Programming in C#
Unit-2

Dr.M.Paul Arokiadass Jerald

Assistant Professor
Department of Computer Science
Periyar Arts College, Cuddalore-1

Syllabus Overview

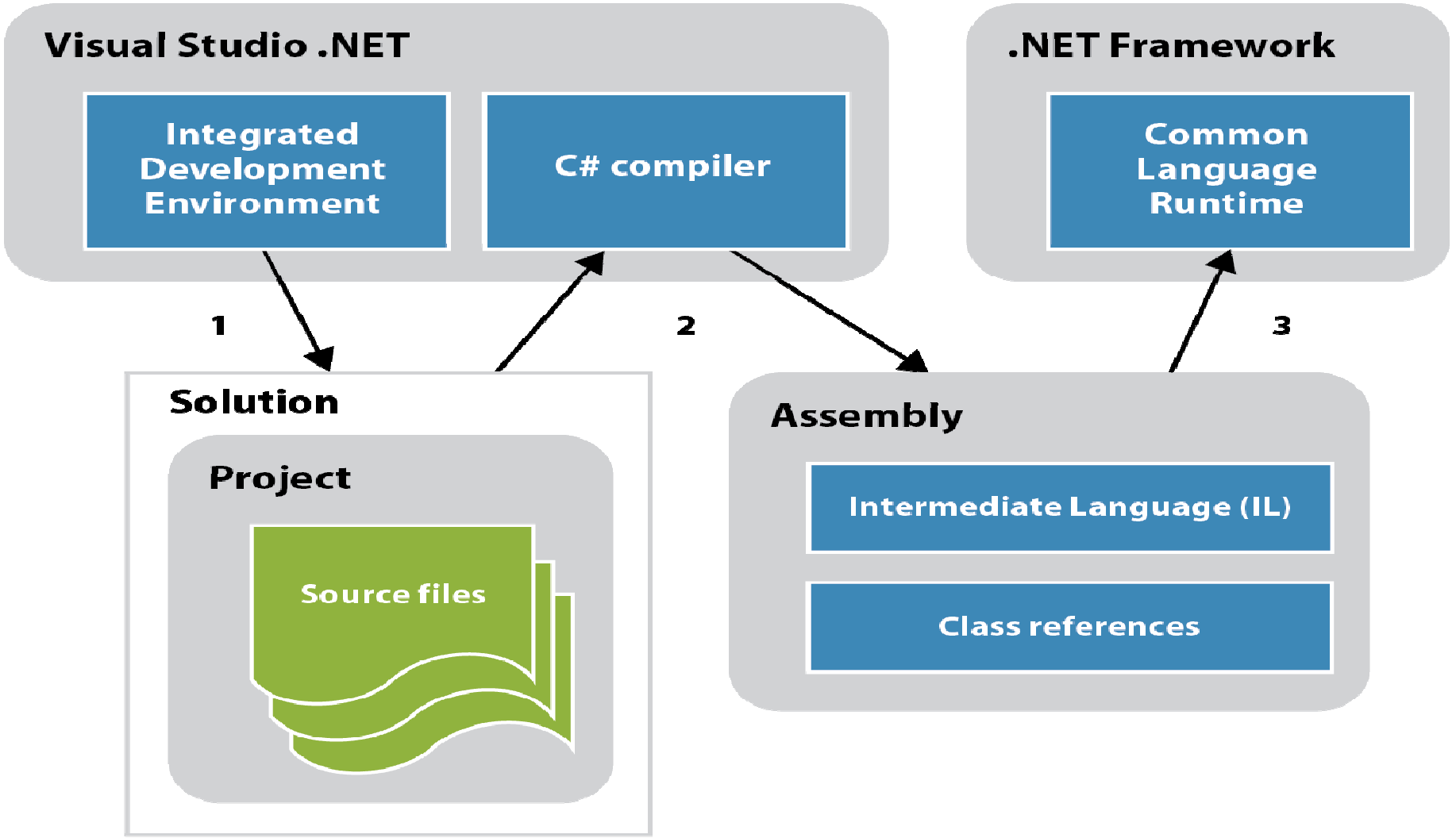
UNIT - II: WINDOWS FORMS

Windows Forms – Form Class – Common Operations on Forms – Creating a Message Box – Handling Events – Mouse Events – Keyboard Events – Common Controls in Windows Forms – Label – TextBox – Button – Combo Box – List Box – Check Box – Radio Button – Group Box – Picture Box – Timer – Open File Dialog – Save File Dialog – Font Dialog – Color Dialog – Print Dialog – Tree View – Menu.

.NET application Types

Type	Description
Windows Forms	Runs in its own window on the user's PC and consists of one or more Windows.
ASP.NET Web Forms	Runs on a web server and consists of one or more web forms that define pages that are displayed in a browser on the client.
ASP.NET MVC	Runs on a web server and consists of pages that are displayed in a browser on the client machine. Unlike ASP.NET Web Forms, ASP.NET MVC uses a Model-View-Controller design pattern to create its pages.
WPF	Runs in windows on the user's PC and provides an enhanced user experience.

