Windows Basic Form's Controls, Control's Properties And Events

Objectives:

- To learn how to deal with controls programmatically and by using property window.
- To learn how to associate different controls together.
- To understand the purpose and usefulness of different controls.
- To learn how to attach events to different controls and invoke them.

MUST Read:

1. Windows Form Button Control:

The Windows Forms **Button** control allows the user to click it to perform an action. When the button is clicked, it looks as if it is being pushed in and released. Whenever the user clicks a button, the **Click** **event** handler is invoked. You place code in the **Click** event handler to perform any action you choose. The text displayed on the button is contained in the **Text property**. The appearance of the text is controlled by the **Font property** and the **TextAlign property**. The Button control can also display images using the **Image** and **ImageList properties**.

1.1. Designating a Windows Forms Button as the Accept Button

On any Windows Form you can designate a **Button** control to be the accept button, also known as the **default button**. Whenever the user presses the ENTER key, the default button is clicked regardless of which other control on the form has the focus. (The exceptions to this are when the control with focus is another button — in that case, the button with the focus will be clicked — or a multiline text box, or a custom control that traps the ENTER key.)

To designate the accept button in the designer

1. Select the form on which the button resides.

2. In the Properties window, set the form's **AcceptButton property** to the **Button** control's name.

1.2. Designating a Windows Forms Button as the Cancel Button

On any Windows Form you can designate a Button control to be the cancel button. A cancel button is clicked whenever the user presses the ESC key, regardless of which other control on the form has the focus. Such a button is usually programmed to allow the user to quickly exit an operation without committing to any action.

To designate the cancel button in the designer

1. Select the form on which the button resides.

2. In the Properties window, set the form's **CancelButton property** to the **Button** control's name.

1.3. Responding to Windows Forms Button Clicks

The most basic use of a Windows Forms **Button** control is to run some code when the button is clicked. Clicking a **Button** control also generates a number of other events, such as the **MouseEnter**, **MouseDown**, and **MouseUp** **events**. If you intend to attach event handlers or these related events, be sure that their actions do not conflict. If the user attempts to double-click the Button control, each click will be processed separately; that is, the control does not support the double-click event.

To respond to a button click

In the button's **Click** **event** handler write the code to run. Button1_Click must be bound to the control; For example add the below given code to **button1_Click** and execute to see the result;

Sample Code: MessageBox.Show("button1 was clicked");

2. Windows Forms TextBox Control

Windows Forms text boxes are used to get input from the user or to display text. The **TextBox** control is generally used for editable text, although it can also be made read-only. Text boxes can display multiple lines, wrap text to the size of the control, and add basic formatting. The **TextBox** control provides a single format style for text displayed or entered into the control. To display multiple types of formatted text, use the **RichTextBox** control. The text displayed by the control is contained in the **Text** **property**. By default, you can enter up to 2048 characters in a text box. If you set the **MultiLine** **property** to **true**, you can enter up to 32 KB of text. The **Text** **property** can be set at design time with the Properties window, at run time in code, or by user input at run time. The current contents of a text box can be retrieved at run time by reading the **Text** property.

2.1. Controlling the Insertion Point in a Windows Forms TextBox Control

When a Windows Forms **TextBox** control first receives the focus, the default insertion within the text box is to the left of any existing text. The user can move the insertion point with the keyboard or the mouse. If the text box loses and then regains the focus, the insertion point will be wherever the user last placed it. In some cases, this behavior can be disturbing to the user. In a word processing application, the user might expect new characters to appear after any existing text. In a data entry application, the user might expect new characters to replace any existing entry. The **SelectionStart** and **SelectionLength** **properties** allow you to modify the behavior to suit your purpose.

To control the insertion point in a TextBox control

1. Set the **SelectionStart** property to an appropriate value. Zero places the insertion point immediately to the left of the first character.

2. (Optional) Set the **SelectionLength** property to the length of the text you want to select.

The **TextBox1_Enter** event handler must be bound to the control.

2.2. Creating a Password Text Box with the Windows Forms TextBox Control

A password box is a Windows Forms text box that displays placeholder characters while a user types a string.

