CS608 Lecture Notes

Visual Basic.NET Programming

Introduction to visual Basic.NET

Language Components – Graphical Elements Reference

These notes are for Reference Only! And will not be covered in class!

These notes are intended a reference for reviewing VB.NET graphical controls and forms on your own time and when needed for the projects

(Part IV of IV)

(Lecture Notes 1D)

Prof. Abel Angel Rodriguez

Chapter 4 (Cont) Graphical Elements Reference

4.5 Graphical Elements of .NET Framework – System.Windows.Forms Namespace

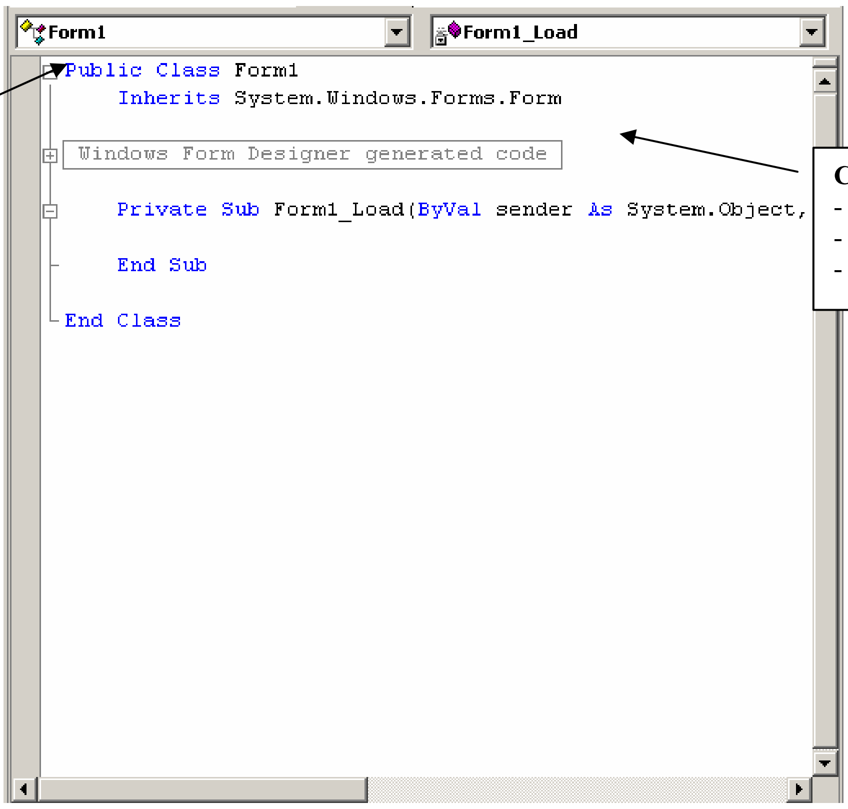
- To create Windows Applications, we need to use graphical elements.
- The **System.Windows.Forms** namespace contains numerous GUI **classes** for us to use with our programs. Such as Labels, TextBox, Button, ListBox, MessageBox etc.
- You must import the **System.Windows.Forms** namespace to make these graphical classes available to the compiler

4.5.1 Form Class

- Forms are the class objects used as the foundation for creating the User Interface (UI) in a windows application
- Forms are created using the IDE Form Designer.
- As with most Visual Basic Objects, Forms have a visible & invisible part. The visible part is the graphical Form that you see and the invisible part is the Code written using the Code Editor Window.

Form Class Basics

- In Visual Basics.NET, when you create a Form, you are actually creating a **Form Class**.
- When you use the *Code Editor Window* to view the invisible part of a Form you will see the Class declaration:



```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub Form1_Load(ByVal sender As System.Object,

    End Sub

End Class
```

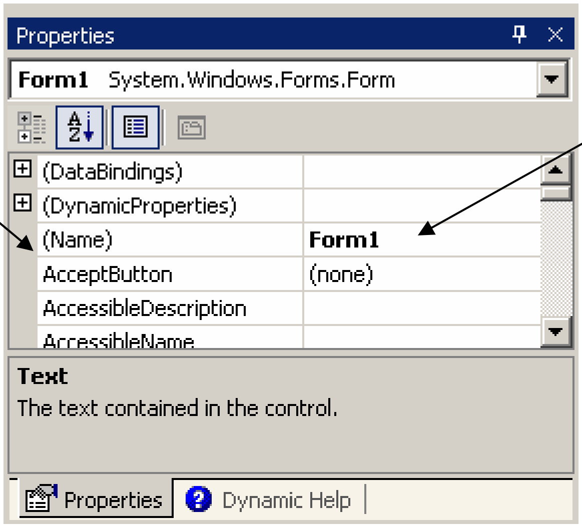
Class Declaration

Class Body Contains:
- Form Properties
- Form Methods
- Form Event-Procedures

Form Class Properties

- Forms are Objects therefore they contain properties.
- The Form Properties can be viewed and modified using the ***Property Window***

Property Name



Property Value

- The table below is a snapshot of some of the most commonly used Properties:

Common Form Class Properties:

BackColor	Gets or sets the background color for the control
BackgroundImage	Gets or sets the background image displayed in the control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
FormBorderStyle	Gets or sets the border style of the form.
Name	Gets or sets the name of the control.
Size	Gets or sets the size of the form.
StartPosition	Gets or sets the starting position of the form at run time.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Text	Gets or sets the text associated with this control.
Visible	Gets or sets a value indicating whether the control is displayed.

Form Class Methods

- The Form Class contains many Methods. If you go to the ONLINE HELP and search for Form Methods, you will see a table with all the methods of the **Form Class**.
- The table below is a snapshot of some of the most commonly used Methods:

Common Form Class Methods:

Close	Closes the form.
Hide (inherited from **Control**)	Conceals the control from the user.
Show (inherited from **Control**)	Displays the control to the user.
ShowDialog	Shows the form as a modal dialog box.

Form Class Events & Event-Procedures

- Events are actions taken upon the Form Object by the User.
- These actions automatically execute a specialized method called an ***Event-Procedure***
- The Form Class contains a list of these *Event-Procedures*. Go to the ONLINE-HELP for a complete listing.
- The table below is a snapshot of some of the most commonly used Event-Procedures:

Form Class Public Events:

Click	Occurs when the control is clicked.
Closed	Occurs when the form is closed.
Closing	Occurs when the form is closing.
Deactivate	Occurs when the form loses focus and is not the active form.
DoubleClick	Occurs when the control is double-clicked.
Enter	Occurs when the control is entered.
GotFocus	Occurs when the control receives focus.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
Load	Occurs before a form is displayed for the first time.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
Resize	Occurs when the control is resized.
TextChanged	Occurs when the Text property value changes.

Using Forms in Your Projects

- In order to use Forms in your project, you need to perform the following two steps:

  1. Add Form to the project
  2. Design the User Interface using Controls Class Objects (TexBox, Button, Labels etc)
  3. Display the Form

Step 1 - Creating & Adding Forms to Project

- When you create a new project a default Form, **Form1** is created by the IDE.
- When you add a form to a project what you are adding is actually a Form Class from the **System.Windows.Forms** namespace
- You can add additional Forms to your project as desired.

Adding Forms to your project:

- To add a Form to your project in the Menu Bar select the following command:

  Project|Add Windows Form..

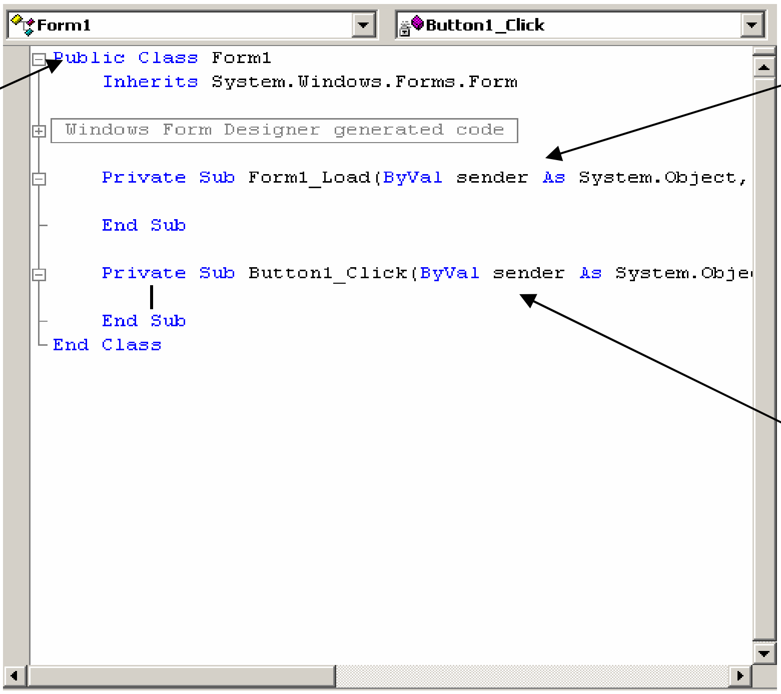
- Another method:

  In the Solution Explorer, Right-Click on the Project Name, in the context menu select **Add|Add Windows Form…**

Step 2- Create the User Interface Using Controls

- Controls from the Toolbox will be dropped onto Forms to create the GUI. (I will review controls in later lectures)
- When this is done, all controls or Object placed on a Form are now members or children objects of the Form Class.
- In other words the invisible part of the Control or the part where you will be writing code for the control will reside within the boundaries of the Form Class.
- This makes sense since the Controls are now residing inside the Form.

Class Declaration



```
Public Class Form1
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code

    Private Sub Form1_Load(ByVal sender As System.Object,

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Obje

    End Sub
End Class
```

Class Body Contains:
- Form Properties
- Form Methods
- Form Event-Procedures

Class Body Also Contains:
- Control Properties
- Control Methods
- Control Event-Procedures

❖ **Note that all Controls placed on the Form, are actually members of the class, therefore their properties, methods & event-procedures will be listed in the body of the class.**

Step 3- Displaying Forms

- In order to display a form from program code you need to perform two parts:
  a) ***Create an object*** of the Form Class that you just added to the project
  b) **Display** the Form Object.

Part A – Creating the Object

- So after you add the Form using step 1, then you need to create an Object of the Form class
- From previous lecture, we used the following syntax for creating objects:

Dim| Public | Private *ObjectName* **As** *ClassName* **= New** *ClassName***()**

- Another syntax used and my preference is the two step method. In this method, we first declare the class reference variable and in another step we create the object. Note that this is the same syntax we used to create string variables.
- Syntax:

Dim| Public | Private *FormObjectName* **As** *ClassName*

FormObjectName = **New** *ClassName***()**

Example:

- Assuming we have previously added a Form Class using step 1 above and named the Class frmLogin. Creating an Object of the Form class is as follows:

```
'Create Object of the Form Class
  Dim objLoginForm As frmLogin
  objLoginForm = New frmLogin()
```

Part B – Displaying the Form Object

- After you create the Form Object, now you need to display the form using one of the Form Class methods available to display the Form
- Before I explain how to display the object, lets look at two definitions:

  - ***Modal:*** When a window or Form displays, all other operations in the other windows are suspended until dialog closes
  - ***Modeless:*** When a Form displays, you can still switch the focus back and forth between the dialog and other windows.