Compiling C# Programs



1. Windows Forms

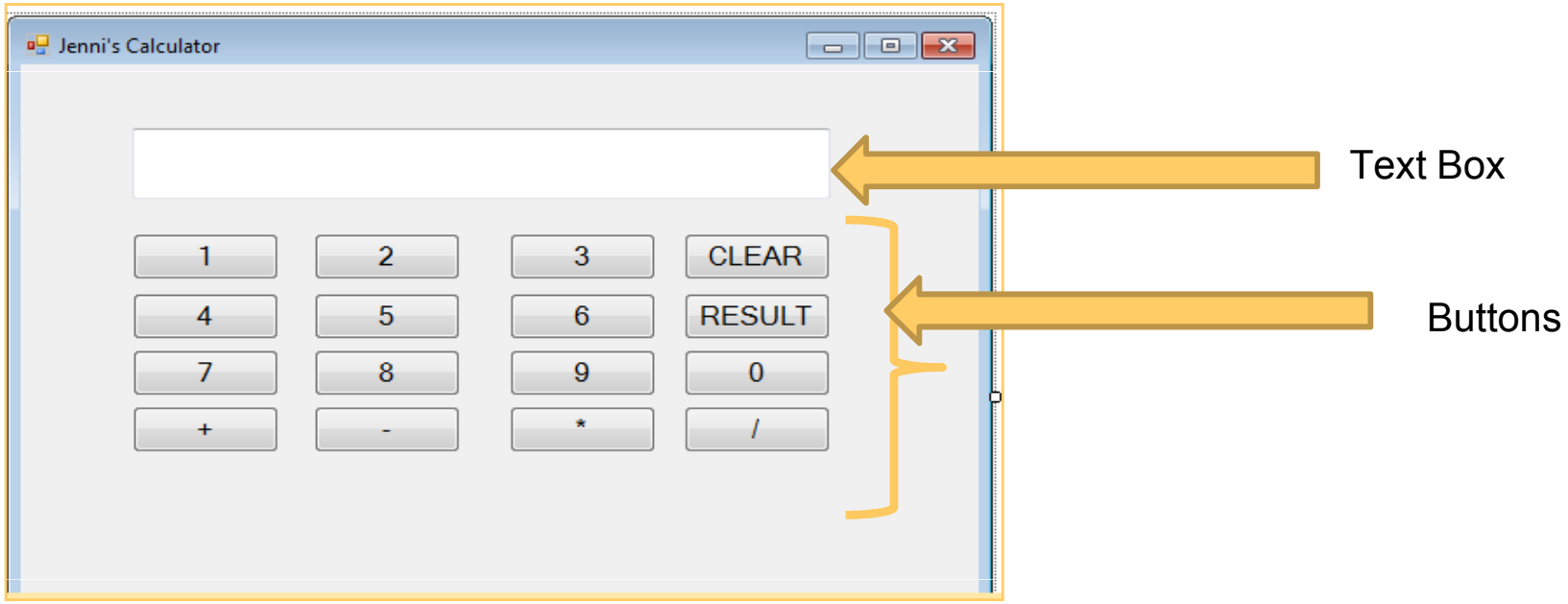
- A windows form application is an application, which is designed to run on a computer.
- Windows Forms is a Graphical User Interface(GUI) class library which is bundled in .Net Framework.
- A Windows forms application will normally have a collection of controls such as labels, textboxes, list boxes, etc.
- Create graphical elements that appear on the desktop (dialog, window, etc.)
- It will **run on desktop** and not on web browser (if it runs on browser then it becomes a web application.)

Windows Forms

- GUI (Graphical User Interface)
  - Allows the user to interact visually with a program
  - GUI's are built from GUI controls
  - Also known as components or widgets
- They are objects that can Display information on the screen, or Enable users to interact with an application via the mouse, keyboard or other form of input
- Commonly used types of GUI controls
  - Label, TextBox, Button, CheckBox ComboBox, ListBox etc.

