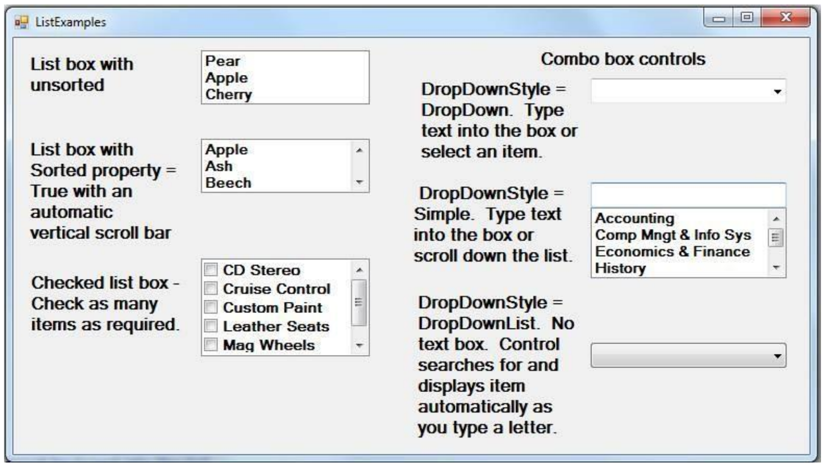
LISTS AND LOOPS

ListBox, CheckedListBox, and ComboBox

Visual basic provides three controls to enable an application user to select from a list of items: **ListBox**, **CheckedListBox**, and **ComboBox**.
This figure below displays different styles for these controls.



ListExamples
List box with unsorted
Pear
Apple
Cherry
List box with Sorted property = True with an automatic vertical scroll bar
Apple
Ash
Beech
Checked list box - Check as many items as required.
CD Stereo
Cruise Control
Custom Paint
Leather Seats
Mag Wheels
Combo box controls
DropDownStyle = DropDown. Type text into the box or select an item.
DropDownStyle = Simple. Type text into the box or scroll down the list.
Accounting
Comp Mngt & Info Sys
Economics & Finance
History
DropDownStyle = DropDownList. No text box. Control searches for and displays item automatically as you type a letter.

- **ListBox** – application user selects from the list – new items cannot be typed into the list.
- **CheckedListBox** – has a list of checkboxes inside a ListBox.
- **ComboBox** – has a **DropDownStyle** property that affects how the control works. The following are different types of ComboBox controls:
  - i) **DropDown** – includes a TextBox and ListBox together, you can type new items into the TextBox portion of the control, and the control does not consume very much space on the form. The list drops down when you click the drop-down arrow.
  - ii) **Simple** – you can type text into the TextBox at the top of the ComboBox control – the list displays in a ListBox format and then you can scroll up and down the listing.
  - iii) **DropDownList** – a ComboBox with **<u>no</u>** textbox into which to type new items – you cannot add new items. The control behaves in the following ways:
    1. As you type a letter, the control automatically searches for the first item in the listing that begins with that letter.
    2. The listing drops down when you click the drop-down arrow.

The Items Collection

The list of items that display one of these controls is called a **collection**.

- The collection is accessed through the **Items** property.
- The Items collection has properties and methods to make it easy to add items, remove items, and refer to individual items.
- The items in the list are **<u>numbered</u>** beginning with the number **0**.

Filling the List

Two ways to fill a List control:

1. If the list of items varies, the best way to fill a list is to store values into the list control's Items collection from a database table
2. If the list of items never changes or does not change much, you can add items at design type by clicking on the **Items** property – a **String Collection Editor** window will open and you type the items into the editor window.

Coding ListBox and ComboBox Controls

Items.Add Method

The **Items.Add** method is used to add an item to a list or ComboBox at run time.
Example commands to add a new country listing to the **CountryListBox:**

```
Dim NewCountryString  As String = "Kenya"
'This adds a value from a string memory variable.
CountryListBox.Items.Add(NewCountryString)
'This adds a literal value.
CountryListBox.Items.Add("Kenya")
'This adds a value from a TextBox control.
CountryListBox.Items.Add(CountryNameTextBox.Text)
```

The `Items.Add` method works for ComboBoxes in the same manner.

Exercise

Task: Code the Click event sub procedure for the **Add Country** button.