- There are two class methods available in the Form Class to display a form: **ShowDialog()** and **Show()**
- The difference between the two are that ***ShowDialog*** displays a *Modal* Form while ***Show()*** displays the Form *Modeless*.
- Syntax:

FormObject.ShowDialog()

Example 1:

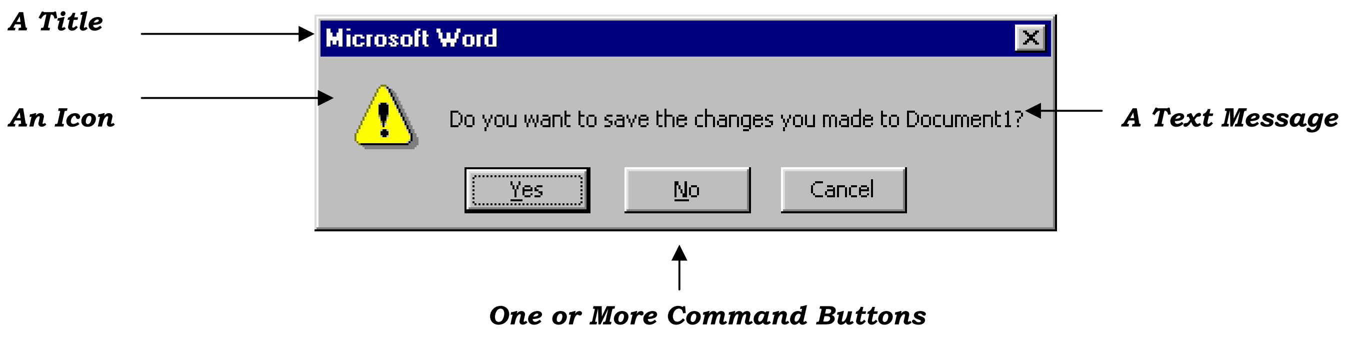
- Assuming we have previously added a Form Class using step 1 above and named the Class frmLogin. Creating an Object of the Form class is as follows:

```
   'Create Object of the Form Class
     Dim objLoginForm As frmLogin
     objLoginForm = New frmLogin()


  'Display Form
     objLoginForm.ShowDialog()
```

4.5.2 Message Box

- Message Boxes are useful for alerting the user to something and requiring some sort of response, such as Ok, Cancel etc.
- They are used when you want to give the user a simple message, such as an error message, warning or simply informing the user of something.
- Visual Basics.NET provides a **Message Box Class**. A **Message Box Object** does not contain Properties, but it does contain several methods. The one that we are interested in is the **Show()** Method.
- The **Show()** Method allows you to display a message box with the desired text, icon and buttons.
- The 4 components of a Message Box:



Do you want to save the changes you made to Document1?

- A message box comes in two flavors:
  - **As a Statement** – Display the message and user simply clicks the available buttons, but results from clicking is not used in code
  - **As a Function** – Display the message, but the buttons clicked is trapped and returned by the code and used to determine some condition

MessageBox Class Method As a Procedure

Message Box Object Show() Methods:

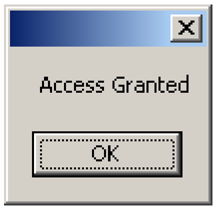
- The syntax to using the Show Method is as follows:

Syntax 1 – Message Statement: Message box shows a text message that the user must acknowledge by clicking on the OK button
MessageBox.Show(*TextMessage)*

Example:

- *'Displaying a simple message with an OK button with no title text*:

```
MessageBox.Show( "Access Granted")
```

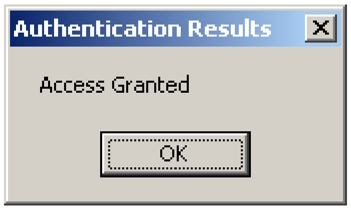


Syntax 2 – Statement with Title Text: Message box shows a text message and the title text describing the message box
MessageBox.Show(*TextMessage, TitlebarText)*

Example:

- *'Displaying a simple message with an OK button with title text*:

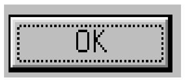
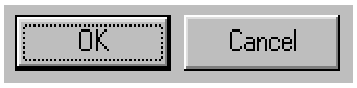
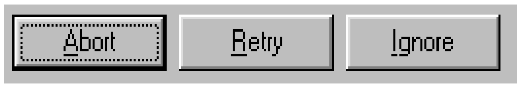
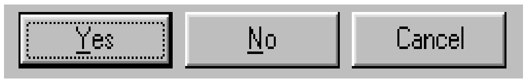
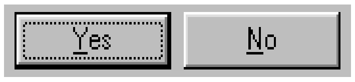
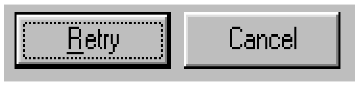
  MessageBox.Show("Access Granted", "Authentication Results")



Syntax 3 – Statement or Function with Title Text and other Buttons: Message box shows a text message, title text and buttons
MessageBox.Show(*TextMessage, TitlebarText, MessageBoxButtons)*

Where:
*MessageBoxButtons.**ButtonOptions***
Example: MessageBoxButtons.OKCancel
(see table below)

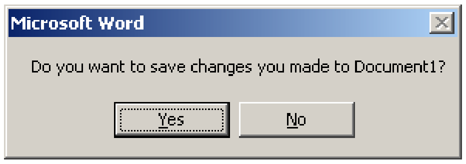
- The Message Box Button Options:

Constant	**Appearance**
OK	 OK
OKCancel	 OK Cancel
AbortRetryIgnore	 Abort Retry Ignore
YesNoCancel	 Yes No Cancel
YesNo	 Yes No
RetryCancel	 Retry Cancel

Example:

- *'Displaying a simple message with an OK button*:

MessageBox.Show("Do you want to save changes you made to Document1?", "Microsoft Word", MessageBoxButtons.YesNo



Do you want to save changes you made to Document1?

Syntax 4 – Statement with Title Text, Buttons and Icons: Message box shows a text message, title text, buttons and icons describing the type of message box
MessageBox.Show(*TextMessage, TitlebarText, MessageBoxButtons, MessageBoxIcon)*

Where:
*MessageBoxIcon.**Icon***
Example: MessageBoxIcon.Question
(see table below)

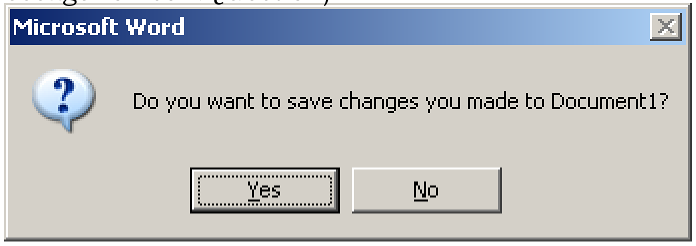
- The Message Box Icon Options:

Constant	**Appearance**
Error	❌
Question	❓
Exclamation	⚠️
Information	ℹ️

Example:

- *'Displaying a simple message with an OK button*:

MessageBox.Show("Do you want to save changes you made to Document1?", "Microsoft Word", MessageBoxButtons.YesNo, MessageBoxIcon.Question)



Do you want to save changes you made to Document1?

MessageBox Class Method As a Function

- You can program the Message box as a function which returns a constant value which represents the button clicked.
- You can use this value with an If/Else statement to perform some action depending on the button the user clicks.

Syntax 4 – Function that returns constant value of button selected: Message box shows a text message, title text, buttons and icons describing the type of message box

variable = MessageBox.Show(*TextMessage, TitlebarText, MessageBoxButtons, MessageBoxIcon)*

Where:

variable is an integer variable that will hold a specialized constant value returned by the Message Box

The constant values are shown in table below:

- Constant values returned:

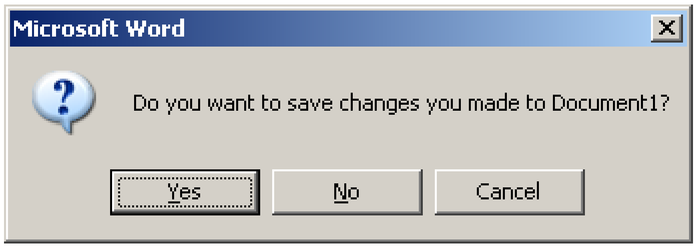
Visual Basic .NET Returned Constants
MsgBoxResult.OK
MsgBoxResult.Cancel
MsgBoxResult.Abort
MsgBoxResult.Retry
MsgBoxResult.Ignore
MsgBoxResult.Yes
MsgBoxResult.No

Example:

- *'Displaying a simple message with an OK button*:

Dim *intAnswer* **As String**

intAnswer = MessageBox.Show("Do you want to save changes you made to Document1?", "Microsoft Word", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question)



Do you want to save changes you made to Document1?

```
If Answer = MsgBoxResult.Yes Then
    'Do something here, for this example, such as saving the file

ElseIf Answer = MsgBoxResult.No Then
    'Do something here, for this example, such as not saving and exiting

ElseIf Answer = MsgBoxResult.Cancel Then
    'Do something here, for this example, such as continuing application

End If
```

4.5.3 Controls

- Controls are the graphical objects that together with Forms, make up the User Interface (UI)
- The Toolbox contains a variety of Controls. In addition you can purchase libraries which contain more custom controls.
- In this section we will look at some of the most common controls, their *properties*, *methods* and *events*

Naming Rules for Control Objects

- When you select a name for an Object, Visual Basics requires that the name begins with a letter.
- The name can contain letters, digits and underscores.
- Object names cannot contain space or punctuations marks.

Naming Conventions

- There is a naming convention that has been adopted for naming Objects in Visual Basics.NET
- The rule is that the Object name should be prefixed by a three letter characters.
- The table below lists the prefixes for the common Control Object naming convention:

Object	**Prefix**	**Example**
Form	**frm**	frmDataEntry
Button	**btn**	btnExit
TextBox	**txt**	txtFirstName
Label	**lbl**	lblTotal
Radio Button	**rad**	radMarriageStatus
Check Box	**Chk**	chkAllergic
Horizontal Scroll Bar	**Hsb**	hsbRate
Vertical Scroll Bar	**Vsb**	vsbTemperature
Picture Box	**Pic**	picLandscape
Combo Box	**Cbo**	cboBookList
List Box	**Lst**	LstIngredients

Control Objects In Visual Basic.NET

- In this section we will cover some of the most popular controls
- We will list the control, what it looks like, some common *properties*, common *methods* and *events*.

  - **Note that as a prerequisite to this course, you should have taken a CS101 & CS508 where the basic controls were covered in details.**
  - **In the following sections I will lists the controls and their usage for your reading only. I will not cover this is class, but I am listing it so that you may review this material if necessary.**

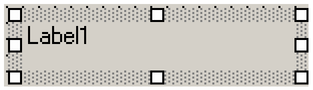
Label

Description

- **Label Control** is used to display static text on a Form.
- This control is used when you want to display titles, labels to other controls ect.

Graphical Representation

- **Label Control** visual representation:



Label Properties & Coding

- The **Label** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Label Properties**:

Common Label Properties:

BackColor	Gets or sets the background color for the control.
BackgroundImage	image to display in the background of the control.
BorderStyle	Gets or sets the border style for the control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
Image	Gets or sets the image that is displayed on a Label.
Name	Gets or sets the name of the control.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can tab to the Label.
Text	Gets or sets the text associated with this control.
TextAlign	Gets or sets the alignment of text in the label.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.