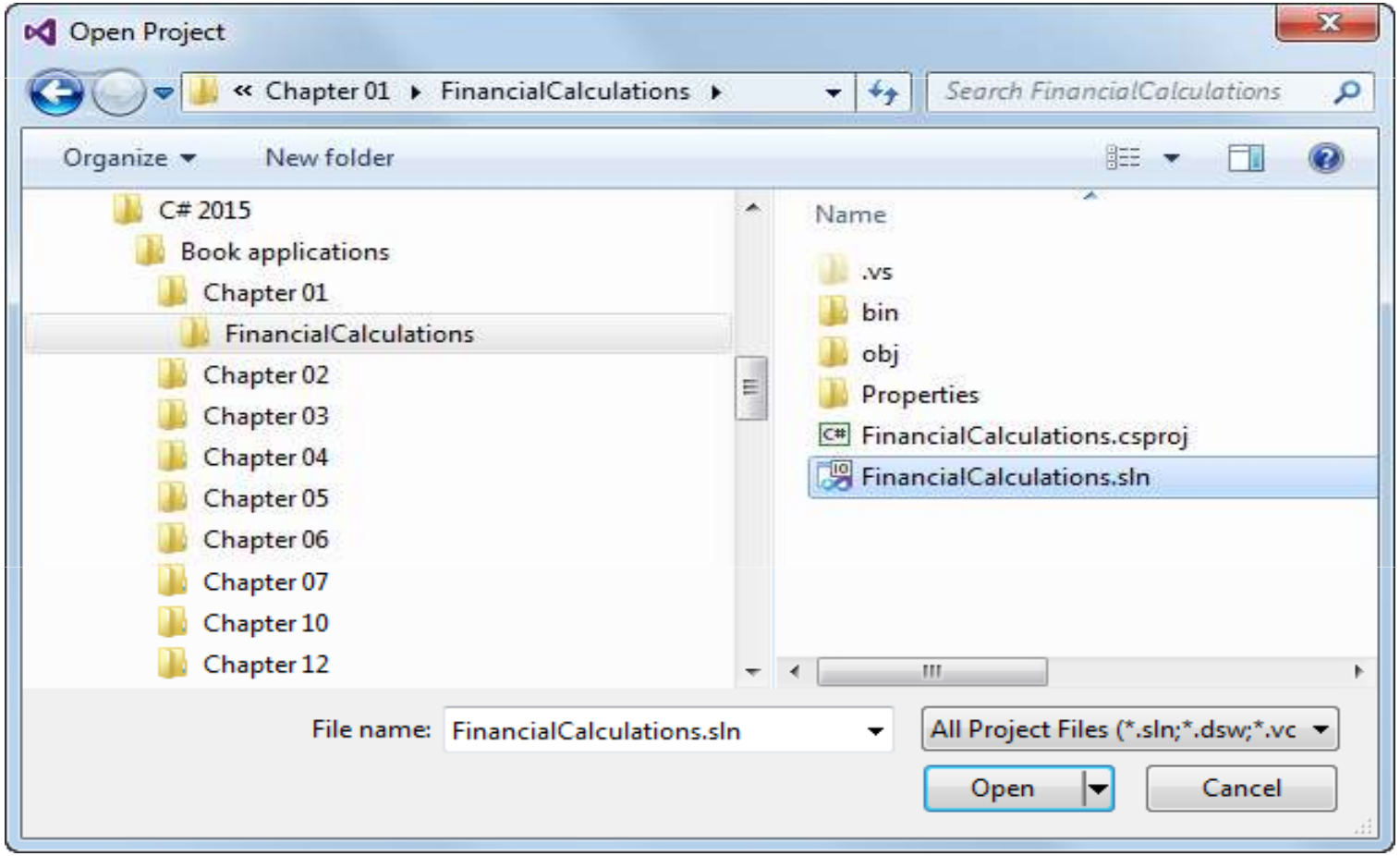
Windows Forms- Sample

Dr.M.Paul Arokiadass Jerald



Screen Shot from University Practical – Exercise 5

Open Project Dialog



2. Windows Forms in IDE

- **Window or Main Window:** For working with forms and code editing. Initially it is blank. Double click the form then to open the code.
- **Solution Explorer Window:** It is used to navigate between all items in solution.
- **Properties Window:** This window is used to change the different properties of the selected item in the Solution Explorer. Used to change the properties of components or controls that is added to the forms.

Project and Solution File

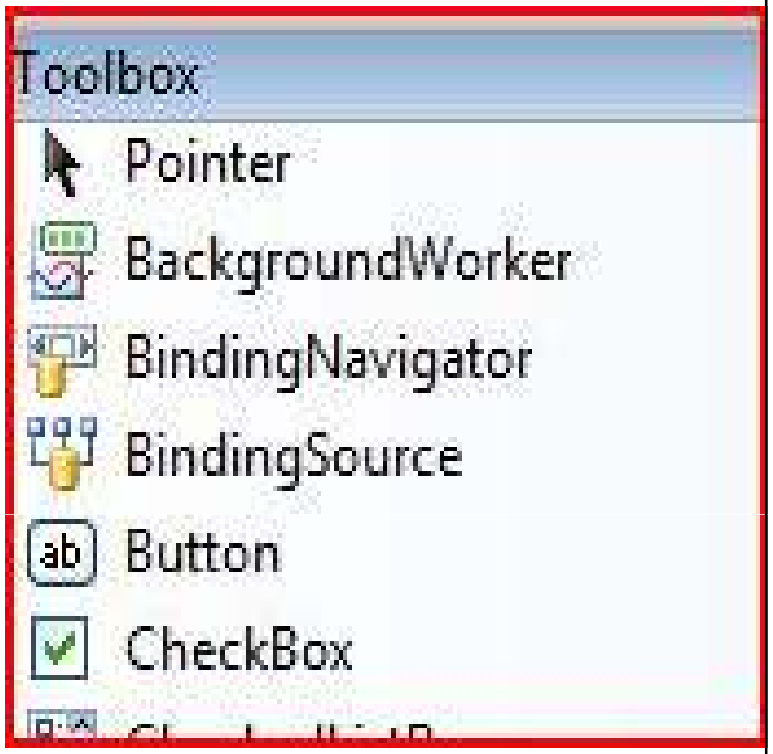
- Every C# **project** has a ***project file*** with an extension of *csproj* that keeps track of the files that make up the project and records various settings for the project.
- Every **solution** has a ***solution file*** with an extension of *sln* that keeps track of the projects that make up the solution.
- When you open a project file, Visual Studio opens the solution that contains the project. And when you open a solution file, Visual Studio automatically opens all the projects contained in the solution.
- The project file can be stored in the same directory as the solution file or in one of its subdirectories.

3. Form Class

- using System.Windows.Forms;
- This class defines the basic functionality of the controls, which is why many properties and events in the controls are identical.
- Many of these classes are themselves base classes for other controls.

4. ToolBox

- The controls and components of C# are found in the C# Toolbox in Visual Studio
  - Organized by functionality
- To open Toolbox (takes time!):
  - Menu/Item: View>>Toolbox:
- To add a component to a Form:
  - Select that component in Toolbox
  - Drag it onto the Form



