .
- 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.

Radio Buttons

- The RadioButton Control's Text Property: holds the text that is displayed next to the radio button's circle.
- The RadioButton Control's Checked Property:
  - Checked property that determines whether the control is selected or deselected.
  - The Checked property is a Boolean property, which means that it may be set to either True or False.
  - When the Checked property is set to True, the RadioButton is selected, and when the Checked property is set to False, the RadioButton is deselected.
  - By default, the Checked property is set to False.

Working with Radio Buttons in Code

- You can determine whether a RadioButton control is selected by testing its Checked property.
- For example, suppose a form has a RadioButton control named choice1RadioButton .
- The following if statement determines whether it is selected:

```
if (choice1RadioButton.Checked)
{
MessageBox.Show("You selected Choice 1");
}
```

- Notice that we did not have to use the == operator to explicitly compare the Checked property to the value true .
- This code is equivalent to the following:

```
if (choice1RadioButton.Checked == true)
{
MessageBox.Show("You selected Choice 1.");
}
```

The CheckedChanged Event

- Any time a RadioButton or a CheckBox control's Checked property changes, a CheckedChanged event happens for that control.
- If you want some action to immediately take place when the user selects (or deselects) a RadioButton or CheckBox control, you can create a CheckedChanged event handler for the control and write the desired code in that event handler.
- To create a CheckedChanged event handler for a RadioButton or a CheckBox, simply double-click the control in the Designer.
- An empty CheckedChanged event handler is created in the code editor

Example

Color Theme application

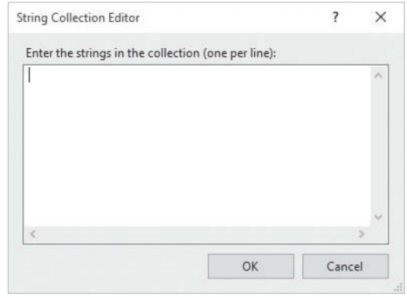
Create an application that allows the user to select a color using RadioButton controls.

When the user selects a color, the form's background color is changed to that color immediately.

List Boxes

- Once you create a ListBox control, you add items to its Items property.
- The items that you add to a ListBox's Items property are displayed in the ListBox.
- To store values in the Items property at design time, follow these steps:
  1. Select the ListBox control in the Designer window.
  2. In the Properties window, the setting for the Items property is displayed as (Collection). When you select the Items property, an ellipses button appears.
  3. Click the ellipses button. The String Collection Editor dialog box appears, as shown in Figure 4-32.
  4. Type the values that are to appear in the ListBox into the String Collection Editor dialog box. Type each value on a separate line by press each entry.
  5. When you have entered all the values, click the OK butt

Figure 4-32 The *String Collection Editor* dialog box



SelectedItem Property

- When the user selects an item in a ListBox, the item is stored in the ListBox's SelectedItem property.
- For example, suppose an application has a ListBox control named fruitListBox and a string variable named selectedFruit.
- The fruitListBox control contains the items Apples, Pears, and Bananas.
- If the user has selected Pears, the following statement assigns the string "Pears" to the variable selectedFruit:

```
selectedFruit = fruitListBox.SelectedItem.ToString();
```

Determining Whether an Item Is Selected

- An exception will occur if you try to get the value of a ListBox's SelectedItem property when no item is selected in the ListBox.
- For that reason, you should always make sure that an item is selected before reading the SelectedItem property.
- You do this with the SelectedIndex property.

SelectedIndex Property

- The items that are stored in a ListBox each have an index.
- The index is simply a number that identifies the item's position in the ListBox.
- The first item has the index 0, the next has the index 1, and so forth.
- The last index value is n − 1, where n is the number of items in the ListBox.
- When the user selects an item in a ListBox, the item's index is stored in the ListBox's SelectedIndex property.
- If no item is selected in the ListBox, the SelectedIndex property is set to −1.

SelectedIndex Property

- You can use the SelectedIndex property to make sure that an item is selected in a ListBox before you try to get the value of the SelectedItem property.
- Example:

```
if (fruitListBox.SelectedIndex != −1)
{
selectedFruit = fruitListBox.SelectedItem.ToString();
}
```

ListBoxes

- In Chapter 4, we introduced the ListBox control, which displays a list of items and allows the user to select one or more items from the list.
- In Chapter 4 you add items to a ListBox control's Items property, and those items are displayed in the ListBox.
- In this chapter, we use ListBox controls to display output.
- At design time, you can use the Properties window to add items to the control's Items property.
- You can also write code that adds items to a ListBox control at run time.
- To add an item to a ListBox control with code, you call the control's Items.Add method.

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

Items.Count Property

- ListBox controls have an Items.Count property that reports the number of items stored in the ListBox.
- If the ListBox is empty, the Items.Count property equals 0.
- For example, assume an application has a ListBox control named employeesListBox.
- The following if statement displays a message box if there are no items in the ListBox:

```
if (employeesListBox.Items.Count == 0)
{
MessageBox.Show("There are no items in the list!");
}
```

- The Items.Count property holds an integer value.
- Assuming numEmployees is an int variable, the following statement assigns the number of items in the employeesListBox to the numEmployees variable:

```
numEmployees = employeesListBox.Items.Count;
```

Items.Clear Method

- ListBox controls have an Items.Clear method that erases all the items in the Items property.

```
ListBoxName.Items.Clear();
```

- For example, assume an application has a ListBox control named employeesListBox.
- The following statement clears all the items in the list:

```
employeesListBox.Items.Clear();
```

Items.RemoveAt

- uses method RemoveAt to remove an item from the ListBox
- ListBox.Items.RemoveAt(Index);