Property Coding Syntax:

```
 ‘Assigning value to property
LabelControl.Property = variable
```

```
‘Getting value of property
variable = LabelControl.Property
```

```
‘Assigning value of one property to another
LabelControl1.Property = LabelControl2.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *‘Assigning value to property*:
  lblTitle.*Text* = *“Terminator 3”*

- Example 2 - *Getting value of property*:
  value = **lblTitle.*Hight***

- Example 3 - *Assigning value of one property to another*:
  lblTitle.*Text* = **lblMovieName.*Text***

Label Method & Coding

- The following table is a list of some of the most common Label Methods:

Common Label Methods:

BringToFront	Brings the control to the front of the z-order.
Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

```
 ‘Calling a Method
LabelControl.Method()
```

```
‘Additional Syntax in future lectures
```

Example:

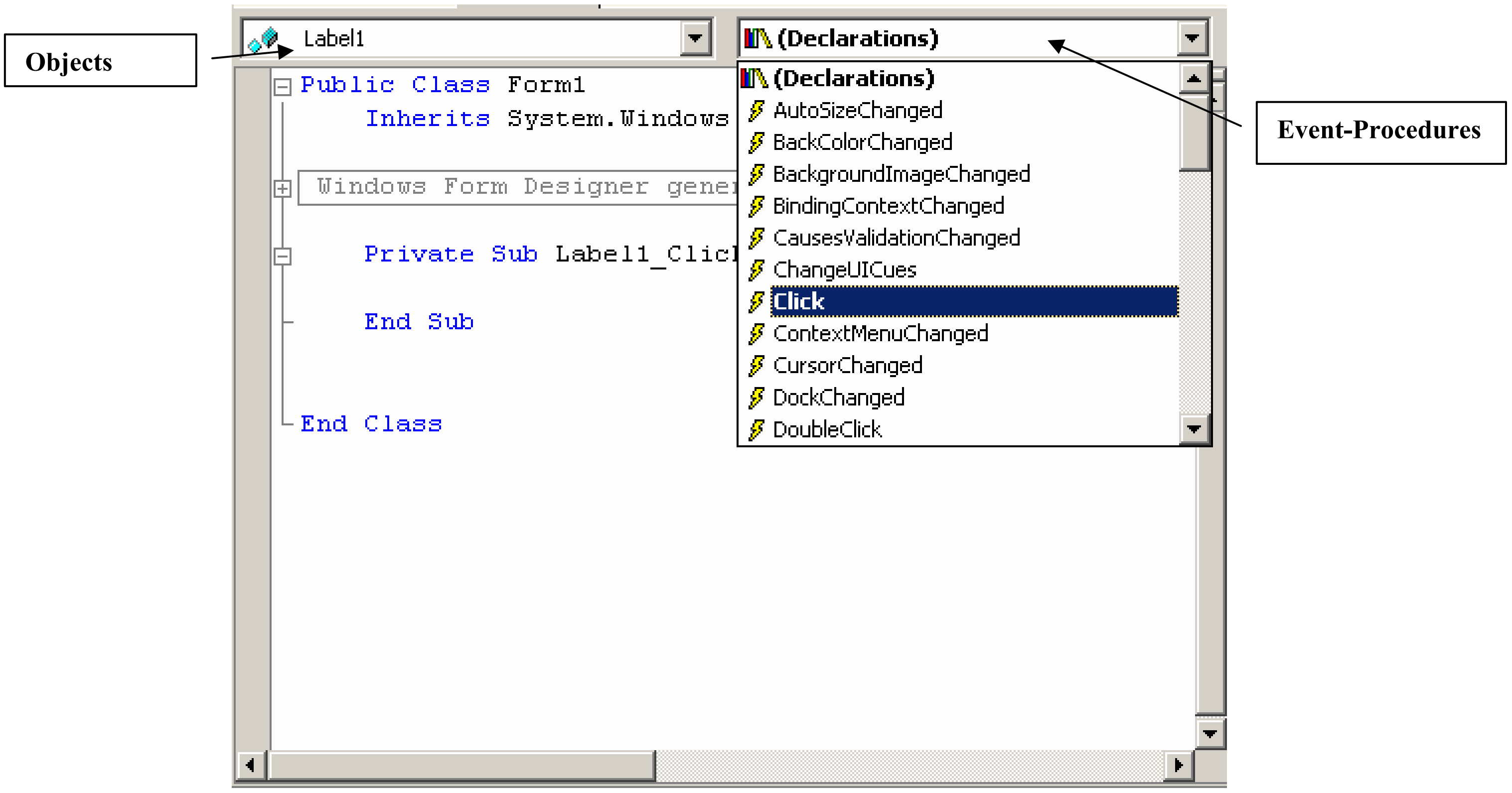
- **Executing or Calling a Method:**

- Example 1:
  lblTitle.*Show()*

- Example 2:
  lblTitle.*Hide()*

Label Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **Label Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windows

 Windows Form Designer gene

    Private Sub Label1_Clic

    End Sub

End Class
```

- The following table is a list of some of the most common Label Events & Event-Procedures:

Common Label Events:

Click	Occurs when the control is clicked.
DoubleClick	Occurs when the control is double-clicked.
Enter	Occurs when the control is entered.
GotFocus	Occurs when the control receives focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TextChanged	Occurs when the Text property value changes.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

- **Coding the Click() Event:**

- Example: *Suppose you want the label to become disabled if the user clicks on it*
-

```
Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Label1.Click

        Label1.Enabled = False


End Sub
```

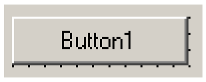
Push Button

Description

- **Push Button Control** is used to execute commands
- The Button can be used to perform any operation we want when it is pushed or other events are taken upon it.
- A **Button** may be clicked by using the mouse, ENTER key, or SPACE B

Graphical Representation

- **Push Button Control** visual representation:



Push Button Properties & Coding

- The **Push Button** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Push Button Properties**:

Common Push Button Properties:

BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
CanFocus	Gets a value indicating whether the control can receive focus.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
Image	Gets or sets the image that is displayed on a button control.
Name	Gets or sets the name of the control.
Size	Gets or sets the height and width of the control.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Text	Gets or sets the text associated with this control.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.

Property Coding Syntax:

'*Assigning value to property*
Push ButtonControl.*Property* = *variable*

'*Getting value of property*
variable = **Push ButtonControl.***Property*

'*Assigning value of one property to another*
Push ButtonControl1.*Property* = **Push ButtonControl2.***Property*

Example:

- **Setting & Retrieving Properties:**

- Example 1 – '*Assigning value to property*:
  btnExit.*Text* = *"Exit"*
  btnExit.*Visible* = *True*

- Example 2 - *Getting value of property*:
  value = **btnOK.*Hight***

- Example 3 - *Assigning value of one property to another*:
  btnExit.*Text* = **btnFinished.*Text***

Push Button Method & Coding

- The following table is a list of some of the most common Push Button Methods:

Common Push Button Methods:

Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

'*Calling a Method*
ButtonControl.*Method()*

'*Additional Syntax in future lectures*

Example:

- **Executing or Calling a Method:**

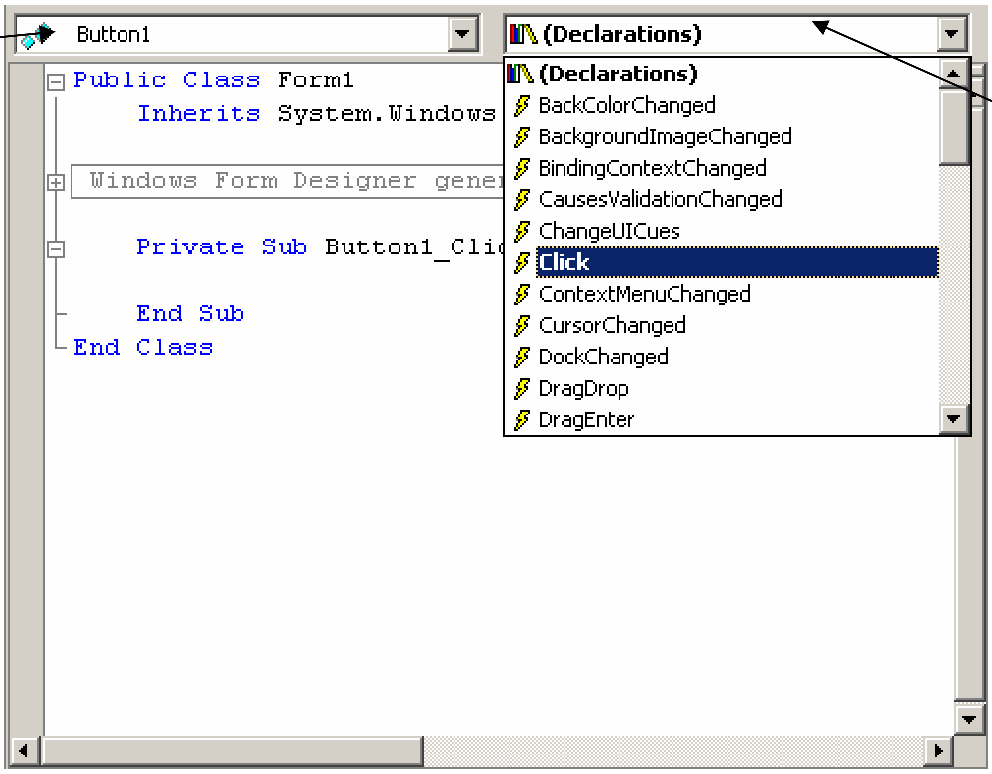
- Example 1:
  btnExit.*Show()*

- Example 2:
  btnOK.*Hide()*

Push Button Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **Push Button Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:

Objects



```
Public Class Form1
    Inherits System.Windows

 Windows Form Designer gene

    Private Sub Button1_Cli

    End Sub
End Class
```

Event-Procedures

- The following table is a list of some of the most common Push Button Events & Event-Procedures:

Common Push Button Events:

Click	Occurs when the control is clicked.
GotFocus	Occurs when the control receives focus.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TabIndexChanged	Occurs when the TabIndex property value changes.
TabStopChanged	Occurs when the TabStop property value changes.
TextChanged	Occurs when the Text property value changes.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

❑ **Coding the Click() Event:**

▪ Example: *Suppose you want the Button to End the program when clicked*

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click

       'Command that ends the program
       End


End Sub
```

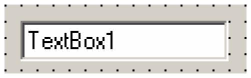
Text Boxes

Description

- **Text Box Control** is used to accept text entries or input from the user.

Graphical Representation

- **Text Box Control** visual representation:



Text Box Properties & Coding

- **Text Boxes** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Text Boxes Properties**:

Common Text Box Properties:

BackColor	Overridden. Gets or sets the background color of the control.
BackgroundImage	Gets or sets the background image displayed in the control.
BorderStyle	Gets or sets the border type of the text box control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Overridden. Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
MaxLength	Gets or sets the maximum number of characters the user can type into the text box control.
Name	Gets or sets the name of the control.
Parent	Gets or sets the parent container of the control.
SelectedText	Gets or sets a value indicating the currently selected text in the control.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Text	Gets or sets the current text in the text box.
TextAlign	Gets or sets how text is aligned in a **TextBox** control.
TextLength	Gets the length of text in the control.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.
WordWrap	Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when

Property Coding Syntax:

```
 'Assigning value to property
TextBoxControl.Property = variable
```

```
'Getting value of property
variable = TextBoxControl.Property
```

```
'Assigning value of one property to another
TextBoxControl1.Property = TextBoxControl2.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *'Assigning value to property*:
  txtName.*Text* = *"Joe"*

- Example 2 - *Getting value of property*:
  Name_value = **txtName.***Text*

- Example 3 - *Assigning value of one property to another*:
  txtLastName.*Text* = **txtEmployeeName.***Text*

Text Boxes Methods & Coding

- The following table is a list of some of the most common Text Box Methods:

Common Text Box Methods:

AppendText	Appends text to the current text of text box.
Clear	Clears all text from the text box control.
Copy	Copies the current selection in the text box to the Clipboard.
Cut	Moves the current selection in the text box to the Clipboard.
Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Paste	Replaces the current selection in the text box with the contents of the Clipboard.
ResetText	Resets the Text property to its default value.
Show	Displays the control to the user.
Undo	Undoes the last edit operation in the text box.