To create a password text box

1. Set the **PasswordChar** property of the **TextBox** control to a specific character.

The **PasswordChar** property specifies the character displayed in the text box. For example, if you want asterisks displayed in the password box, specify * for the**PasswordChar** property in the Properties window. Then, regardless of what character a user types in the text box, an asterisk is displayed.

2. (Optional) Set the **MaxLength** property.

The property determines how many characters can be typed in the text box. If the maximum length is exceeded, the system emits a beep and the text box does not accept any more characters.

Security Note Using the **PasswordChar** property on a text box can help ensure that other people will not be able to determine a user's password if they observe the user entering it.

2.3. Creating a Read-Only Text Box (Windows Forms)

You can transform an editable Windows Forms text box into a read-only control. For example, the text box may display a value that is usually edited but may not be currently, due to the state of the application.

To create a read-only text box

- Set the **TextBox** control's **ReadOnly** property to **true**. With the property set to **true**, users can still scroll and highlight text in a text box without allowing changes. A Copy command is functional in a text box, but Cut and Paste commands are not.

Note The **ReadOnly** property only affects user interaction at run time. You can still change text box contents programmatically at run time by changing the **Text** property of the text box.

3. Windows Forms Label Control

Windows Forms Label controls are used to display text or images that cannot be edited by the user. They are used to identify objects on a form — to provide a description of what a certain control will do if clicked, or to display information in response to a run-time event or process in an application. For example, you can use labels to add descriptive captions to text boxes, list boxes, combo boxes, and so on. You can also write code that changes the text displayed by a label in response to events at run time. For example, if your application takes a few minutes to process a change, you can display a processing-status message in a label. Because the **Label** control cannot receive the focus, it can also be used to create access keys for other controls. An access key allows a user to select the other control by pressing the **ALT** key with the access key. The caption displayed in the label is contained in the **Text** property. The **Alignment** property allows you to set the alignment of the text within the label.

4. Windows Forms CheckBox Control

The Windows Forms **CheckBox** control indicates whether a particular condition is on or off. It is commonly used to present a Yes/No or True/False selection to the user. You can use check box controls in groups to display multiple choices from which the user can select one or more. The check box control is similar to the radio button control in that each is used to indicate a selection that is made by the user. They differ in that only one radio button in a group can be selected at a time. With the check box control, however, any number of check boxes may be selected. A check box may be connected to elements in a database using simple data binding. Multiple check boxes may be grouped using the **GroupBox** control. This is useful for visual appearance and also for user interface design, since grouped controls can be moved around together on the form designer. The **CheckBox** control has two important properties, **Checked** and **CheckState**. The **Checked** property returns either **true** or **false**. The **CheckState** property returns either **CheckState.Checked** or **CheckState.Unchecked**; or, if the **ThreeState** property is set to **true**, **CheckState** may also return **CheckState.Indeterminate**. In the indeterminate state, the box is displayed with a dimmed appearance to indicate the option is unavailable. Whenever a user clicks a Windows Forms CheckBox control, the Click event occurs. You can program your application to perform some action depending upon the state of the check box.

4.1. Responding to Windows Forms CheckBox Clicks

Whenever a user clicks a Windows Forms **CheckBox** control, the **Click** event occurs. You can program your application to perform some action depending upon the state of the check box.

To respond to CheckBox clicks

In the **Click** event handler, use the **Checked** property to determine the control's state, and perform any necessary action.

Sample Code:

```
private void checkBox1_Click(object sender, System.EventArgs e)
{
// The CheckBox control's Text property is changed each time the
// control is clicked, indicating a checked or unchecked state.
switch(checkBox1.CheckState)
{
case CheckState.Checked:
checkBox1.Text = "Checked";
break;
case CheckState.Unchecked:
checkBox1.Text = "Unchecked";
break;
case CheckState.Indeterminate:
// Code for indeterminate state.
break;
}
}
```

5. Panel Control (Windows Forms)

Windows Forms **Panel** controls are used to provide an identifiable grouping for other controls. Typically, you use panels to subdivide a form by function. For example, you may have an order form that specifies mailing options such as which overnight carrier to use. Grouping all options in a panel gives the user a logical visual cue. At design time all the controls can be moved easily — when you move the **Panel** control, all its contained controls move, too. The controls grouped in a panel can be accessed through its **Controls** property. The **Panel** control is similar to the **GroupBox** control; however, only the **Panel** control can have scroll bars, and only the **GroupBox** control displays a caption. To display scroll bars, set the **AutoScroll** property to **true**. You can also customize the appearance of the panel by setting the **BackColor**, **BackgroundImage**, and **BorderStyle** properties.The **BorderStyle** property determines if the panel is outlined with no visible border (None), a plain line (FixedSingle), or a shadowed line (Fixed3D).