- Add the value typed into the **CountryComboBox** control's **Text** property to the **Items** collection of the ComboBox control.
- Check to ensure that the **Text** property is not the empty string.

```
    Private Sub AddCountryButton_Click(sender As Object, e As EventArgs)
Handles AddCountryButton.Click
        If CountryComboBox.Text.Trim <> String.Empty Then
            CountryComboBox.Items.Add(CountryComboBox.Text)
        Else
            MessageBox.Show("You must type a new Country name.",
               "Name Missing Error", MessageBoxButtons.OK,
               MessageBoxIcon.Error)
            CountryComboBox.Focus()
        End If
    End Sub
```

- In the above coding segment, if the **Text** property is not the **empty string**, the **Text** property value is added to the Items collection. The code does not handle the insertion of duplicate country name items.

Items.Insert Method

If a list is **<u>unsorted</u>**, you can use the **Items.Insert** method to specify exactly where within an Items collection to add a new item. You do this by specifying the number position in the collection – this is also called the index position of the new item.
This coding statement adds the item **Kenya** as the very first item in the **CountryComboBox** control.
`CountryComboBox.Items.Insert(0, "Kenya")`

If the **ListBox** or **ComboBox** control is **<u>sorted</u>**, then the **Insert** method appears to work exactly as the **Add** method – the new item is added at the location specified, but then control then immediately **<u>resorts</u>** the listing.

Items.Clear Method

The **Items.Clear** method will remove the contents of a ListBox or ComboBox.

```
 CountryComboBox.Items.Clear()
```

SelectedIndex and SelectedItem Properties
When you select an item from a list, the **index number** associated with the item is stored to the **SelectedIndex** property.

- When no item is selected, **SelectedIndex** = **-1**.
- You can select (highlight) an item in a list by storing a numeric value to the **SelectedIndex** property.

```
'This selects the 4th country item
CountryComboBox.SelectedIndex = 3
'This unselects any selected (highlighted) item
CountryComboBox.SelectedIndex = -1
```

When you select an item from a list, the **selected item** is stored to the **SelectedItem** property. This property can be used to display the selection elsewhere, for example, in a message box or to a TextBox control.

```
'Displays the selected current faculty member to a message box
MessageBox.Show(CountryListBox.SelectedItem.ToString)
```

To display an item in a list to another control, such as a textbox named **CountryInformationTextBox**, the command is shown below.

```
'Displays first item to the textbox
CountryInformationTextBox.Text = CountryListBox.Items(0).ToString()
```
The **ToString** method ensures the item (that is stored in the list as an **object**) converts to string for storage to the Text property of the TextBox control.

Exercise
Task: Code the Click event sub procedure to display **Selected Country** item.

- Use a message box to display the current selected item from the **CountryListBox** control.

```
Private Sub DisplaySelectedButton_Click(sender As Object, e As EventArgs)
Handles DisplaySelectedButton.Click
     'Use messagebox to display selected Country
     If CountryListBox.SelectedIndex = -1 Then
          MessageBox.Show("You must select an country display.", "No
          Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
     Else
          MessageBox.Show(CountryListBox.SelectedItem.ToString,
          "Current selection", MessageBoxButtons.OK,
          MessageBoxIcon.Information)
     End If
  End Sub
```

Items.Count Property
The **Items.Count** property stores a number equal to the number of items in a list or ComboBox Items collection. Its value is always 1 more than the maximum allowable **SelectedIndex** value. This property is used to display the number of items in a list.

```
Private Sub CountCountriesButton_Click(sender As Object, e As EventArgs)
Handles CountCountriesButton.Click
     'Display a message box with the count of the number of Countrys
     Dim MessageString As String = "Number of Countries: " &
     CountryListBox.Items.Count.ToString()
```

```
        Dim TitleString As String = "Count of Countries"
        MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK,
        MessageBoxIcon.Information)
    End Sub
```

Items.RemoveAt and Items.Remove Methods

To remove an individual item from a list, specify either the index or text of the item.

- **Items.RemoveAt** method – removes an item by specifying the **index position**.
- **Items.Remove** method – removes an item by specifying the **item name**.