Method Coding Syntax:

```
 'Calling a Method
Text BoxControl.Method()
future lectures
```

Example:

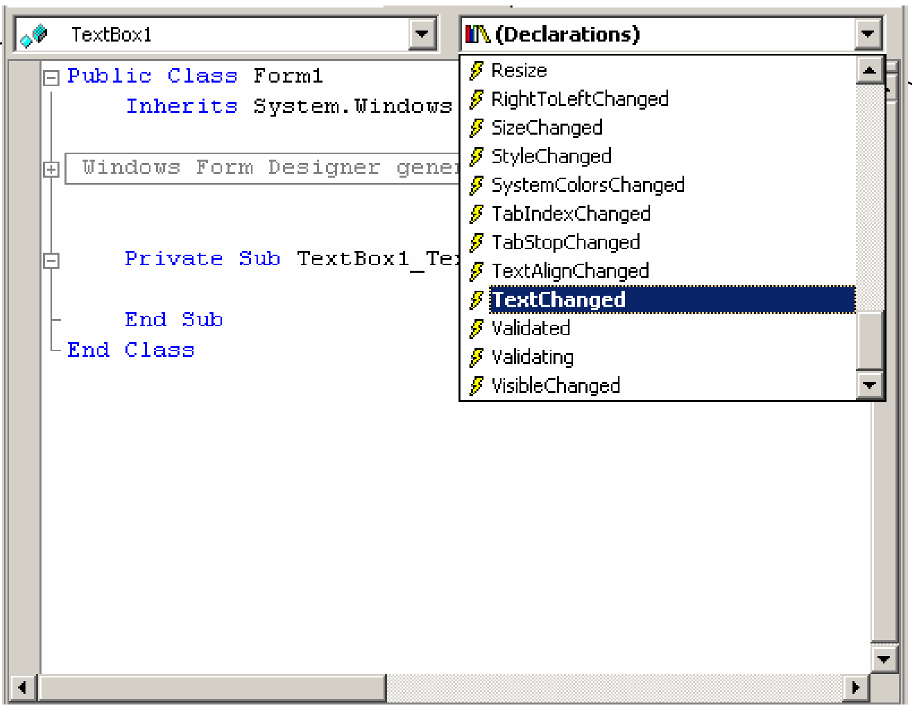
- **Executing or Calling a Method:**

- Example 1:
  txtExit.*Show()*

Text Box Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **Text Box Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:

Objects



```
Public Class Form1
    Inherits System.Windows
Windows Form Designer gene

    Private Sub TextBox1_Te

    End Sub
End Class
```

Event-Procedures

- The following table is a list of some of the most common Text Box Events & Event-Procedures:

Common Text Box Events:

Click	Occurs when the text box is clicked.
DoubleClick	Occurs when the control is double-clicked.
Enter	Occurs when the control is entered.
FontChanged	Occurs when the Font property value changes.
ForeColorChanged	Occurs when the ForeColor property value changes.
GotFocus	Occurs when the control receives focus.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TabIndexChanged	Occurs when the TabIndex property value changes.
TabStopChanged (inherited from **Control**)	Occurs when the TabStop property value changes.
TextAlignChanged	Occurs when the value of the TextAlign property has changed.
TextChanged (inherited from **Control**)	Occurs when the Text property value changes.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

- ❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

- ❑ **Coding the Text_Changed() Event:**

- ▪ Example: *Suppose you want to extract the text entered into the text box as the user enters text into the text box*

```
Private Sub txtName_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
txtName_TextChanged

        Variable = txtName.text


End Sub
```

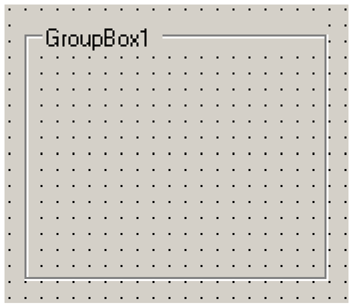
Group Boxes

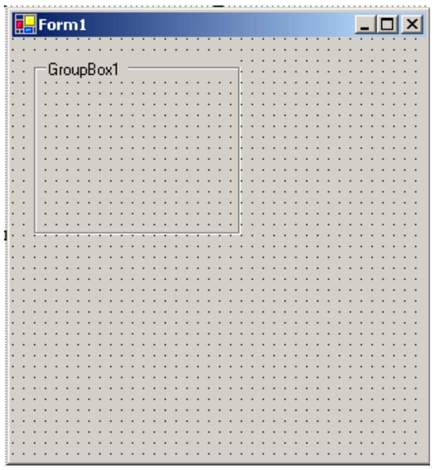
Description

- **Group Box Control** is a container to other control.
- Usually groups of Check Boxes and Radio Buttons are place in **Group Boxes**.
- Using **Group Boxes** to group controls on a Form, makes the Forms easier to understand and manage

Graphical Representation

- **Group Box Control** visual representation:





Group Box Properties & Coding

- **Group Boxes** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Group Boxes Properties**:

Common Group Box Properties:

BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
Name	Gets or sets the name of the control.
RightToLeft	Gets or sets a value indicating whether control's elements are aligned to support locales using right-to-left fonts.
Site	Overridden. Gets or sets the site of the control.
TabIndex	Gets or sets the tab order of the control within its container.
Text	Gets or sets the current text in the control.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control

Property Coding Syntax:

```
 'Assigning value to property
GroupBoxControl.Property = variable
```

```
'Getting value of property
variable = GroupBoxControl.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *'Assigning value to property*:
  grpGender.*Text* = *"Gender"*

Group Boxes Methods & Coding

- The following table is a list of some of the most common Group Box Methods:

Common Group Box Methods:

Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

```
 'Calling a Method
GroupBoxControl.Method()
future lectures
```

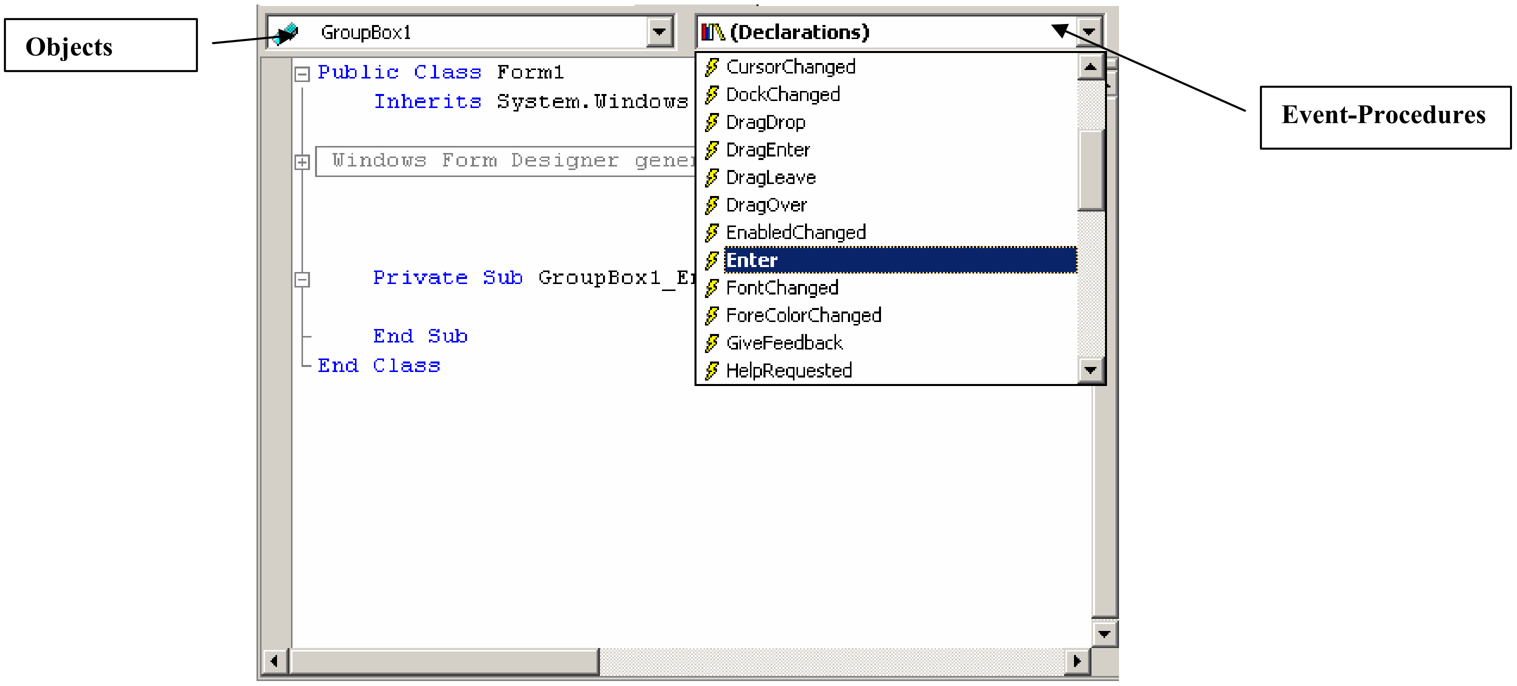
Example:

- **Executing or Calling a Method:**

- Example 1:
  grpDayOfWeek.*Show()*

Group Box Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **Group Box Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windows

 Windows Form Designer gener

    Private Sub GroupBox1_E

    End Sub
End Class
```

- The following table is a list of some of the most common Group Box Events & Event-Procedures:

Common Group Box Events:

Enter	Occurs when the control is entered.
GotFocus	Occurs when the control receives focus.
Leave	Occurs when the input focus leaves the control.
LostFocus	Occurs when the control loses focus.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TabIndexChanged	Occurs when the TabIndex property value changes.
TabStopChanged	Occurs when the TabStop property value changes.
TextChanged	Occurs when the Text property value changes.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

❑ **Coding the Enter() Event:**

▪ Example: *Suppose you a label control to indicate when a user is a citizen*

```
Private Sub grpGender_Enter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
grpGender_Enter

        lblInformation.Text = "User is using items in group"


End Sub
```

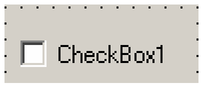
Check Boxes

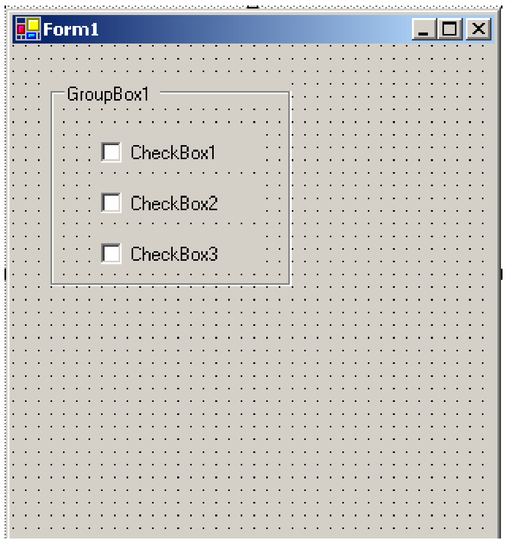
Description

- **Check Box Control** is used to select or de-select an option with a check mark
- In any group of Check Boxes any number of Check Boxes can be selected. You can check one or more.
- When a check box is checked, a property known as ***Checked Property*** is TRUE, and FALSE when unchecked
- **Check Boxes Controls** usually works together with the **Group Boxes Control**.

Graphical Representation

- **Check Box Control** visual representation:





Check Box Properties & Coding

- The **Check Box** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Check Box Properties**:

Common Check Box Properties:

BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
CanSelect	Gets a value indicating whether the control can be selected.
CheckAlign	Gets or sets the horizontal and vertical alignment of a check box on a check box control.
Checked	Gets or set a value indicating whether the check box is in the checked state.
CheckState	Gets or sets the state of the check box.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Focused	Gets a value indicating whether the control has input focus.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
Image	Gets or sets the image that is displayed on a button control.
Name	Gets or sets the name of the control.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Text	Gets or sets the text associated with this control.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.

