Sub:	Programming Using C# and .Net						
Date:	16/11/2019	Duration:	90 min's	Max Marks:	50	**Sem**	5th

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

1 Illustrate the working of checkBox, TextBox and GrouupBox controls with windows form application example.

❖ **TextBox Control**

➢ The TextBox control is one of the most used controls form from the basic tools. A TextBox control accepts user input on a Form. Windows Forms text boxes are used to get input from the user or to display text. The TextBox control is generally used for editable text, although it can be made read-only. TextBoxes can display multiple lines also add basic formatting.

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
{
   if (e.KeyCode == Keys.Enter)
   {
      MessageBox.Show("You pressed enter! Good job!");
   }
   else if (e.KeyCode == Keys.Escape)
   {
      MessageBox.Show("You pressed escape! What's wrong?");
   }
}
private void Form1_Load(object sender, EventArgs e)
{
   textBox2.Text = "Apple";
}
private void textBox2_TextChanged(object sender, EventArgs e)
{
   if (textBox2.Text.Length == 1)
   {
      if (textBox2.Text == "B" || textBox2.Text == "b")
      {
         textBox2.Text = "Ball";
      }
   }
}
```

❖ **CheckBox Control**

➢ A CheckBox control indicates whether a particular condition is on or off. You can also use check box controls in groups to display multiple choices from which the user can select one or more.

➢ The Check Box control is similar to the radio button control in that each is used to indicate a selection that is made by user. They differ in that only one radio button in a group can be selected at a time but you can make multiple selections using CheckBox.

```
private void checkBox1_CheckedChanged(object sender, EventArgs          e)
      {
         if (checkBox1.Checked == true)
```

```
            MessageBox.Show("Sports Checked", "checkbox");
        else if (checkBox1.Checked == false)
            MessageBox.Show("Sports Unchecked", "checkbox");
    }

    private void checkBox2_CheckedChanged(object sender, EventArgs  e)
    {
        if (checkBox2.Checked == true)
            MessageBox.Show("Dance Checked", "checkbox");
        else if (checkBox2.Checked == false)
            MessageBox.Show("Dance Unchecked", "checkbox");
    }
}
```

- **GroupBox**
- A GroupBox control is a container control that is used to place Windows Forms child controls in a group. The purpose of a GroupBox is to define user interfaces where we can categories related controls in a group. They allow you to place controls on it and all the control inherit the properties from the container in which they sit. This is useful when you to form a group of controls that performs some action or collect logically similar information from the user.

```
private void InitializeMyGroupBox()
{
   GroupBox groupBox1 = new GroupBox();
   Button button1 = new Button();
   button1.Location = new Point(20,10);
   groupBox1.FlatStyle = FlatStyle.Flat;
   groupBox1.Controls.Add(button1);
   Controls.Add(groupBox1);

   GroupBox groupBox2 = new GroupBox();
   Button button2 = new Button();
   button2.Location = new Point(20, 10);
   groupBox2.Location = new Point(0, 120);
   groupBox2.FlatStyle = FlatStyle.Standard;

   // Add the Button to the GroupBox.
   groupBox2.Controls.Add(button2);