6. Windows Forms ComboBox Control

The Windows Forms **ComboBox** control is used to display data in a drop-down combo box. By default, the **ComboBox** control appears in two parts: the top part is a text box that allows the user to type a list item. The second part is a list box that displays a list of items from which the user can select one. The **SelectedIndex** property returns an integer value that corresponds to the selected list item. You can programmatically change the selected item by changing the **SelectedIndex** value in code; the corresponding item in the list will appear in the text box portion of the combo box. If no item is selected, the **SelectedIndex** value is -1. If the first item in the list is selected, then the **SelectedIndex** value is 0. The **SelectedItem** property is similar to **SelectedIndex**, but returns the item itself, usually a string value. The **Items.Count** property reflects the number of items in the list, and the value of the **Items.Count** property is always one more than the largest possible **SelectedIndex** value because **SelectedIndex** is zero-based. To add or delete items in a **ListBox** control, use the **Items.Add**, **Items.Insert**, **Items.Clear** or **Items.Remove** method. Alternatively, you can add items to the list by using the **Items** property in the designer.

7. Windows Forms RadioButton Control

Windows Forms **RadioButton** controls present a set of two or more mutually exclusive choices to the user. While radio buttons and checkboxes may appear to function similarly, there is an important difference: when a user selects a radio button, the other radio buttons in the same group cannot be selected as well. In contrast, any number of check boxes can be selected. Defining a radio button group tells the user, "Here is a set of choices from which you can choose one and only one." When a **RadioButton** control is clicked, its **Checked** property is set to **true** and the **Click** event handler is called. The **CheckedChanged** event is raised when the value of the **Checked** property changes. If the **AutoCheck** property is set to **true** (the default), when the radio button is selected all others in the group are automatically cleared. This property is usually only set to **false** when validation code is used to make sure the radio button selected is an allowable option. The text displayed within the control is set with the **Text** property, which can contain access key shortcuts. An access key allows a user to "click" the control by pressing the ALT key with the access key. The **RadioButton** control can appear like a command button, which appears to have been depressed if selected, if the **Appearance** property is set to **Appearance.Button**. Radio buttons can also display images using the **Image** and **ImageList** properties. Windows Forms **RadioButton** controls are designed to give users a choice among two or more settings, of which only one can be assigned to a procedure or object. For example, a group of **RadioButton** controls may display a choice of package carriers for an order, but only one of the carriers will be used. Therefore only one **RadioButton** at a time can be selected, even if it is a part of a functional group. Group the radio buttons by drawing them inside a container such as a Panel control, a

GroupBox control, or a form. All radio buttons that are added directly to a form become one group. To add separate groups, you need to place them inside panels or group boxes.

1. Drag a **GroupBox** or **Panel** control from the Windows Forms tab on the Toolbox onto the form.

2. Draw **RadioButton** controls on the **GroupBox** or **Panel** control.

Grouping Controls with the Windows Forms Panel Control

Windows Forms **Panel** controls are used to group other controls. There are three reasons to group controls. One is visual grouping of related form elements for a clear user interface; another is programmatic grouping of radio buttons; the last is for moving the controls as a unit at design time.

To create a group of controls

1. Drag a **Panel** control from the **Windows Forms** tab of the Toolbox onto a form.

2. Add other controls to the panel, drawing each inside the panel.

3. If you have existing controls that you want to enclose in a panel, you can select all the controls, cut them to the Clipboard, select the **Panel** control, and then paste them into the panel. You can also drag them into the panel.

4. (Optional) If you want to add a border to a panel, set its **BorderStyle** property. There are three choices: **Fixed3D**, **FixedSingle**, and **None**.