Property Coding Syntax:

```
'Assigning value to property
CheckBoxControl.Property = variable
```

```
'Getting value of property
variable = CheckBoxControl.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *'Assigning value to property*:
  chkMale.*Checked* = *True*
  chkFemale.*Text* = *"Female"*

- Example 2 - *Getting value of property*:
  value = **chkMale.***Checked*

Check Box Method & Coding

- The following table is a list of some of the most common Check Box Methods:

Common Check Box Methods:

Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
ResetText	Resets the Text property to its default value.
SendToBack	Sends the control to the back of the z-order.
Show	Displays the control to the user.

Method Coding Syntax:

```
'Calling a Method
CheckBoxControl.Method()
```

Example:

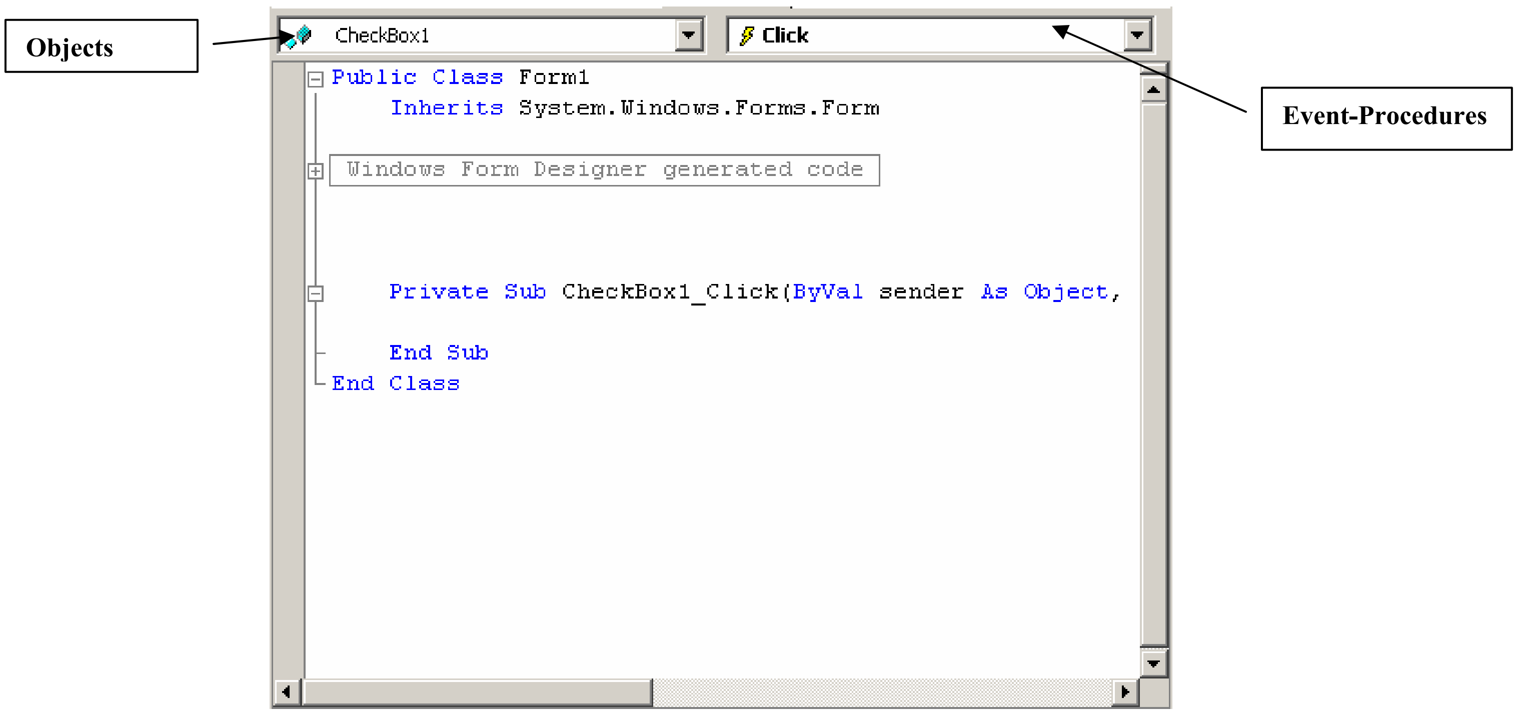
- **Executing or Calling a Method:**

- Example 1:
  chkAccept.*Show()*

- Example 2:
  chkAccept.*Hide()*

Check Box Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **Check Box Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Windows Form Designer generated code

    Private Sub CheckBox1_Click(ByVal sender As Object,

    End Sub
End Class
```

- The following table is a list of some of the most common Check Box Events & Event-Procedures:

Common Check Box Events:

CheckedChanged	Occurs when the value of the Checked property changes.
CheckStateChanged	Occurs when the value of the CheckState property changes.
Click	Occurs when the control is clicked.
DoubleClick	Occurs when the control is double-clicked.
EnabledChanged	Occurs when the Enabled property value has changed.
Enter	Occurs when the control is entered.
GotFocus	Occurs when the control receives focus.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TabIndexChanged	Occurs when the TabIndex property value changes.
TabStopChanged	Occurs when the TabStop property value changes.
TextChanged	Occurs when the Text property value changes.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

❑ **Coding the CheckedChanged() Event:**

▪ Example: *Suppose you a label control to indicate when a user is a citizen*

```
Private Sub chkUSCitizen_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
chkUSCitizen_ CheckedChanged

        lblTitle.Text = "User is a Citizen"


End Sub
```

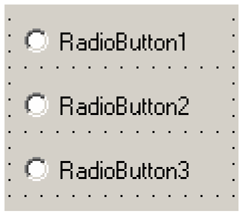
Radio Buttons (Option Buttons)

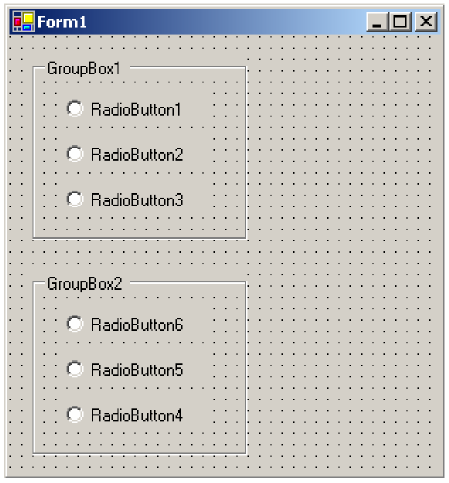
Description

- **Radio Button Control** is used to select one option or button from a group of many
- Radio Button gives the user a single choice from several options.
- The key point here is only one option out of many, you can only select ONE!
- When a **Radio Button** is selected, a property known as ***Checked Property*** is TRUE, and FALSE when unselected
- **Radio Buttons** work as a group. That is if you place several Radio Button in a Form (No matter location), they will work as a group automatically, therefore selecting one button will turn of the other button that was previously selected.
- In order to allow you to program several groups of **Radio Buttons** differently, the Radio Button works together with the **Group Boxes Control**.
- A group of Radio Buttons inside a Group Box function together as one unit.
- The best method for using **Radio Buttons** is to first create the **Group Box**, then create each **Radio Button** inside the **Group Box**

Graphical Representation

- **Radio Button Control** visual representation:





Radio Button Properties & Coding

- The **Radio Button** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Radio Button Properties**:

Common Radio Button Properties:

Appearance	Gets or set the value that determines the appearance of the radio button control.
BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
Checked	Gets or sets a value indicating whether the control is checked.
ContainsFocus	Gets a value indicating whether the control, or one of its child controls, currently has the input focus.
Disposing	Gets a value indicating whether the control is in the process of being disposed of.
Dock	Gets or sets which edge of the parent container a control is docked to.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
FlatStyle	Gets or sets the flat style appearance of the button control.
Font	Gets or sets the font of the text displayed by the control.
ForeColor	Gets or sets the foreground color of the control.
Height	Gets or sets the height of the control.
Image	Gets or sets the image that is displayed on a button control.
Name	Gets or sets the name of the control.
TabIndex	Gets or sets the tab order of the control within its container.
Text	Gets or sets the text associated with this control.
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.

Property Coding Syntax:

```
'Assigning value to property
RadioButtonControl.Property = variable
```

```
'Getting value of property
variable = RadioButtonControl.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *Assigning value to property*:
  ```
  radON.Checked  = True
  radOFF.Checked  = FALSE
  ```

- Example 2 - *Assigning value to property*:
  ```
  radOFF.Text  = "Off"
  ```

- Example 3 - *Getting value of property*:
  ```
  value = radON.Checked
  ```

- Example 4 – *Verifying which button is selected and placing result in text box*
  ```
  If  radON.Checked Then
              txtStatus.Txt = "The Television is ON"
  ElseIf radOFF.Checked  Then
             txtStatus.Txt = "The Television is OFF"
  End If
  ```

Radio Button Method & Coding

- The following table is a list of some of the most common Radio Button Methods:

Common Radio Button Methods:

Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

```
'Calling a Method
RadioButtonControl.Method()
```

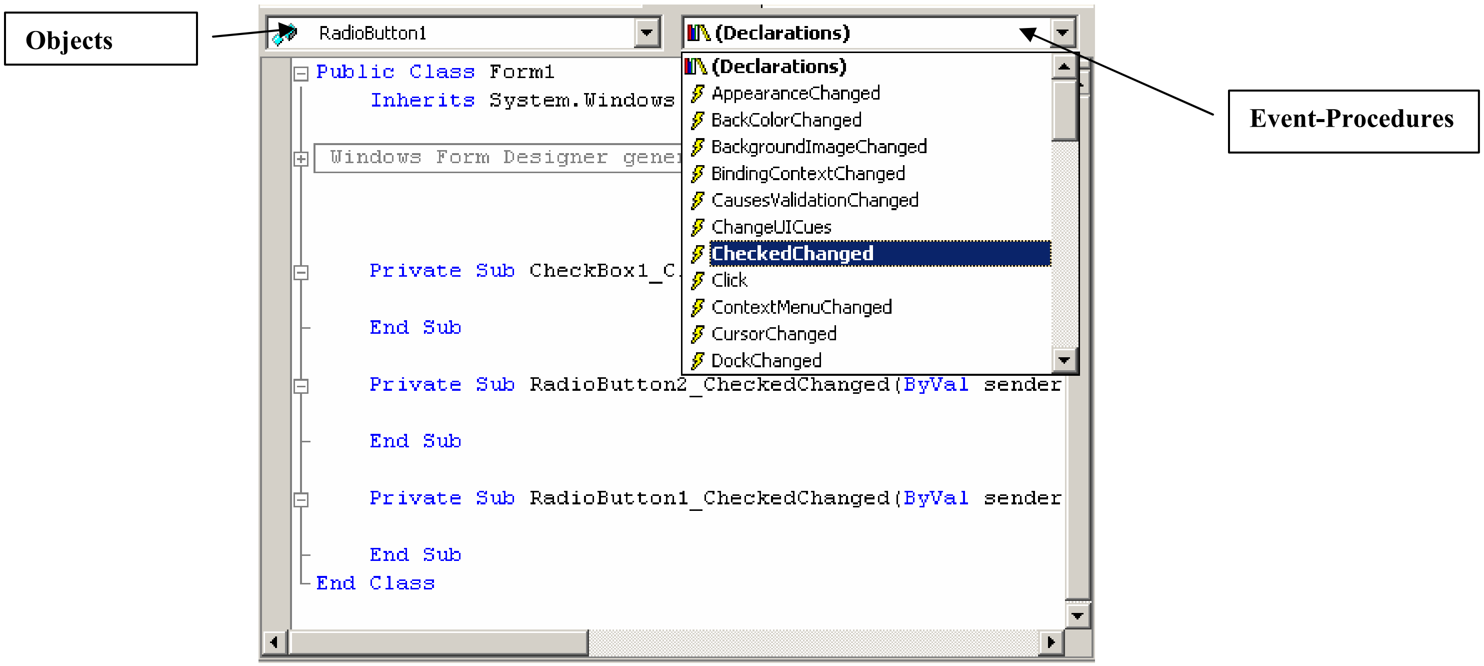
Example:

- **Executing or Calling a Method:**

- Example 1:
  ```
  radMale.Show()
  ```
- Example 2:
  ```
  radMale.Hide()
  ```

Radio Buttons Event, Event-Procedures & Coding

- ❑ Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- ❑ You can view the list of Event in the Code Editor Window by selecting the **Radio Button Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windows

 Windows Form Designer gene

    Private Sub CheckBox1_C

    End Sub

    Private Sub RadioButton2_CheckedChanged(ByVal sender

    End Sub

    Private Sub RadioButton1_CheckedChanged(ByVal sender

    End Sub
End Class
```

- ❑ The following table is a list of some of the most common Radio Button Events & Event-Procedures:

Common Radio Button Events:

CheckedChanged	Occurs when the value of the Checked property changes.
Click	Occurs when the control is clicked.
DoubleClick	Occurs when the control is double-clicked.
GotFocus	Occurs when the control receives focus.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.
TabIndexChanged	Occurs when the TabIndex property value changes.
TabStopChanged	Occurs when the TabStop property value changes.