5. Properties and Methods for GUI Controls

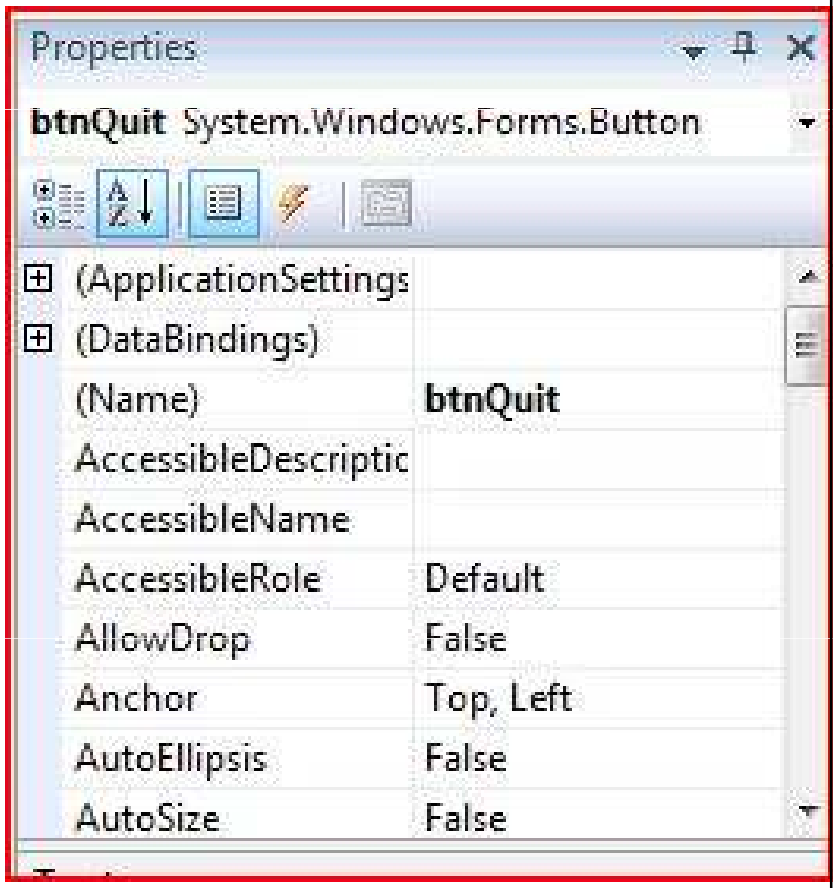
- Each control has *properties* (and some *methods*)
  - Example properties:
    - Enable
    - Font
    - Text
    - Visible
  - Example methods:
    - Hide
    - Select
    - Show

PROPERTIES

- Anchor
- BackColor
- Bottom
- Dock
- Enabled
- ForeColor
- Height
- Left
- Name
- Parent
- Right
- TabIndex
- TabStop
- Tag
- Top
- Visible
- Width

6. Editing the Properties

- Click on the control for which you want to change the properties
  - E.g., click on form, button, label, etc.
- You can make these changes in the **Properties** window located on the bottom right



7. Naming Controls

- C#, *default names* for controls/components are:
  - button1, label1, textbox1, etc.
  - not very descriptive (“generic”)
- Use better, descriptive names
  - Names to have meanings that make sense

Control	Begin name with
Button	btn
TextBox	txt
ListBox	lbox
Label	lbl
SaveFileDialog	sfd

8. Creating a Message Box

- C# MessageBox in Windows Forms displays a message with the given text and action buttons.
- MessageBox class has an overloaded static Show method that displays a message box with a message and action buttons OK and Cancel, Yes and No etc.
- **SIMPLE MESSAGE BOX**

```
string message = "Simple MessageBox";
MessageBox.Show(message);
```

- **MESSAGE BOX WITH TITLE**

```
string message = "Simple MessageBox";
string title = "Title";
MessageBox.Show(message, title);
```

9. Handling Events

- Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications.
- Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the **publisher** and the classes that receive (or handle) the event are called **subscribers**.

Mouse Events in C#

- **Click** : This event occurs when the mouse button is released, typically before the MouseUp event.
- **MouseClick** : This event occurs when the user clicks the control with the mouse.
- **DoubleClick** : This event occurs when the control is double-clicked.
- **MouseDoubleClick** : This event occurs when the user double-clicks the control with the mouse..
- **MouseDown** : This event occurs when the mouse pointer is over the control and the user presses a mouse button. The h
- **MouseHover** : This event occurs when the mouse pointer stops and rests over the control.
- **MouseLeave, MouseMove, MouseUp etc.**

Keyboard Events in C#

- Windows Forms processes keyboard input by raising keyboard events in response to Windows messages.
- Windows Forms provides two events that occur when a user presses a keyboard key and one event when a user releases a keyboard key:
  - The **KeyDown** event occurs once.
  - The **KeyPress** event, which can occur multiple times when a user holds down the same key.
  - The **KeyUp** event occurs once when a user releases a key.

10. Common Controls in Windows Forms

Function	Control	Description
Text edit	Text Box	Displays text entered at design time that can be edited by users at run time, or changed programmatically.
	RichTextBox	Enables text to be display with formatting in plain text or rich - text format (RTF).
Text display (read-only)	Label	Displays text that users cannot directly edit.
	LinkLabel	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.
	StatusBar	Displays information about the application s current state using a formed window, usually at the bottom of a parent form.
Selection from a list	CheckedListBox	Displays a scrollable list of items, of items,each accompanied by a check box.
	ComboBox	Displays a drop-down list of items.
	DomainUpDown	Displays a list of text items that users can scroll through with up and down buttons.
	ListBox	Displays a list of text and graphical items (icons).
	ListView	Displays items in one of four different views. Views include text only, text with small icons, text with large icons, and a report view.
	NumericUpDown	Displays a list of numerals the users can scroll through with up and down buttons.

Common Controls in Windows Forms

	Tree View	Display s hierarchical collection of node objects which can consist of text with optional check boxes or icons.
Graphics display	PictureBox	Displays graphical files, such as bitmaps and icons, in a frame.
Graphics storage	ImageList	Serves as a repository for images. Image List controls and the image they contain can be reused frame one application to the next.
Value setting	CheckBox	Displays a check box and a label for text Generally used to set options.
	CheckedListBox	Displays a scrollable list of items, each accompanied by a check box.
	RadioButton	Displays a button that can be turned on or off.
	Trackbar	Allows users to set values on a scale by moving a thumb along a scale.
Data setting	Data TimePicker	Displays a graphical calendar to allow users to select a data or a time.
	Month Calendar	Displays a graphical calender to allow users to select a range of dates.
Dialog boxes	Color Dialog	Displays the color picker dialog box that allows users to set the color of an interface element.
	FontDialog	Displays a dialog box that allows users to set font to and select a file.
	OpenFileDialog	Displays a dialog box that allows users to select a printer and set its attributes.
	PrintPreviwDialog	Displays a dialog box that Displays how a Print Document object will appear when printed.
	SaveFielDialog	Displays a dialog box that allows users to save a file.
Menu controls	MainMenu	Provides a design - time interface for creating menus.
	ContextMenu	Implements a menu that appears when the user right-chicks an object.
Commands	Button	Used to start, stop, or interrupt a process.
	LinkLabel	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site
	NotifyIcom	Displays an icon in the status notification are of the taskbar that represents an application running in the background.
	ToolBar	Contains a collection of button controls.