The **RemoveAt** method fails if an item has not been selected for removal because the value of **SelectedIndex** = **-1** and you cannot remove a non-existent item – this throws an exception: **ArgumentOutOfRangeException**. When the **Remove** method is used and an item has not been selected for removal, the method simply does nothing.

```
        'Removes the first item from the list
        CountryComboBox.Items.RemoveAt(0)

         'Removes the item according to the value of IndexInteger
        CountryComboBox.Items.RemoveAt(IndexInteger)

        'Remove the currently selected item
        CountryComboBox.Items.RemoveAt(CountryComboBox.SelectedIndex)

        'Remove item by name where the item name is stored
        CountryComboBox.Items.Remove(CountryComboBox.SelectedItem)
```

Exercise

Task #1: Code the Click event sub procedure for the **Remove Country** button.

- Use the Items collection **Remove** method.
- Use a message box to ask the system user to confirm to remove the selected country as shown below.

```
    Private Sub RemoveCountryButton_Click(sender As Object, e As EventArgs) Handles
RemoveCountryButton.Click
        'Declare dialog result variable
        Dim ResponseDialogResult As DialogResult = MessageBox.Show("Remove the
        selected country?", "Remove ?", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)
        If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then
            'Remove the selected Country from the listing
            CountryComboBox.Items.Remove(CountryComboBox.SelectedItem)
           'This next line of code clears the Text property of the ComboBox - this
           'is automatic for the Remove method, but not for the last item in a list.
           CountryComboBox.Text = String.Empty
        End If
    End Sub
```

Task #2: Code for the Click event sub procedure for the **Remove At Country** button.

- Use the Items collection **RemoveAt** method.
- Use a message box to ask the system user to confirm to remove the selected country.
- Use a Try-Catch block to trap the possible **ArgumentOutOfRangeException** that will be thrown if an item is not selected.

```
  Private Sub RemoveAtCountryButton _Click(sender As Object, e As
EventArgs) Handles RemoveAtCountryButton.Click
        'Try to remove the Country if one is selected
      Try
      'Declare dialog result variable
        Dim ResponseDialogResult As DialogResult =
        MessageBox.Show("Remove the selected Country?", "Remove ?",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button2)
        If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then
           'Remove the selected country from the listing
            CountryComboBox.Items.RemoveAt(CountryComboBox.SelectedIndex)
            CountryComboBox.Text = String.Empty
        End If
        Catch ex As ArgumentOutOfRangeException
            MessageBox.Show("You must select a country to remove.",
            "No Selection Was Made", MessageBoxButtons.OK,
             MessageBoxIcon.Error)
        End Try
    End Sub
```

ListBox and ComboBox Events

Code can be written for several events of ListBoxes and ComboBoxes. These include:

i). **SelectedIndexChanged Event**. Occurs when a system user makes a new selection for a list or ComboBox.

ii). **TextChanged Event**. As a system user types characters into the TextBox portion of a ComboBox, this event is triggered for each character typed. ListBoxes do not have this event.

LOOPS (ITERATION STRUCTURES)

These structures allow for execution of one or more lines of code repetitively. There are two types of loops:

- **Condition-controlled**. In VB, these are **Do Loops** – loops that execute over and over until a specific condition occurs.
- **Count-controlled**. In VB, these are **For…Next Loops** – loops that execute for a specified number of times.

i) Do … Loop

Do loop is used to execute a block of statements indefinite number of times. The loop is executed either While the *condition is true* or *Until the condition becomes True*. The loops can use either a **While** or **Until** option to test the condition i.e. there are the mirror image of each other and you need to use the approach that you like best.

To execute a block of statements while the condition is true, the following syntax is used:

```
Do While Condition
     Statement(s)
Loop
```

The statements can execute any number of times as long as the condition is true. The number of iterations need not be known before the loop starts. If the condition is initially **False**, the statement may never execute. To execute a block of statements until the condition becomes true, the following syntax is used:

```
Do Until Condition
     Statement(s)
Loop
```

Do Loops can have the **condition test** (**stopping condition**) at either the top or bottom of a loop. The next loops will always execute the body of the loop at least one time because the stopping condition is not tested until the end of the loop.