Event Coding Syntax:

```
  'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

- **Coding the CheckedChanged() Event:**

- Example: *Suppose you use a label control to indicate when the status of an option has changed from ON to OFF or OFF to ON*

```
Private Sub radON_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
radON_CheckedChanged

        lblIndicator.Text = "The Television status has changed"


End Sub
```

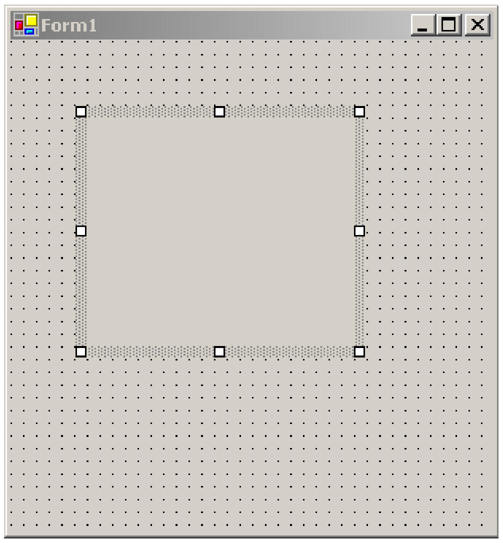
Picture Boxes

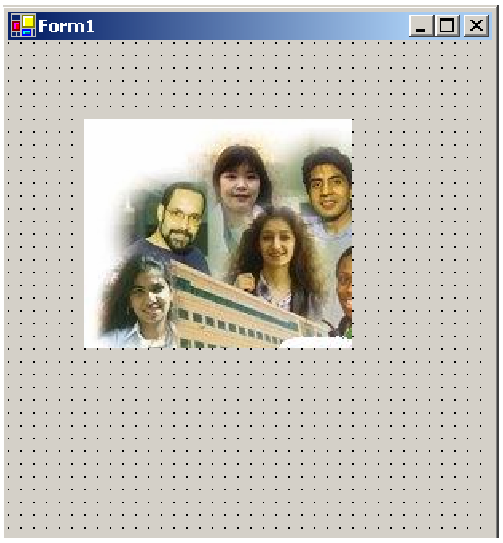
Description

- **Picture Box Control** is used to hold images.
- This is done by setting the **Image Property** of the **Picture Box** to a graphic file with extension of .**bmp**, **gif**, **jpg**, **png**, **ico**, **emf**, or **wmf**.
- Note that it's a good idea to place the graphic file in the same folder of the project prior to setting it to the Picture Box.

Graphical Representation

- **Picture Box Control** visual representation without graphics & with graphic file assigned :





Picture Box Properties & Coding

- **Picture Boxes** Properties can be viewed and modified using the ***Property Window***
- The following table is a list of some of the most common **Picture Boxes Properties**:

Common Picture Box Properties:

BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
BorderStyle	Indicates the border style for the control.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Height	Gets or sets the height of the control.
Image	Gets or sets the image that the **PictureBox** displays.
Name	Gets or sets the name of the control.
SizeMode	Indicates how the image is displayed, either size to fit the picture file or stretch to fit the picture box
Visible	Gets or sets a value indicating whether the control is displayed.
Width	Gets or sets the width of the control.

Property Coding Syntax:

```
 'Assigning value to property
PictureBoxControl.Property = variable
```

```
'Getting value of property
variable = GroupBoxControl.Property
```

Example:

- **Setting & Retrieving Properties:**

- Example 1 – *'Assigning value to image property at design time*:
  picStudentLogo.*Image* = *use browse button and browse to desired file*

- Example 2 – *'Assigning value to image property at Run time via code*:
  picStudentLogo.*Image* = **Image.FromFile(***filePath***)**

Picture Boxes Methods & Coding

- The following table is a list of some of the most common Picture Box Methods:

Common Picture Box Methods:

Public Methods

Focus	Sets input focus to the control.
Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

```
 'Calling a Method
PictureBoxControl.Method()
future lectures
```

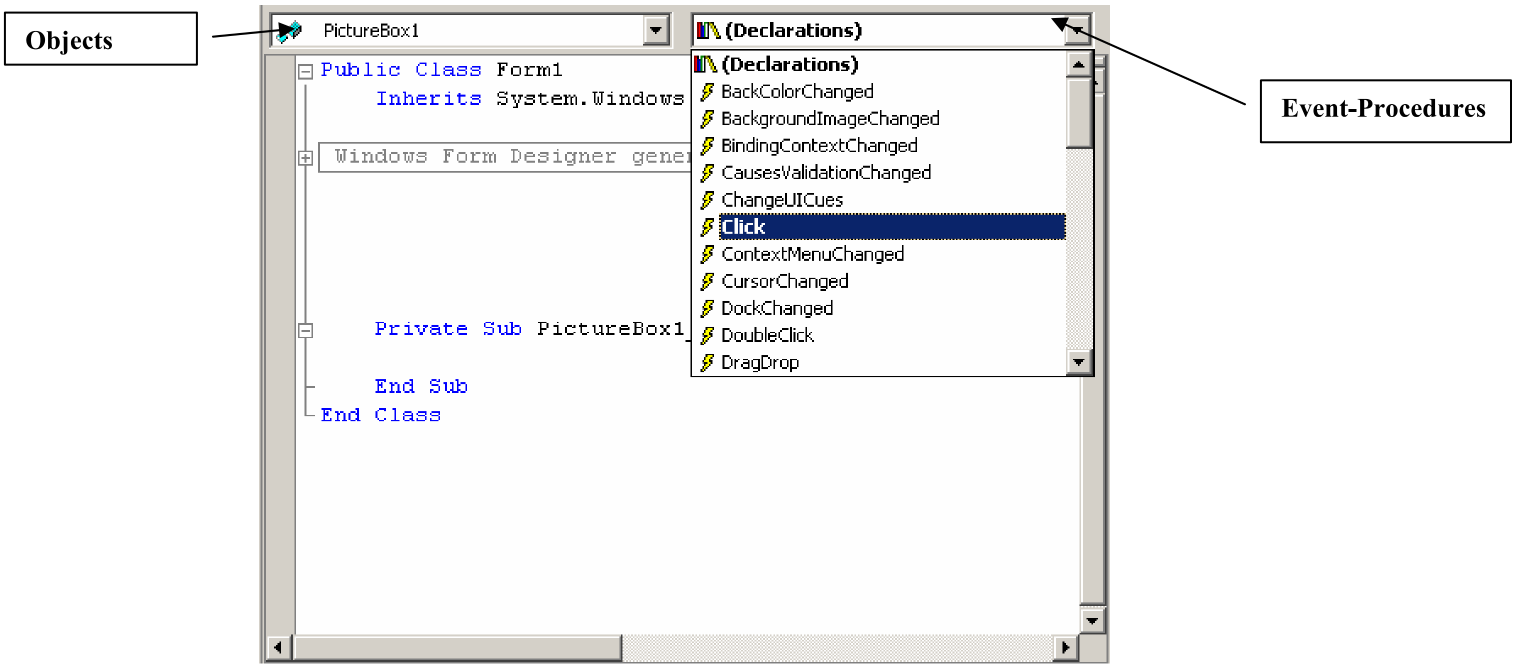
Example:

- **Executing or Calling a Method:**

- Example 1:
  picSchoolLogo.*Show()*

Picture Box Event, Event-Procedures & Coding

- ❑ Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- ❑ You can view the list of Event in the Code Editor Window by selecting the **Picture Box Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windows

 Windows Form Designer gene

    Private Sub PictureBox1_

    End Sub
End Class
```

- ❑ The following table is a list of some of the most common Picture Box Events & Event-Procedures:

Common Picture Box Events:

Public Events

Click	Occurs when the control is clicked.
DoubleClick	Occurs when the control is double-clicked.
GotFocus	Occurs when the control receives focus.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseWheel	Occurs when the mouse wheel moves while the control has focus.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

❑ **Coding the Click() Event:**

▪ Example: *Suppose you a label control to indicate when the picture has been clicked*

```
Private Sub picSchoolLogo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
picSchoolLogo_Click

        lblInformation.Text = "User has selected the picture"


End Sub
```

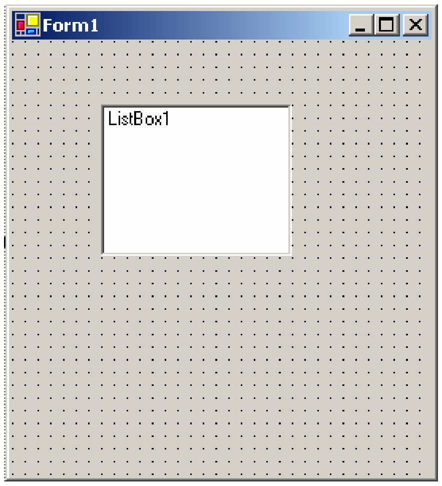
List Box

Description

- ❑ **List Box Control** store a list of items from which the user can make a selection.
- ❑ The items stored on the list are strings of characters.
- ❑ The items stored on the list can be set a design time or by code during run time.
- ❑ If the **List Box** is too small to display all the items on the list, VB automatically adds scroll bars.

Graphical Representation

- ❑ **List Box Control** visual representation:



List Box Properties & Coding

- ❑ **List Boxes** Properties can be viewed and modified using the ***Property Window***
- ❑ The following table is a list of some of the most common **List Boxes Properties**:

Common List Box Properties:

Public Properties

BackColor	Set the back color
BackgroundImage	Sets the background image
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
Font	Gets or sets the font of the text displayed by the control.
Items	**Collection Object** - Gets the items of the **ListBox**.
Left	Gets or sets the x-coordinate of a control's left edge in pixels.
Name	Gets or sets the name of the control.
SelectedIndex	Overridden. Gets or sets the zero-based index of the currently selected item in a **ListBox**.
Sorted	Gets or sets a value indicating whether the items in the **ListBox** are sorted alphabetically.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Visible	Gets or sets a value indicating whether the control is displayed.