Setting the Background of a Panel

A Windows Forms **Panel** control can display both a background color and a background image. The **BackColor** property sets the background color for the contained controls, such as labels and radio buttons. If the **BackgroundImage** property is not set the **BackColor** selection will fill the entire panel. If the **BackgroundImage** property is set, the image will be displayed behind the contained controls.

To set the background in the Windows Forms Designer

1. Select the **Panel** control.

2. In the Properties window, click the arrow button next to the **BackColor** property to display a window with three tabs.

3. Select the **Custom** tab to display a palette of colors. Select the **Web** or **System** tab to display a list of predefined names for colors.

4. Select a color.

5. In the Properties window, click the arrow button next to the **BackgroundImage** property to display the **Open** dialog box.

6. Select the file you want to display.

8. GroupBox Control (Windows Forms)

Windows Forms GroupBox controls are used to provide an identifiable grouping for other controls. Typically, you use groupboxes to subdivide a form by function. Grouping all options in a group box gives the user a logical visual cue. The **GroupBox** control is similar to the **Panel** control; however, only the **GroupBox** control displays a caption, and only the **Panel** control can have scroll bars. The groupbox's caption is defined by the **Text** property.

9. Grouping Controls with the Windows Forms GroupBox Control

Windows Forms **GroupBox** controls are used to group other controls. There are three reasons to group controls. One is visual grouping of related form elements for a clear user interface; another is programmatic grouping of radio buttons; the last is for moving the controls as a unit at design time.

To create a group of controls

1. Draw a **GroupBox** control on a form.

2. Add other controls to the group box, drawing each inside the group box. If you have existing controls that you want to enclose in a group box, you can select all the controls, cut them to the Clipboard, select the **GroupBox** control, and then paste them into the group box. You can also drag them into the group box.

3. Set the **Text** property of the group box to an appropriate caption.

10. Windows Forms ListBox Control

A Windows Forms **ListBox** control displays a list of items from which the user can select one or more. If the total number of items exceeds the number that can be displayed, a scroll bar is automatically added to the **ListBox** control. When the **MultiColumn** property is set to **true**, the list box displays items in multiple columns and a horizontal scroll bar appears. When the **MultiColumn** property is set to **false**, the list box displays items in a single column and a vertical scroll bar appears. When **ScrollAlwaysVisible** property is set to **true**, the scroll bar appears regardless of the number of items. The **SelectionMode** property determines how many list items can be selected at a time. The **SelectedIndex** property returns an integer value that corresponds to the first selected item in the list box. You can programmatically change the selected item by changing the **SelectedIndex** value in code; the corresponding item in the list will appear highlighted on the Windows Form. If no item is selected, the SelectedIndex value is -1. If the first item in the list is selected, then the SelectedIndex value is 0. When multiple items are selected, the SelectedIndex value reflects the selected item that appears first in the list. The SelectedItem property is similar to SelectedIndex, but returns the item itself, usually a string value. The **Items.Count** property reflects the number of items in the list, and the value of the Items.Count property is always one more than the largest possible SelectedIndex value because SelectedIndex is zero-based. To add or delete items in a ListBox control, use the **Items.Add**, **Items.Insert**, **Items.Clear** or **Items.Remove** method. Alternatively, you can add items to the list by using the **Items property** at design time.

Exercise 1: Simple Calculator

1. Create Windows Form Application Project and save into directory Zfolder/SWE344/Lab8/Exercise1.

2. Drag and Drop 3 text boxes (2 for input and 1 for output), 3 labels (2 along with input text boxes and 1 along with output text box) and 5 buttons using toolbox window.

3. Using property window make following changes:

    a. Text of Label1 as “**First Value**” and Name as “**FirstValue**”.

    b. Text of Label2 as “**Second Value**” and Name as “**SecondValue**”.

    c. Text of Label3 as “**Result**” and Name as “**Result**”.

    d. Text of button1 as **“+”** symbol and Name as “**Plus**”.