- **Do…Loop While**

```
Do
     Statement(s)
Loop While Condition
```

- **Do…Loop Until**

```
Do
     Statement(s)
Loop Until Condition
```

Searching a ListBox or ComboBox

One problem with adding new country to the **CountryComboBox** control is that you can end up with duplicate Countrys. The following procedure will prevent duplicates to be added into the ComboBox control.

```
Private Sub AddCountryButton_Click(sender As Object, e As EventArgs)
Handles AddCountryButton.Click
     Dim FoundBoolean As Boolean = False
     Dim IndexInteger As Integer = 0
     Do Until FoundBoolean = True Or IndexInteger =
     CountryComboBox.Items.Count
          If CountryComboBox.Text.Trim.ToUpper =
          CountryComboBox.Items(IndexInteger).ToString.Trim.ToUpper Then
                FoundBoolean = True
           Else
                'Add 1 to index value
                IndexInteger += 1
            End If
       Loop  'Loop back and try again
       'Now decide whether to add item or not
       If FoundBoolean = False And CountryComboBox.Text.Trim <>
       String.Empty Then
             CountryComboBox.Items.Add(CountryComboBox.Text.Trim)
        Else
            MessageBox.Show("Duplicate or Invalid Country Name",
             "Duplicate Data Error", MessageBoxButtons.OK,
             MessageBoxIcon.Exclamation)
            CountryComboBox.Focus()
            CountryComboBox.SelectAll()
        End If
    End Sub
```

ii) For … Next

A **For…Next** loop is a **count-controlled** loop – the code executes a set number of times based on a **loop index**.

Syntax:

```
For IndexInteger = StartInteger To EndInteger Step [IncrementInteger]
      Statement(s)
Next Counter
```

The increment can either be positive or negative. If positive, then Start must be less or equal to End argument and if negative, then Start must be greater or equal to End argument for the loop to execute. In most cases, the **For … Loop** can be represented with other repetition structures e.g. the Do…While loop equivalent is:

```
Counter = Start
Do While Counter <= End
     statements
     Counter = Counter + Increment
Loop
```

Example
The following program generates and displays even numbers between two and twenty in a listbox control using a count-controlled loop after click of a button control.

```
 Private Sub CountButton_Click(sender As Object, e As EventArgs)
Handles CountButton.Click
    Dim IndexInteger As Integer
    For IndexInteger = 2 To 20 Step 2
        CountListBox.Items.Add(IndexInteger.ToString)
    Next IndexInteger
    'Control transfers to here when the loop ends
    MessageBox.Show("Finished Counting")
End Sub
```

Other example **For…Next** statements:

```
For IndexInteger = 0 To (ComboBox.Items.Count - 1)

For InterestRateDecimal = 0.05D To 0.25D Step 0.05D

For BackwardsIndexInteger = 10 To 0 Step -1
```

Nested Loops (A Loop inside a Loop)
This example shows a **For...Next** loop inside another **For...Next** loop. The inner loop must be completely contained within the outer loop. The inner loop executes a "normal number" of times for each execution of the outer loop.

Example: The following code will generate and display a ten by ten multiplication table in a listbox control

```
Private Sub MultiplicationTableButton_Click(sender As Object, e As
EventArgs) Handles MultiplicationTableButton.Click
     Dim ProductString As String
     Dim OuterIndexInteger As Integer
     Dim InnerIndexInteger As Integer
     'Start of the outer loop
     For OuterIndexInteger = 1 To 10
          ProductString = String.Empty
          'Start of the inner loop - each time we get here the inner loop starts
          'all over again
         For InnerIndexInteger = 1 To 10
                ProductString &= (InnerIndexInteger * OuterIndexInteger).ToString
                & ControlChars.Tab
         Next InnerIndexInteger
         'Exited inner loop - control transfers to outer loop
```

```
            MultiplicationTableListBox.Items.Add(ProductString.ToString)
        Next OuterIndexInteger
        'Control transfers here when the outer loop ends
End Sub
```

Alternatively: The following will generate and display a ten by ten multiplication table in an output window

```
Private Sub MultiplicationTableButton2_Click(sender As Object, e As
EventArgs) Handles MultiplicationTableButton2.Click
   Dim Factor1, Factor2 As Integer
   For Factor1 = 1 To 10
       For Factor2 = 1 To 10
          Debug.Write((Factor1 * Factor2).ToString() & ControlChars.Tab)
       Next
       Debug.WriteLine("")
    Next
End Sub
```

Exit Statement
The **Exit For** statement is used to exit a **For...Next** loop before the ending value is reached. Example:

```
For IndexInteger = 1 To 10
    If InputTextBox.Text = String.Empty Then 'Blank input
        MessageBox.Show("Input is invalid")
        Exit For
    End If
    'Other statements to process if the input is valid
Next IndexInteger
```

There is also **Exit Do**, **Exit While** and **Exit If** statements. For the most part, using any **Exit** statement is unnecessary if your code is constructed properly.