Label

- The label is a simple to present a caption or short hint to explain something on the form to the user.
- .NET includes two label controls
  - **Label**, the standard Windows label
  - **LinkLabel**, a label like the standard one (and derived from it), but presents itself as an internet link (a hyperlink)
- **LABEL PROPERTIES**
  - BorderStyle, DisabledLinkColor, FlatStyle, Image, ImageAlign, LinkArea, inkColor, Links, LinkVisited, Text, TextAlign

TextBox

- The primary function of a text box is for the user to enter text, but any characters can be entered, and it is quite possible to force the user to enter numeric values only.
- .NET comes with two basic controls to take text input from the user: TextBox and RichTextBox.
- Both controls are derived from base class called TextBoxBase which is derived from Control.
- **TextBox events :** Enter, GotFocus, Leave, Validating, Validated, LostFocus, KeyDown, KeyPress, KeyUp, Change, Enter, GotFocus

RichTextBox

- RichTextBox control is derived from TextBoxBase.
- RichTextBox is used to display and enter formatted text (for example bold, underline, and italic).
- It does so using a standard for formatted text called Rich Text Format or RTF.
- Rich Text Box Properties
  - LinkedClick
  - Protected
  - SelectionChanged

Button

- Button class is not derived directly from Control, but from another class called ButtonBase.
- A button is primarily used to perform three kinds of tasks:
  - To close a dialog with a state
  - To perform an action on data entered on a dialog
  - To open another dialog or application
- Button Event
  - Click, RightClick, DoubleClick
  - Default event is Click.

Combo Box

- ComboBox control provides combined functionality of a text box and a listbox in a single control.
- Only one list item is displayed at one time in a ComboBox and rest of the available items are loaded in a drop down list.
- Code for creating a ComboBox control object.

```
ComboBox comboBox1 = new ComboBox();
ComboBox Items
```

Combo Box - Property

- **Property : items**
- The Items property is used to add and access items in a ComboBox.
- We can add items to a ComboBox at design-time from Properties Window by clicking on Items Collection add same items at run-time by using the following code snippet.
  - comboBox1.Items.Add("Mahesh");
  - comboBox1.Items.Add("Rahul");
  - comboBox1.Items.Add("Kumar");
  - comboBox1.Items.Add("Rajesh");

Combo Box - Property

- **Property : DropDownStyle**
- DropDownStyle property is used to gets and sets the style of a ComboBox.
  - Simple - List is always visible and the text portion is editable.
  - DropDown – List is displayed by clicking the down arrow and that the text portion is editable.
  - DropDownList - List is displayed by clicking the down arrow and that the text portion is not editable.
- **Property : Sorted**
- Set to true, the ComboBox items are sorted.
  - comboBox1.Sorted = true;

Combo Box – Reading item

- **Reading items from Combo Box :**

```
private void GetItemsButton_Click(object sender, EventArgs e)
{
StringBuilder sb = new StringBuilder();
foreach (string name in Combo1.Items)
   {
      sb.Append(name);
      sb.Append(" ");
   }
MessageBox.Show(sb.ToString());
}
```

List Box

- ListBox control is used to show multiple elements in a list, from which a user can select one or more elements.
- The ListBox class is used to represent the windows list box and also provide different properties, methods, and events.
- Defined in *System.Windows.Forms* namespace.
- Create a ListBox control using the ListBox() constructor:

```
ListBox mylist = new ListBox();
```

- **Adding items in ListBox**

```
mylist.Items.Add("Grapes");
mylist.Items.Add("Orange");
mylist.Items.Add("Apple");
```

List Box

- **Selected Text and Item of a ListBox**
- The Text property is used to set and get text of a ListBox.
- Code snippet to set and get the current text of a ListBox:

```
MessageBox.Show(listBox1.Text);
```

- Get the currently selected item using Items property:

```
string selectedItem = listBox1.Items[listBox1.SelectedIndex].ToString();
```

List Box – Find a String

- FindString method is used to find a string or substring in a ListBox.
- Code to find a string in a ListBox and select if found:

```
private void FindItemButton_Click(object sender, EventArgs e)
{     listBox1.ClearSelected();
      int index = listBox1.FindString(textBox1.Text);
      if (index < 0)
   {   MessageBox.Show("Item not found.");
       textBox1.Text = String.Empty;
   }
   else
   {         listBox1.SelectedIndex = index;
   } }
```

Check Box

- CheckBox control allows to select single or multiple elements from the given list or it can provide options like yes or no, true or false, etc.
- It can be displayed as an image or text or both.
- The CheckBox is defined in
  System.Windows.Forms namespace.
- **Create a checkbox** using CheckBox() constructor
  CheckBox Mycheckbox = new CheckBox();

Check Box – Read Contents

- Read CheckBox Contents using the Text property.

```
string CheckBoxContents = dynamicCheckBox.Text;
```

- **Appearance property of CheckBox :** can be used to set the appearance of a CheckBox to a Button or a CheckBox.