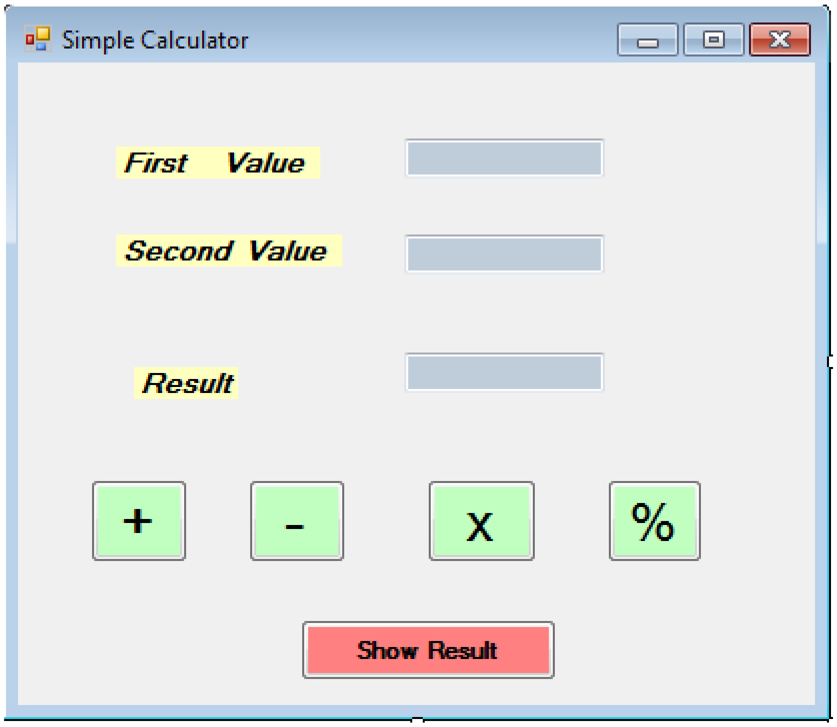
    e. Text of button2 as “-“symbol and Name as “**Minus**”.

    f. Text of button3 as “**x”** symbol and Name as “**Multiply**”.

    g. Text of button4 as “%” symbol and Name as “**Divide**”.

    h. Text of button5 as “**Show Result**” and Name as “**ShowResult**”.

5. The GUI should look as given below



6. At startup of the calculator all the buttons with captions +, -, **X** and **%** and the output textbox **Result** should be disabled. These buttons should also be disabled if any of the text boxed is empty.

7. After entering the numbers in both the text boxes (**First Value** and **Second Value**), the entire function button +, -, **X** and **%** should get enabled.

8. On clicking any function button, the calculator should calculate the result.

9. On Clicking "**ShowResult**" button the calculated result should be in the **Result** Text Box.

Exercise 2: Adding/Removing Elements to/from ListBox and attaching ListBox to Button:

1. Create Windows Form Application Project and save into directory Zfolder/SWE344/Lab8/Exercise2.

2. Drag and Drop 2 List Boxes and 1 button to the form using toolbox window.

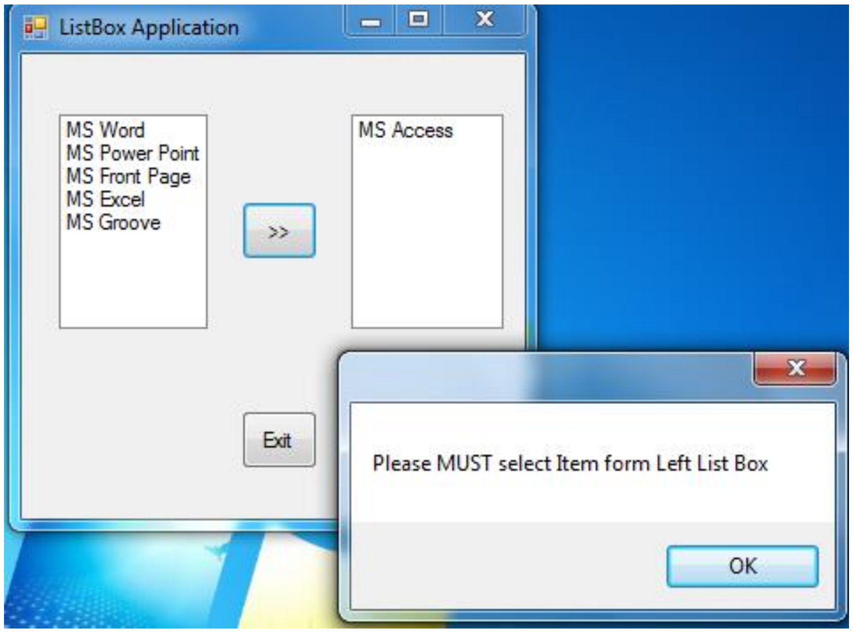
3. Using property window:

   a. Rename the left list box to “**LeftListBox**”.

   b. Rename the right list box to “**RightListBox**”.

   c. Change the button caption to “>>”.