Example

Adds, removes and clears ListBox items Application

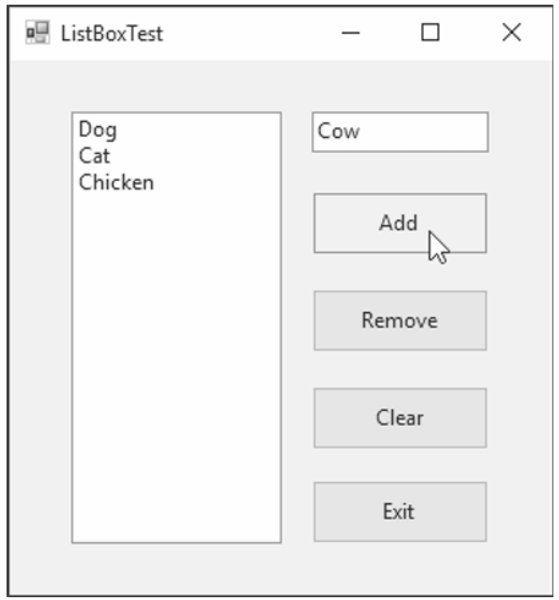
When the user clicks the Add Button, the new item appears in displayListBox.

Similarly, if the user selects an item and clicks Remove, the item is deleted.

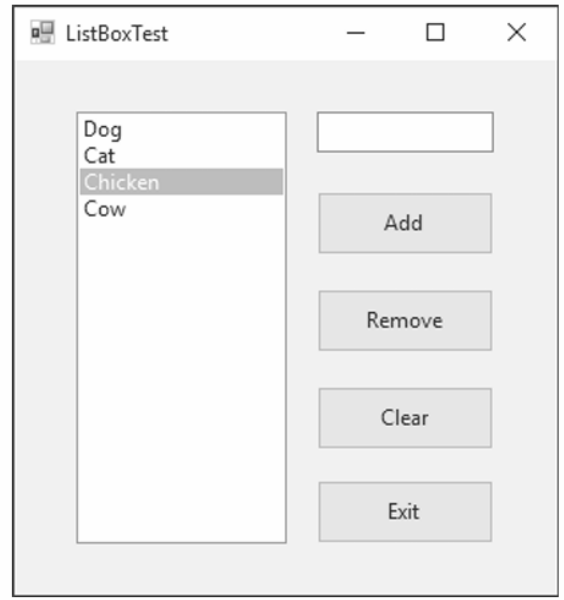
When clicked, Clear deletes all entries in displayListBox.

The user terminates the app by clicking Exit.

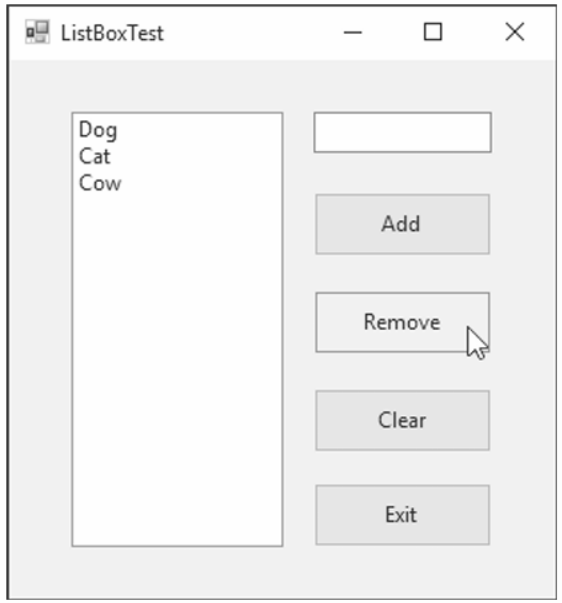
a) GUI after adding **Dog**, **Cat** and **Chicken** and before adding **Cow**



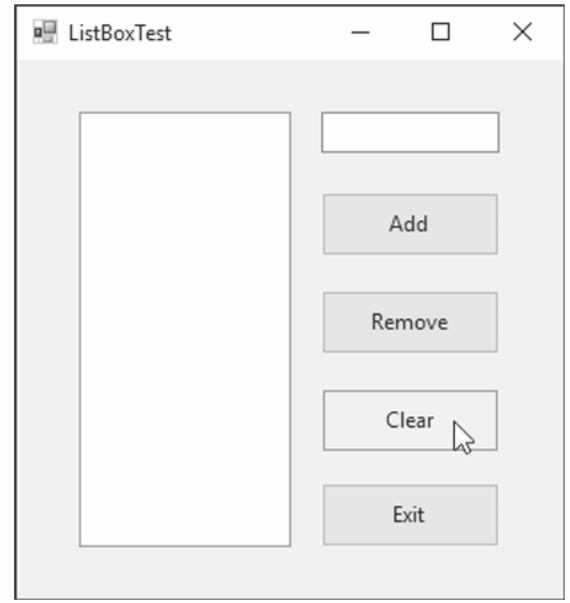
b) GUI after adding **Cow** and before deleting **Chicken**



c) GUI after deleting **Chicken**



d) GUI after clearing the `ListBox`



```
// add new item to ListBox (text from input TextBox)
// and clear input TextBox
private void addButton_Click(object sender, EventArgs e)
{
   displayListBox.Items.Add(inputTextBox.Text);
   inputTextBox.Clear();
}

// remove item if one is selected
private void removeButton_Click(object sender, EventArgs e)
{
   // check whether item is selected; if so, remove
   if (displayListBox.SelectedIndex != -1)
   {
      displayListBox.Items.RemoveAt(displayListBox.SelectedIndex);
   }
}

// clear all items in ListBox
private void clearButton_Click(object sender, EventArgs e)
{
   displayListBox.Items.Clear();
}
```