- Property to make CheckBox look like Button control.

```
dynamicCheckBox.Appearance = Appearance.Button;
```

Image in Check Box

- The Image property of a CheckBox control is used to set the background as an image.
- Code to set an image as a CheckBox background.

```
dynamicCheckBox.Image =
Image.FromFile(@"C:\Images\logo.jpg");
```

Radio Button

- A radio button or option button enables the user to select a single option from a group of choices when paired with other RadioButton controls.
- When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked.
- RadioButton is a class and it is defined under ***System.Windows.Forms*** namespace.
- Create a radio button using Code :

```
RadioButton r1 = new RadioButton();
```

Radio Button

- **Radio Button events :** Click, CheckedChanged, AppearanceChanged, DoubleClick, Leave, MouseClick, MouseDoubleClick, MouseHover, MouseLeave
- **Image in RadioButton :** The Image property of a RadioButton control is used to set the background as an image.

  dynamicRadioButton.Image = Image.FromFile(@ "C:\Images\trans.jpg");

Group Box

- GroupBox is a container which contains multiple controls on it.
- Group Box is often used in conjunction with the RadioButton and CheckBox controls to display a frame around, and a caption above.
- Group box can itself contain controls, it becomes the parent of these controls.
- Moving the GroupBox will move all of the controls placed on it.
- Parent – Child Relationship

Picture Box

- Windows Forms PictureBox control is used to display images in bitmap, GIF , icon , or JPEG formats.
- **Image property** is set to display the image, either at design time or at run time.

  pictureBox1.Image = Image.FromFile("c:\\Image.jpg");
- **PictureBoxSizeMode Property.**

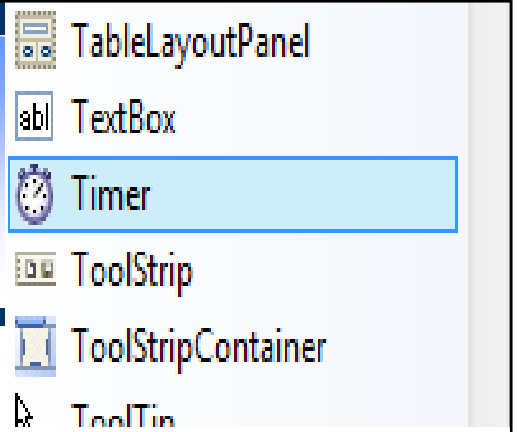
  AutoSize - Sizes the picture box to the image.

  CenterImage - Centers the image in the picture box.

  Normal – Places the image at upper left in picture box

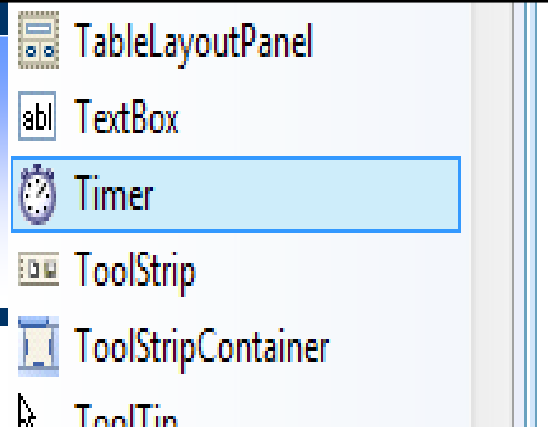
  StretchImage - Allows you to stretch the image in code

Timer



- Timer Control can raise events at a specific interval of time without the interaction of another thread.
- The Timer class in C# represents a Timer control that executes a code block at a specified interval of time repeatedly.
- Timer control does is not shown in form and works as a component in the background.
- Timer Control allows to set **Interval property** in milliseconds. (one second is equal to 1000 milliseconds.)

Timer – Properties



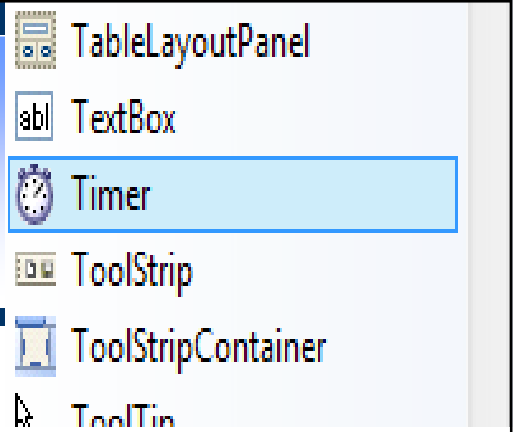
- **Timer Interval Property :** gets or sets the time, in milliseconds, before the Tick event is raised relative to the last occurrence of the Tick event .

```
// Sets the timer interval to 1 seconds.
myTimer.Interval = 1000;
```

- Timer Reset Property : gets or sets a Boolean whether the Timer should raise the Elapsed event only once (false) or repeatedly (true).

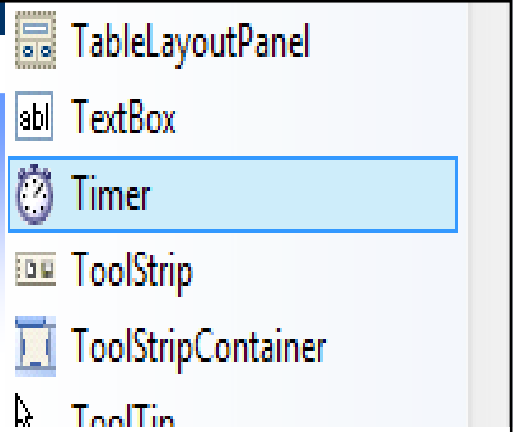
```
MyTimer.AutoReset = false;
```

Timer – Events



- **Timer Tick Event:** occurs when the specified timer interval has elapsed and the timer is enabled.
- **Timer Elapsed Event:** occurs when the interval elapses. The Elapsed event is raised if the Enabled property is true and the time interval (in milliseconds) defined by the Interval property elapses.

Timer – Delegate



TimerCallback Delegate

- Callback represents the method that handles calls from a Timer. This method does not execute in the thread that created the timer; it executes in a separate thread pool thread that is provided by the system.