4. The GUI should look as given.



Please MUST select Item form Left List Box

5. Using property window:

   a. Using Item property of list box Add MS Word, MS Power Point, MS Access, MS Front Page, MS Excel and MS Groove to the left **ListBox**.

   b. Allow the user to select only one item at a time by setting the **SelectionMode** property.

6. On clicking button the selected item of left **ListBox** should be added to the right **ListBox** and at the same time the item should be removed from the left list box.

7. If the user clicks the button(>>) without selecting any item in left list box, a message box displaying “Please MUST select Item form Left List Box “.

8. Add Exit button as shown in the GUI to close the form.

Exercise 3: Adding/Removing Elements to/from treeView:

1. Create Windows Form Application Project and save into directory Zfolder/SWE344/Lab8/Exercise3.

2. Drag and Drop 1 treeView control and 4 buttons to the form using toolbox window.

3. Using property window make following changes:

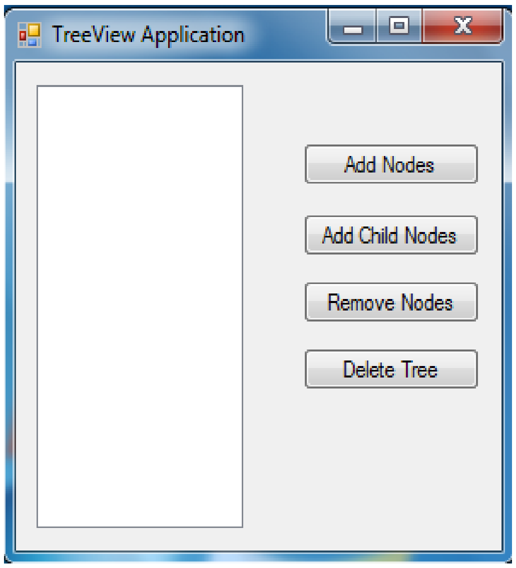
   a. Text of butoon1 as "**Add Nodes**" and Name as "**AddNodes**".

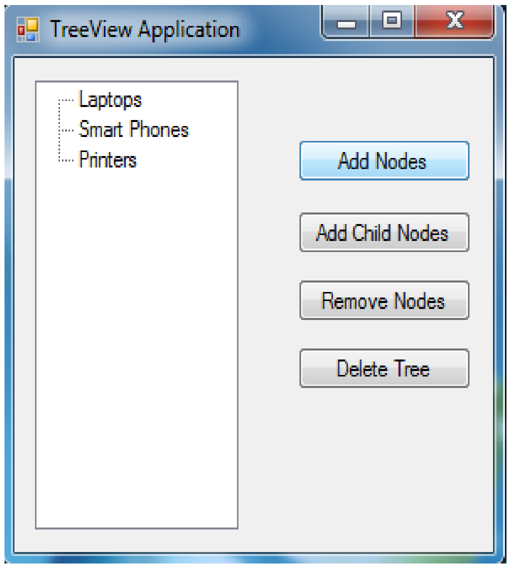
   b. Text of butoon2 as "**Add Child Nodes**" and Name as "**AddChildNodes**".

   c. Text of butoon3 as "**Remove Nodes**" and Name as "**RemoveNodes**".

   b. Text of butoon4 as "**Delete Tree**" and Name as "**DeleteTree**".