Property Coding Syntax:

```
 'Assigning value to property
ListBoxControl.Property = variable
```

```
'Getting value of property
variable = ListBoxControl.Property
```

```
'Assigning value of one property to another
ListBoxControl1.Property = ListBoxControl2.Property
```

Important Properties of a ListBox

- The following Properties are important when programming a Listbox:

Name Property:

- **Name**: As usual this property is important since it's how the object is referenced in the code
- If no items is selected by the user, **SelectedIndex = -1**

Items Property:

- **Items**: This property is actually an Object residing as a member of the ListBox Object. Through this property you will add, remove, and clear items from the ListBox (More on this below)

SelectedIndex Property:

- **SelectedIndex Property**: Stores the index of the item selected or highlighted by the user.
- If no items is selected by the user, **SelectedIndex = -1**

Sorted Property:

- **Sorted Property**: Sorts the items on the list alphabetically

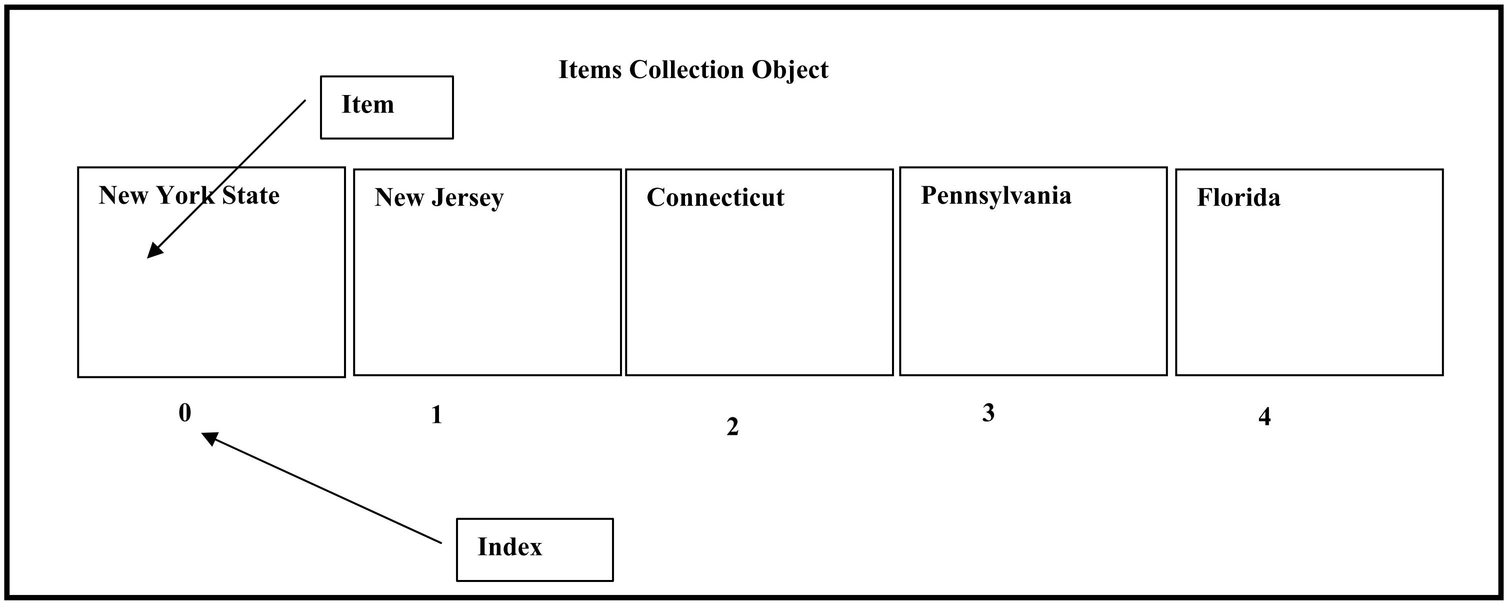
Working with the Items on the List The Items Collection Object

- The elements stored on the list are know as ***Items***

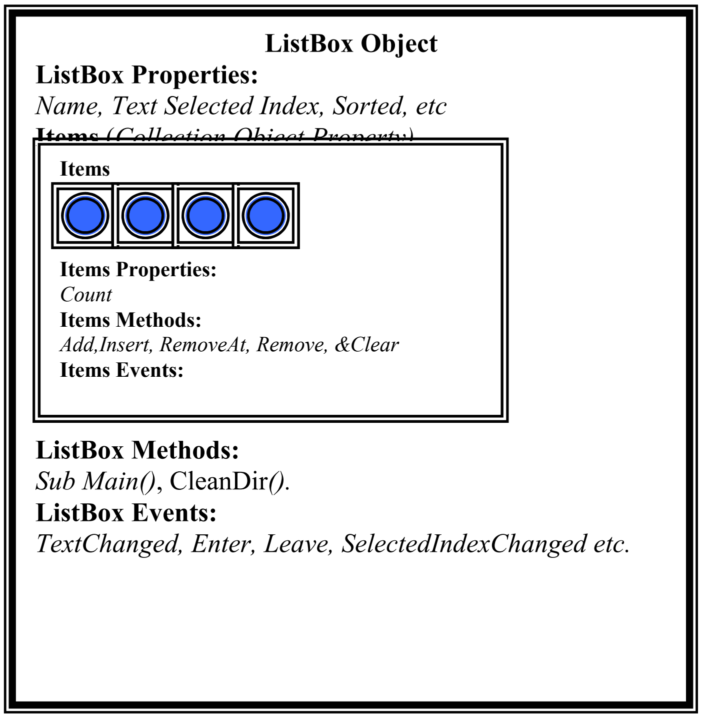
The Items Collection

- The lists of items in the ListBox are stored in a **Collection Object**.
- A **Collection Object** is a special Object built into Visual Basic that stores & manages groups of items
    - This concept is a little advanced for this course, but think of the **Collection Object** as another object that lives inside the **ListBox Object** (An Object within an Object).
    - This may seem confusing but it's quite easy, think about it, you wallet is an object, yet is stores other objects inside such as a Credit Card Object, Money Object, etc.
    - Since the Collection Object is an Object, it has Properties, Methods & Events.
- Right now we are not going to go into this with too much detail, but a brief introduction is required
- Think of the Collection Object as an closet with several shelves or cells to store items in it, and the things that you can do is store items into a shelve, remove items, retrieve item etc.

- In a **Collection Object**, the elements or **Items** stored in each cell, are identified by the ***Index*** number of the cell, starting with ***zero***. The first cell has an index of 0, the second 1, the third 2, etc.
- **Important:** Items are identified by their Index, not the content or what's inside! So manipulate and refer to the items by their index number.



- The following is a graphical representation of the interaction between the **ListBox Object** and it's Object Property member ***Items Object*** :


ListBox Object
ListBox Properties:
Name, Text Selected Index, Sorted, etc
Items (Collection Object Property)
Items
Items Properties:
Count
Items Methods:
Add,Insert, RemoveAt, Remove, &Clear
Items Events:
ListBox Methods:
Sub Main(), CleanDir().
ListBox Events:
TextChanged, Enter, Leave, SelectedIndexChanged etc.

Properties & Methods of the Items Object

- The Items Object has the following Properties:
- Properties:
  - **Item.Count Property**: The count property indicates the number of items on the list

- Methods:
  - **Item.Add() Method**: Add items to the end of the list
  - **Item.Insert() Method**: Add Items to a specified index location on the list
  - **Item.RemoveAt() Method**: Removes an Item at a specified index location
  - **Item.Remove() Method**: Removes an Item that matches a character string
  - **Item.Clear() Method**: Removes all Items from the list

Populating or Filling a List

- There are two ways to do this.
  - ***Design Time:*** If your list is never going to change, you can assign values during design time via the Property Window.
  - ***Run Time:*** If your list will change during program execution then you need to use the **Items Object Methods** to do so:

    - Use the dot operator (.) in order to use the Items Object Methods from within the ListBox Object as follows: **ListBoxObject.ItemsObject.Add()** or **ListBoxObject.ItemsObject.Insert()**
    - Syntax:

```
'Object calling Object Member Method
Object.Object.Method()
```

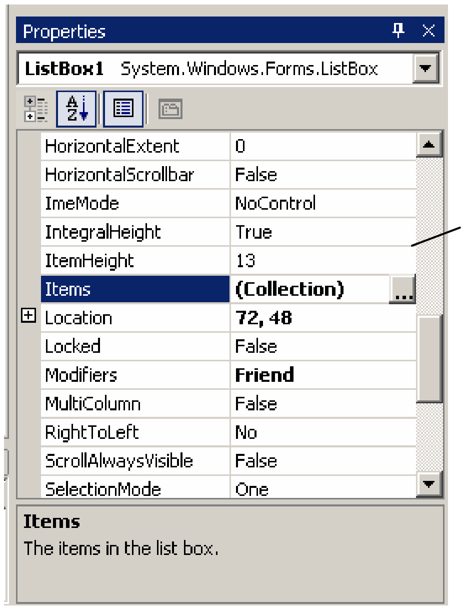
```
'ListBox Object calling it's Items Object Add Method
ListBox.Items.Add()
```

```
'ListBox Object calling it's Items Object Insert Method
ListBox.Items.Insert()
```

Populating the List at Design Time

- Populating the list at design time is done as follows:

Step 1: After you drop the ListBox Control onto the form.
Step 2: Select the ListBox Control and in the Property Window, click on the Items Collection *Browse button* to invoke the "String Collection Editor:



Properties

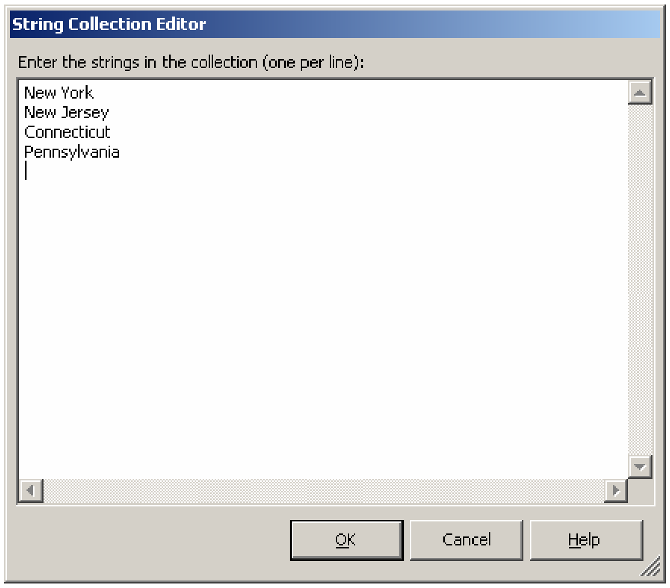
ListBox1 System.Windows.Forms.ListBox

HorizontalExtent	0
HorizontalScrollbar	False
ImeMode	NoControl
IntegralHeight	True
ItemHeight	13
Items	**(Collection)**
Location	**72, 48**
Locked	False
Modifiers	**Friend**
MultiColumn	False
RightToLeft	No
ScrollAlwaysVisible	False
SelectionMode	One

Items
The items in the list box.

Click here to open the String Collection Editor

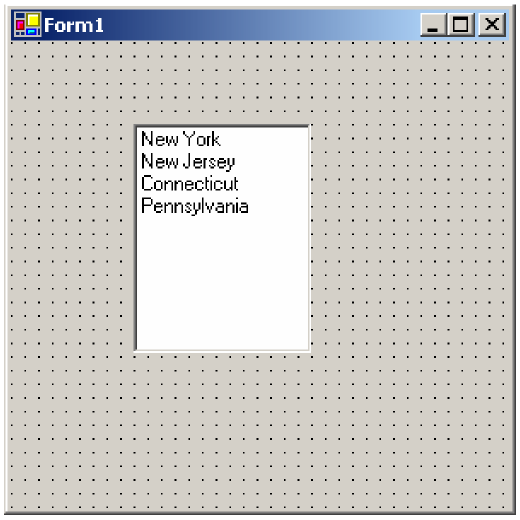
Step 3: In the "String Collection Editor" simply using the keyboard enter each string. Press "Enter" after each entry. When finished simply click OK and the items will now reside inside the ListBox.