11. Dialog Boxes in C#

- A dialog box is a type of window, which is used to enable common communication or dialog between a computer and its user.
- A dialog box used to provide the user with the means for specifying how to implement a command or to respond to a question.
- **Windows.Form** is a base class

Dialog Boxes in C#

- **Two types Dialog box**
  1. **Modal Dialog box** : A dialog box that temporarily halts the application and the user cannot continue until the dialog has been closed is called modal dialog box.
  2. **Modeless dialog box** :is used when the requested information is not essential to continue, so the Window can be left open, while work continues
- **Dialog Boxes in C#**
  1. FontDialog
  2. ColorDialog
  3. OpenDialog
  4. SaveDialog
  5. PrintDialog

Open File Dialog

- Displays a standard dialog box that prompts the user to open a file. This class cannot be inherited.
- Creating using Design :
- Creating in Runtime (Code)

openFileDialog1 = new OpenFileDialog();

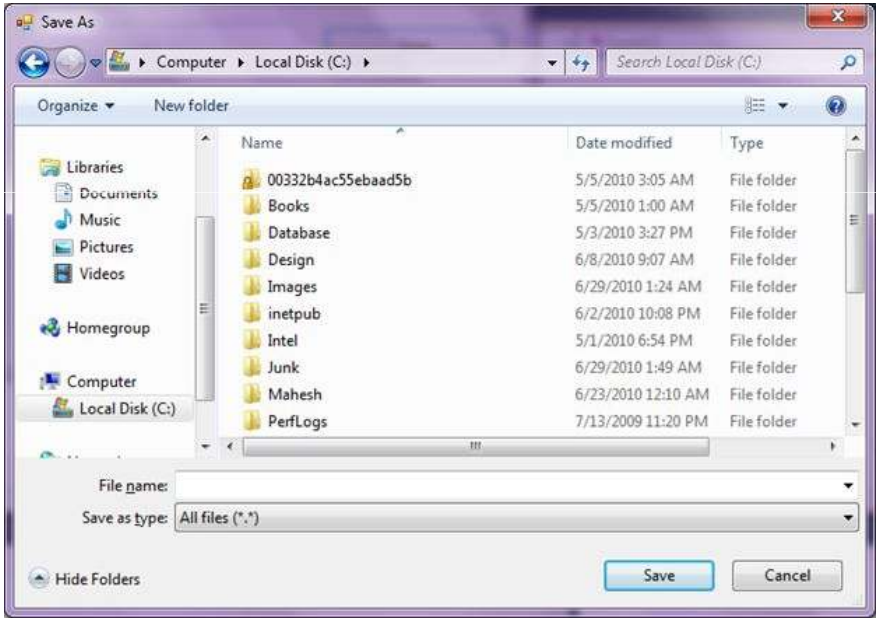
openFileDialog1.ShowDialog();

Open File Dialog

- **Open File Dialog – Property**
  - **Initial and Restore Directories :** represents the directory to be displayed when the open file dialog appears first time.
  - **Title :** used to set or get the title of the open file dialog.
  - **Filter and Filter Index :** Filter property represents the filter on an open file dialog that is used to filter the type of files
- **Open File Dialog EVENTS :**
  - Disposed, FileOk, HelpRequest

Save File Dialog

- Prompts the user to select a location for saving a file.
- A SaveFileDialog control is used to save a file using Windows SaveFileDialog.



Font Dialog

- A FontDialog control in WinForms Prompts the user to choose a font from among those installed on the local computer
- Font Dialog has a list of fonts, styles, size and other options.
- Adding a Font dialog (Design time) : Drag and drop
- Code to Add a FontDialog to a Form (Runtime)

```
private System.Windows.Forms.FontDialog fontDialog1;
this.fontDialog1 = new System.Windows.Forms.FontDialog();
FontDialog fontDlg = new FontDialog();
fontDlg.ShowDialog();
```

Font Dialog – Methods, Events

- **Font Dialog Methods :**

CreateObjRef(Type), Dispose(), Equals(Object), GetHashCode(), GetLifetimeService(), GetService(Type), GetType(), HookProc(IntPtr, Int32, IntPtr, IntPtr), MemberwiseClone(), InitializeLifetimeService(), OnHelpRequest(EventArgs), Reset(), RunDialog(IntPtr), ShowDialog(), ToString()

- **Font Dialog : Events**
  - **Apply** : Occurs when the user clicks the Apply button in the font dialog box.
  - **Disposed** : Occurs when the component is disposed by a call to the Dispose() method.
  - **HelpRequest** : Occurs when the user clicks the Help button on a common dialog box.

Font Dialog - Example

```
private void button1_Click(object sender, System.EventArgs e)
 {
    fontDialog1.ShowColor = true;
    fontDialog1.Font = textBox1.Font;
    fontDialog1.Color = textBox1.ForeColor;
    if(fontDialog1.ShowDialog() != DialogResult.Cancel )
    {
       textBox1.Font = fontDialog1.Font ;
       textBox1.ForeColor = fontDialog1.Color;
    }
 }
```

Color Dialog

- Represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
- Adding a Color dialog (Design time) : Drag and drop
- Code to Add a ColorDialog to a Form (Runtime)

```
private System.Windows.Forms.ColorDialog colorDialog1;
this.colorDialog1 = new System.Windows.Forms.ColorDialog();
ColorDialog colorDlg = new ColorDialog();
colorDlg.ShowDialog();
```

Color Dialog

- **Color Dialog Properties :** AllowFullOpen, AnyColor, CanRaiseEvents, Color, Container, CustomColors, DesignMode, Events, FullOpen, Instance, Options, ShowHelpSite, SolidColorOnly, Tag
- **Color Dialog Methods :** CreateObjRef(Type), Dispose(), Equals(Object), GetHashCode(), GetLifetimeService(), GetService(Type), GetType(), HookProc(IntPtr, Int32, IntPtr, IntPtr), MemberwiseClone(), InitializeLifetimeService(), OnHelpRequest(EventArgs), Reset(), RunDialog(IntPtr), ShowDialog(), ToString()
- **Color Dialog : Events**
  - **Apply** : Occurs when the user clicks the Apply button in the font dialog.
  - **Disposed** : Occurs when the component is disposed by a call to the Dispose() method.
  - **HelpRequest** : Occurs when the user clicks the Help button on a common dialog box.