4. The GUI should look as given below:





5. On clicking "**Add Nodes**" button the following items should appear in the treeView control:

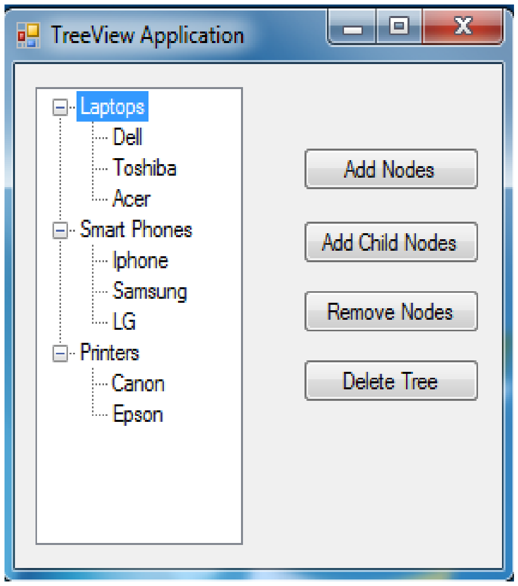
   Laptops
   Smart Phones
   Printers

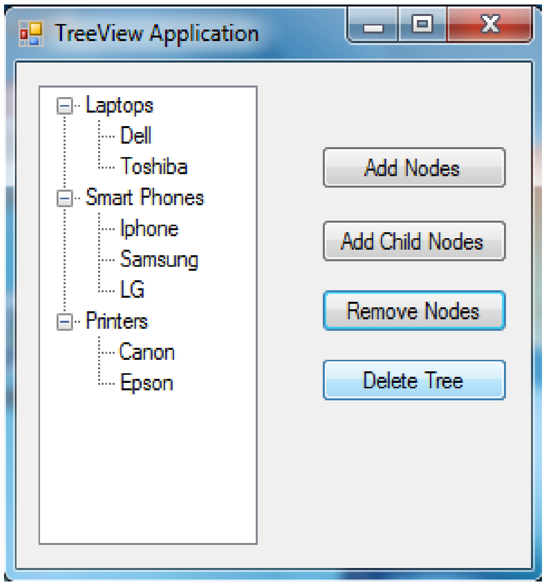
6. On clicking “**Add Child Nodes**” button the following items should be added under each node in the treeView control:

Laptops
- **Dell**
- **Toshiba**
- **Acer**

Smart phones
- **iPhone**
- **Samsung**
- **LG**

Printers
- **Canon**
- **Epson**





7. On clicking “**Remove Nodes**” button the selected items should be deleted from treeView.

8. On clicking “**Delete Tree**” button all the items should be deleted from treeView.

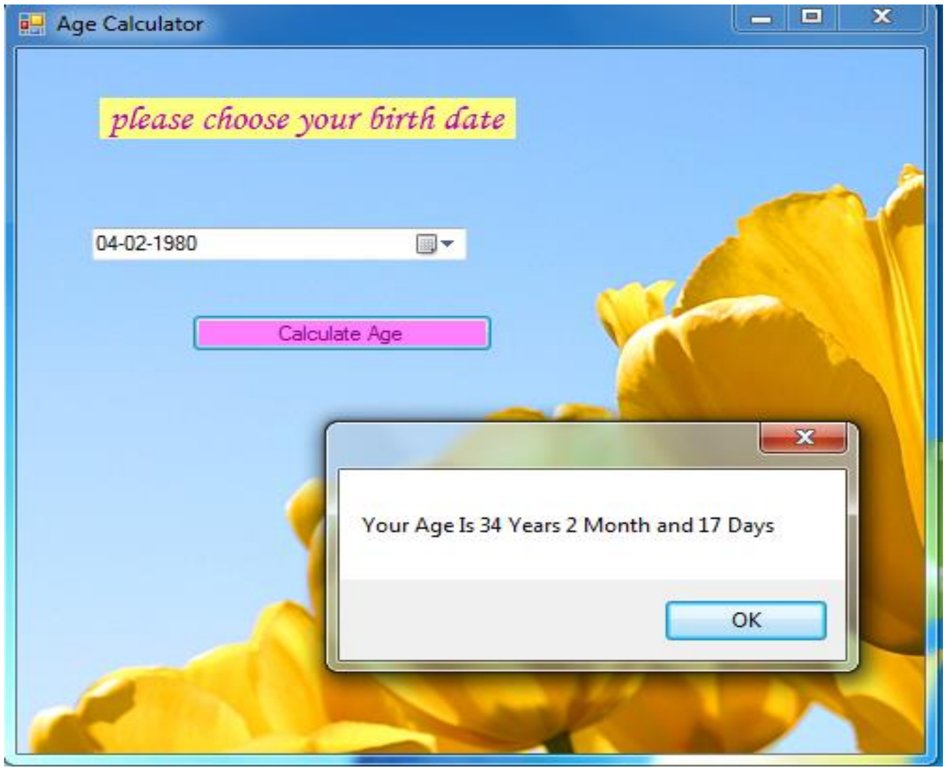
Exercise 4: Age Calculator by using dateTimePicker control