String Collection Editor

Enter the strings in the collection (one per line):

New York
New Jersey
Connecticut
Pennsylvania



Populating the List at Run Time

- ❑ If your list is going to change while the program is running, then you need to write code to make this happen.
- ❑ Populating the list at Run time requires the following code:

Items.Add() Method

- The **Items.Add(….)** method is used to add an Item at the end of the list.
- The value added to the list is a *character string* that is placed inside the parentheses (*ItemValue*...)
- Since the value is added to the end of the list, you can use the **Sorted Property** to sort alphabetically the items is the list after the addition.
- The syntax:

```
'Adding an Item value to the list
ListBox.Items.Add( ItemValue )
```

Example:

❑ **Add Items to List:**

- Example 1 – *Adding the States to the list*:

```
lstStates.Items.Add("New York")
lstStates.Items.Add("New Jersey")
lstStates.Items.Add("Connecticut")
lstStates.Items.Add("Pennsylvania")
```

- Example 2 – *Adding schools to a listing of schools using With Block Statement*:

```
With lstSchools.Items
    .Add("Harvard University")
    .Add("Standford")
    .Add("New York City Technical College")
End With
```

- Example 3 – *Adding items to list and then sorting*:

```
With lstSchools.Items
    .Add("Harvard University")
    .Add("Standford")
    .Add("New York City Technical College")
End With
lstSchools.Sorted = True
```

Items.Insert() Method

- The **Items.Insert(….)** method is used to add an Item to a desired location on the list based on the Index location.
- The value added to the list is a *character string* that is placed inside the parentheses and the Index location where you want to store the list is also within the parentheses separated by a comma (*IndexPosition, ItemValue*)
- The syntax:

```
'Adding an Item value to the list
ListBox.Items.Insert(IndexValue,  ItemValue )
```

Example:

- **Inserting Items to the List at a specified location:**

  - Example 1 – *Inserting a States item to the index location*:

```
lstStates.Items.Insert(2,"New York")
lstStates.Items.Insert(1"New Jersey")
lstStates.Items.Insert(0"Connecticut")
lstStates.Items.Insert(3"Pennsylvania")
```

  - Example 2 – *Inserting schools to a listing of schools using With Block Statement*:

```
With lstSchools.Items
    .Insert(5,  "Harvard University")
    .Insert(2,  "Standford")
    .Insert(0,  "New York City Technical College")
End With
```

Retrieving Items from the Items Object Property

- You need to use the **Items(*Index*)** statement to retrieve an item from the location indicated by the index.
- Note that you need to specify the index of the item you wish to retrieve from the list
- The syntax:

```
'Getting items from the list
variable = ListBox.Items( Index )
```

Example:

- **Retrieving Items from the List:**
  - Example 1 – *retrieving a items from the list*:
    myState = lstStates.Items*(2) 'the variable myState now holds the string from the list*
    value = lstSchool.Items*(0) ' the value variable now holds the string from the list*

Retrieving Selected Items from the Items Object Property

- If you need to retrieve an item that is selected by the user, which is usually the case, you need to also include the **SelectedIndex Property** of the Items Object in parentheses instead of the index.
- The syntax:

```
'Getting items from the list
variable = ListBox.Items( ListBox.SelectedIndex )
```

Example:

- **Retrieving the Items selected by the User from the List:**
  - Example 1 – *retrieving a items selected by user and assigning to variables*:
    myState = **lstStates.Items***(**lstState.**SelectedIndex)*
    value = **lstSchool.Items***(**lstSchool.**SelectedIndex)*

Removing Items from the List by Location

- You can remove an item from the list that is located at a specific index location.
- The syntax:

```
'Removing an Item at the specified index
ListBox.Items.RemoveAt(Index )
```

Example:

❑ **Removing an Items at a specified location:**

- Example 1 – *Removing an items from the list*:
  lstStates.Items.*RemoveAt(2) 'the item in location 2 is removed from the list*
  lstSchool.Items.*RemoveAt(0) 'the item in location 0 is removed from the list*

Removing Items from the List that was Selected by the User

- Normally we want to remove an item that is selected by the user.
- You can remove an item from the list that is selected by the user by using the **SelectedIndex Property** of the Items Object in parentheses <u>instead</u> of the index.
- The syntax:

```
'Removing an item selected by the user
ListBox.Items.RemoveAt( ListBox.SelectedIndex )
```

Example:

❑ **Retrieving the Items selected by the User from the List:**

- Example 1 – *retrieving a items selected by user and assigning to variables*:
  lstStates.Items.*RemoveAt(***lstState.***SelectedIndex)*
  lstSchool.Items.*RemoveAt(* ***lstSchool.***SelectedIndex)*

Removing Items from the List by string pattern matching

- You can remove an item from the list that matches a character string
- The syntax:

```
'Adding an Item value to the list
ListBox.Items.Remove(TextString )
```

Example:

- **Removing an Items at a specified location:**
  - Example 1 – *Removing an items from the list by matching a string*:
    lstStates.Items.*Remove("New York") 'Remove the matching string*
    lstSchool.Items.*RemoveAt("Hardvard") 'Removes the matching string*

Clearing a List

- You can clear or remove all items from a list
- The syntax:

```
'Clearing an item from the list using the Clear Method
ListBox.Items.Clear()
```

Example:

- **Removing an Items at a specified location:**
  - Example 1 – *Removing an items from the list by matching a string*:
    lstStates.Items.*Clear() 'Remove all items from the list*

List Boxes Methods & Coding

- The following table is a list of some of the most common List Box Methods:

Common List Box Methods:

Hide	Conceals the control from the user.
Show	Displays the control to the user.

Method Coding Syntax:

```
 'Calling a Method
List BoxControl.Method()
```

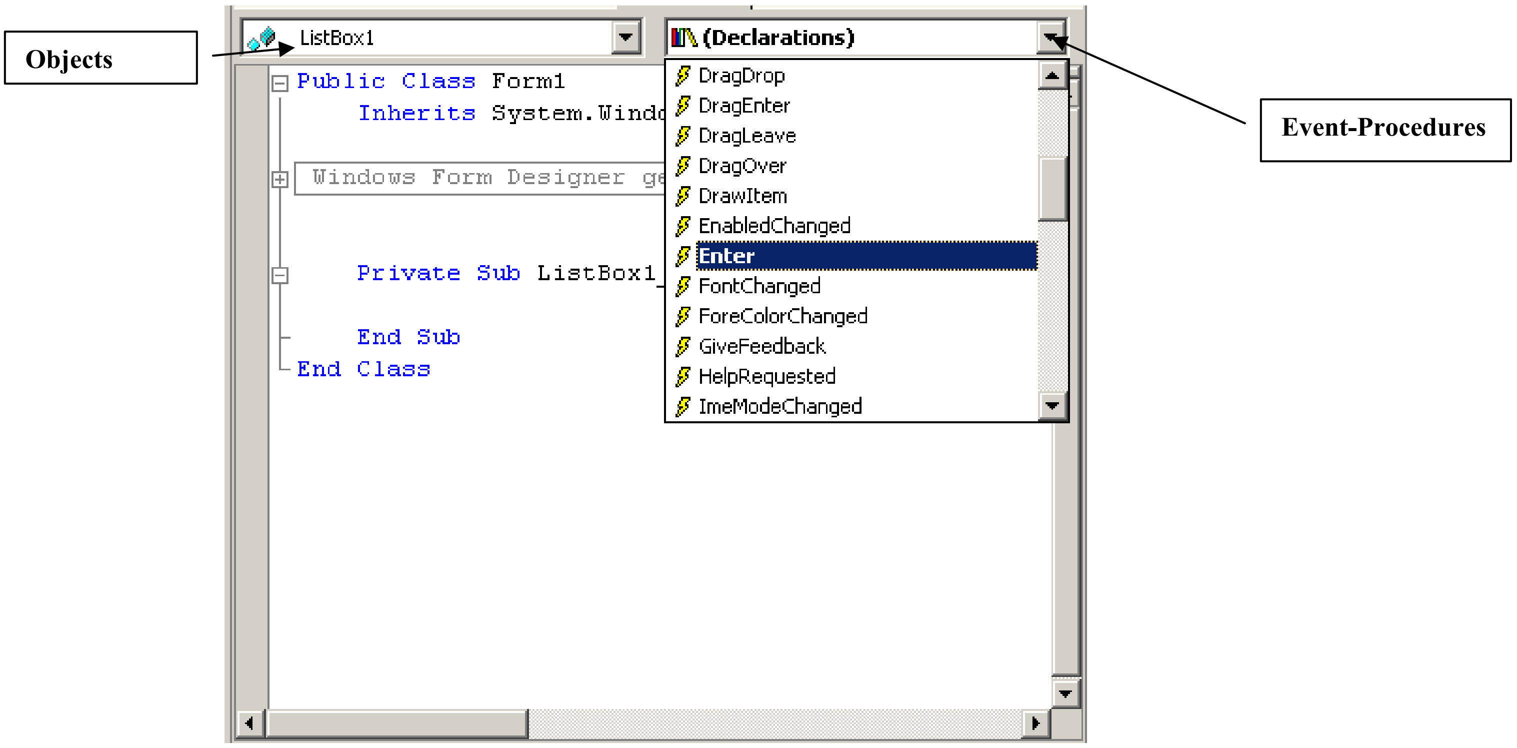
Example:

- **Executing or Calling a Method:**

- Example 1:
  lstStates.*Show()*

List Box Event, Event-Procedures & Coding

- Events are actions taken by the user upon the control which trigger an automatic execution of an ***Event-Procedure*** Method.
- You can view the list of Event in the Code Editor Window by selecting the **List Box Object** in the *Object Drop-Down List Box* & available events in the *Declaration Drop-Down List Box*:



```
Public Class Form1
    Inherits System.Windo
Windows Form Designer ge

    Private Sub ListBox1_

    End Sub
End Class
```

- The following table is a list of some of the most common List Box Events & Event-Procedures:

Common List Box Events:

Enter	Occurs when the control is entered.
KeyDown	Occurs when a key is pressed while the control has focus.
KeyPress	Occurs when a key is pressed while the control has focus.
KeyUp	Occurs when a key is released while the control has focus.
Leave	Occurs when the input focus leaves the control.
LostFocus	Occurs when the control loses focus.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseEnter	Occurs when the mouse pointer enters the control.
MouseHover	Occurs when the mouse pointer hovers over the control.
MouseMove	Occurs when the mouse pointer is moved over the control.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.

Event Coding Syntax:

```
 'Procedure- Declaration Statement or Procedure Header
Private Sub ControlObject1_Event(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ControlObject1.Event

'Place code that you want to execute when the event is trigger inside this body between the declaration and End Sub marking

End Sub
```

❖ Note that the header declaration looks complex, don't worry, this is automatically generated by Visual Basics.NET

Example:

❑ **Coding the Text_Changed() Event:**

▪ Example: *Suppose you want assign to a label text indicating that the listbox has been entered and it's being used*

```
Private  Sub  lstStates_Enter(ByVal  sender  As  System.Object,  ByVal  e  As  System.EventArgs)  Handles
lstStates_Enter

        lblInformation.Text = "ListBox is currently being accessed

End Sub
```