Color Dialog - Example

```
private void button1_Click(object sender, System.EventArgs e)
 {    ColorDialog MyDialog = new ColorDialog();
    // Keeps the user from selecting a custom color.
    MyDialog.AllowFullOpen = false ;
    // Allows the user to get help. (The default is false.)
    MyDialog.ShowHelp = true ;
    // Sets the initial color select to the current text color.
    MyDialog.Color = textBox1.ForeColor ;
    // Update the text box color if the user clicks OK
    if (MyDialog.ShowDialog() == DialogResult.OK)
        textBox1.ForeColor =  MyDialog.Color;
 }
```

Print Dialog

- A PrintDialog control is used to open the Windows Print Dialog and let the user select the printer, set printer and paper properties, and print a file.
- Adding a Print dialog (Design time) : Drag and drop
- Code to Add a PrintDialog to a Form (Runtime)

```
PrintDialog PrintDialog1 = newPrintDialog();
PrintDialog1.ShowDialog();
```

Print Dialog

- **Print Dialog Properties :** ShowCurrentPage, AllowPrintToFile, AllowSelection, AllowSomePages, CanRaiseEvents, Container, DesignMode, Document, Events, PrinterSettings, PrintToFile, ShowHelp, ShowNetwork, Site, Tag, UseEXDialog
- **Print Dialog Methods :** CreateObjRef(Type), Dispose(), Equals(Object), GetHashCode(), GetLifetimeService(), GetService(Type), GetType(), HookProc(IntPtr, Int32, IntPtr, IntPtr), MemberwiseClone(), InitializeLifetimeService(), OnHelpRequest(EventArgs), Reset(), RunDialog(IntPtr), ShowDialog(), ToString()
- **Print Dialog : Events**
  - **Disposed** : Occurs when the component is disposed by a call to the Dispose() method.
  - **HelpRequest** : Occurs when the user clicks the Help button on a common dialog box.

Print Dialog - Example

```
privatevoid PrintButton_Click(object sender, EventArgs e)
 {
    PrintDialog printDlg = newPrintDialog();
    PrintDocument printDoc = newPrintDocument();
    printDoc.DocumentName = "Print Document";
    printDlg.Document = printDoc;
    printDlg.AllowSelection = true;
    printDlg.AllowSomePages = true;
    //Call ShowDialog
    if (printDlg.ShowDialog() == DialogResult.OK)
                printDoc.Print();
}
```

12.Tree View

- Displays a hierarchical collection of labeled items, each represented by a TreeNode.
- It provides a way to display information in a hierarchical structure by using collapsible nodes
- The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes.

Tree View

- Displays a hierarchical collection of labeled items, each represented by a TreeNode.
- The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes.
- **Namespace:** System.Windows.Forms
- Tree nodes can be expanded to display the next level of child tree nodes. The user can expand the TreeNode by clicking the plus-sign (+) button.
- Can be collapsed to the child TreeNode level by calling the TreeNode.Collapse method, or press the minus-sign (-) button

Tree View- Example

```
// Populates a TreeView control with example nodes.
private void InitializeTreeView()
{
    treeView1.BeginUpdate();
    treeView1.Nodes.Add("Parent");
    treeView1.Nodes[0].Nodes.Add("Unit 1");
    treeView1.Nodes[0].Nodes.Add("Unit 2");
    treeView1.Nodes[0].Nodes[1].Nodes.Add("Common Controls");
    treeView1.Nodes[0].Nodes[1].Nodes[0].Nodes.Add("Dialog box
     controls");
    treeView1.EndUpdate();
}
```

13. Menu

- A menu on a form is created with a MainMenu object, which is a collection of MenuItem objects.
- **Namespace:** System.Windows.Forms
- The **MenuItems property** contains a list of MenuItem objects stored in the menu class.
- Menu class provides **methods, CloneMenu and MergeMenu**, that enables to create new menus from existing menus, and also merge two menu structures together.

Menu

- **Menu Control :Propeties**

  CanRaiseEvents, Container, DesignMode, Events, Handle, IsParent, MdiListItem, MenuItems, Name, Site,Tag

- **Menu Control :Methods**

  CloneMenu(Menu), CreateMenuHandle(), Dispose(), CreateObjRef(Type), Equals(Object), Dispose(Boolean), FindMenuItem(Int32, IntPtr), FindMergePosition(Int32), GetContextMenu(), GetHashCode(), GetLifetimeService(), GetMainMenu(), GetService(Type), MemberwiseClone(), GetType(), InitializeLifetimeService(), MergeMenu(Menu), MemberwiseClone(Boolean), ProcessCmdKey(Message, Keys), ToString()

References

- **http://msdn.microsoft.com**
- https://www.guru99.com/c-sharp-windows-forms-application.html
- http://www.csharphelp.com/
- http://www.csharp-station.com/
- http://www.csharpindex.com/
- http://www.c-sharpcorner.com/
- https://www.w3schools.com/cs/
- https://www.javatpoint.com/c-sharp-tutorial
- http://csharp.net-informations.com/gui/cs-treeview.htm
  - https://www.codeproject.com/Articles/1465/Beginning-C-Chapter-13-Using-Windows-Form-Controls
  - https://ecomputernotes.com/csharp/windows-application/windows-forms-controls