1. Create Windows Form Application Project and save into directory Zfolder/SWE344/Lab8/Exercise4.

2. Drag and Drop 1 dateTimePicker control and 1 button to the form using toolbox window.

3. Using property window rename the button to "**CalculateAge**" and Text as "**Calculate Age**".

4. The GUI should look as given below:



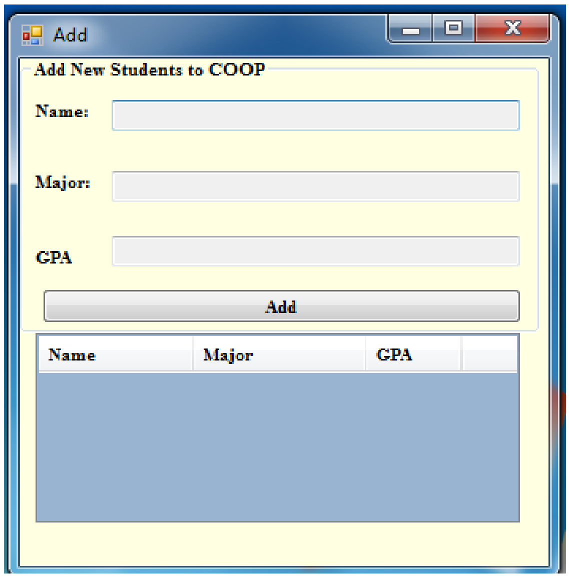
Your Age Is 34 Years 2 Month and 17 Days

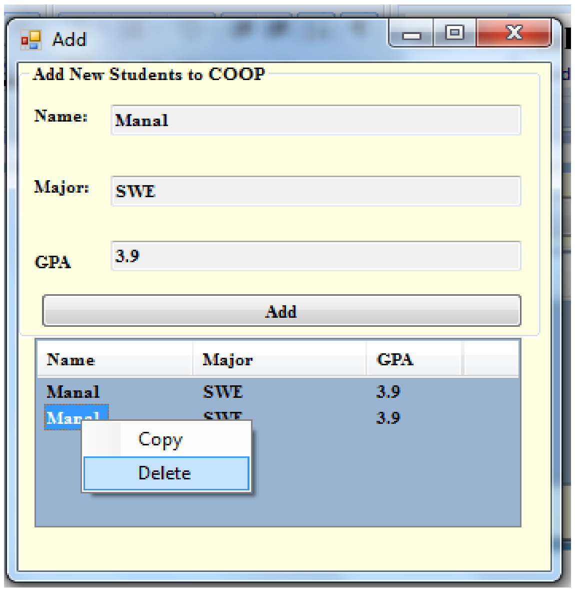
5. On click of "Calculate Age" button, the message should be displayed showing the age of that person.

Exercise 5: Adding/Removing Elements to/from listView with help of contextMenu controls

1. Create Windows Form Application Project and save into directory Zfolder/SWE344/Lab8/Exercise5.

2. The GUI should look as given below:





Name: Manal
Major: SWE
GPA 3.9

Name	Major	GPA
Manal	SWE	3.9
Manal	SWE	3.9

3. Using Property window, make following changes in **listView1** control's property:

   a. **Columns** property should be added as Name, Major and GPA. These are **Text** of **ColumnHeaders**.

   b. **View** property must be chosen **"Details"**.

   c. **ContextMenuStrip** property must be chosen **ContextMenuStrip1**.

4. On click of "Add" button, the information of textBoxes should be added in listView control.

5. On selection of "Name" column in the listView control, the context menu should appear with "Copy" and "Delete".

6. On choosing "Copy" operation from context menu the row should be duplicated in listView control.

7. On choosing "Delete" operation from context menu the row should be removed from listView control.