   // Add the GroupBox to the Form.
   Controls.Add(groupBox2);
}
```

2 Write short notes on MDI window forms.

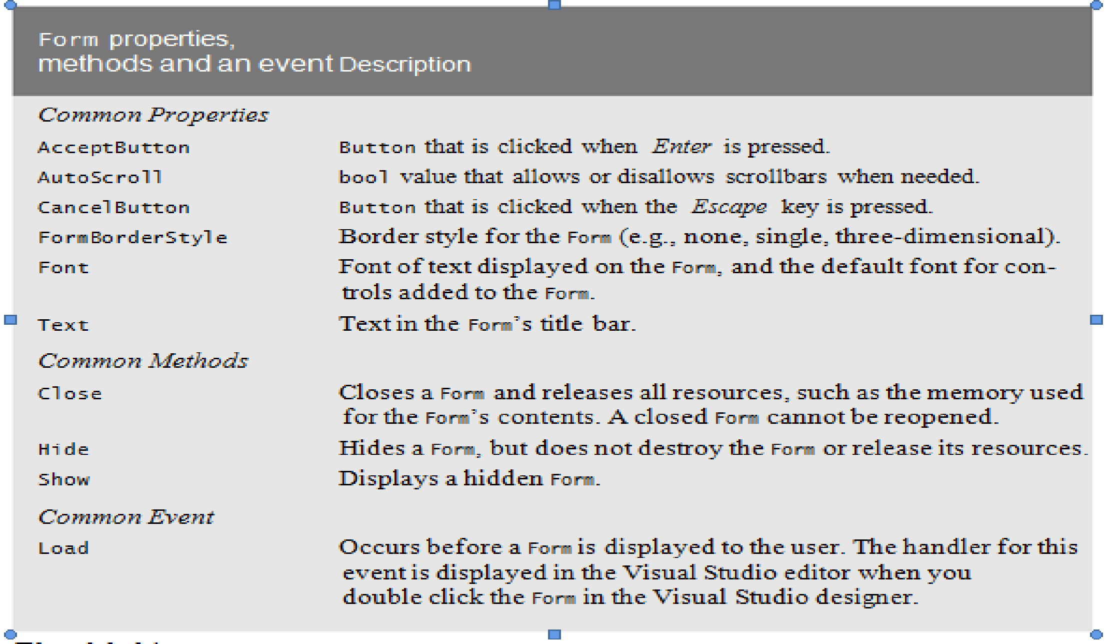
Windows Forms

Windows Forms are used to create the GUIs for programs. A `Form` is a graphical element that appears on your computer’s desktop; it can be a dialog, a window or an **MDI window (multiple**

document interface window).

A component is an in-stance of a class that implements the **IComponent interface**, which defines the behaviors that components must implement, such as how the component is loaded.

A control, such as a Button or Label, has a graphical representation at runtime. Some components lack graphical representations . Such components are not visible at run time.



Form properties, methods and an event	Description
Common Properties	
AcceptButton	Button that is clicked when *Enter* is pressed.
AutoScroll	bool value that allows or disallows scrollbars when needed.
CancelButton	Button that is clicked when the *Escape* key is pressed.
FormBorderStyle	Border style for the Form (e.g., none, single, three-dimensional).
Font	Font of text displayed on the Form, and the default font for controls added to the Form.
Text	Text in the Form's title bar.
Common Methods	
Close	Closes a Form and releases all resources, such as the memory used for the Form's contents. A closed Form cannot be reopened.
Hide	Hides a Form, but does not destroy the Form or release its resources.
Show	Displays a hidden Form.
Common Event	
Load	Occurs before a Form is displayed to the user. The handler for this event is displayed in the Visual Studio editor when you double click the Form in the Visual Studio designer.

Fig. 14.4 | Common Form properties, methods and an event.

3 What is GUI? List and Explain basic controls of GUI.

1. GUI and Basic Controls:

A graphical user interface (GUI) allows a user to interact visually with a program gives a program a distinctive "look" and "feel".

GUIs are built from GUI controls which are sometimes called **components or widgets (window gadgets).** GUI controls are ***objects*** that can display information on the screen or enable users to interact with an application via the *mouse*, *keyboard* or some other form of input.

Several common GUI controls are listed in below table:

Table 1.1: Some basic GUI Controls

Control	Description
Label	Displays images or uneditable text.
TextBox	Enables the user to enter data via the keyboard. It can also be used to display editable or uneditable text.
Button	Triggers an event when clicked with the mouse.

CheckBox	Specifies an option that can be selected (checked) or unselected (not checked).
ComboBox	Provides a drop-down list of items from which the user can make a selection either by clicking an item in the list or by typing in a box.
ListBox	Provides a list of items from which the user can make a selection by clicking one or more items.
Panel	A container in which controls can be placed and organized.
NumericUpDown	Enables the user to select from a range of numeric input values.

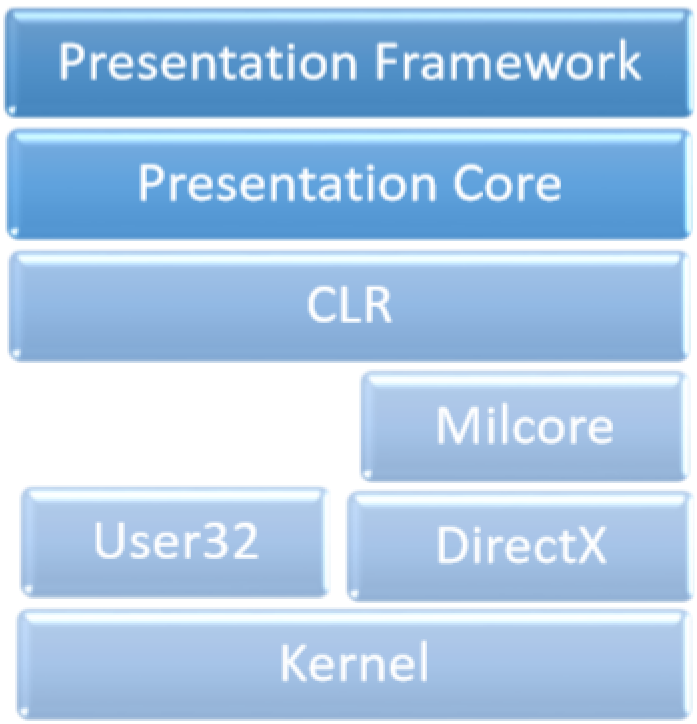
4 Explain Architecture of WPF controls.

Before WPF, the other user interface frameworks offered by Microsoft such as MFC and Windows forms, were just wrappers around User32 and GDI32 DLLs, but WPF makes only minimal use of User32. So,

- WPF is more than just a wrapper.
- It is a part of the .NET framework.
- It contains a mixture of managed and unmanaged code.

The major components of WPF architecture are as shown in the figure below. The most important code part of WPF are –

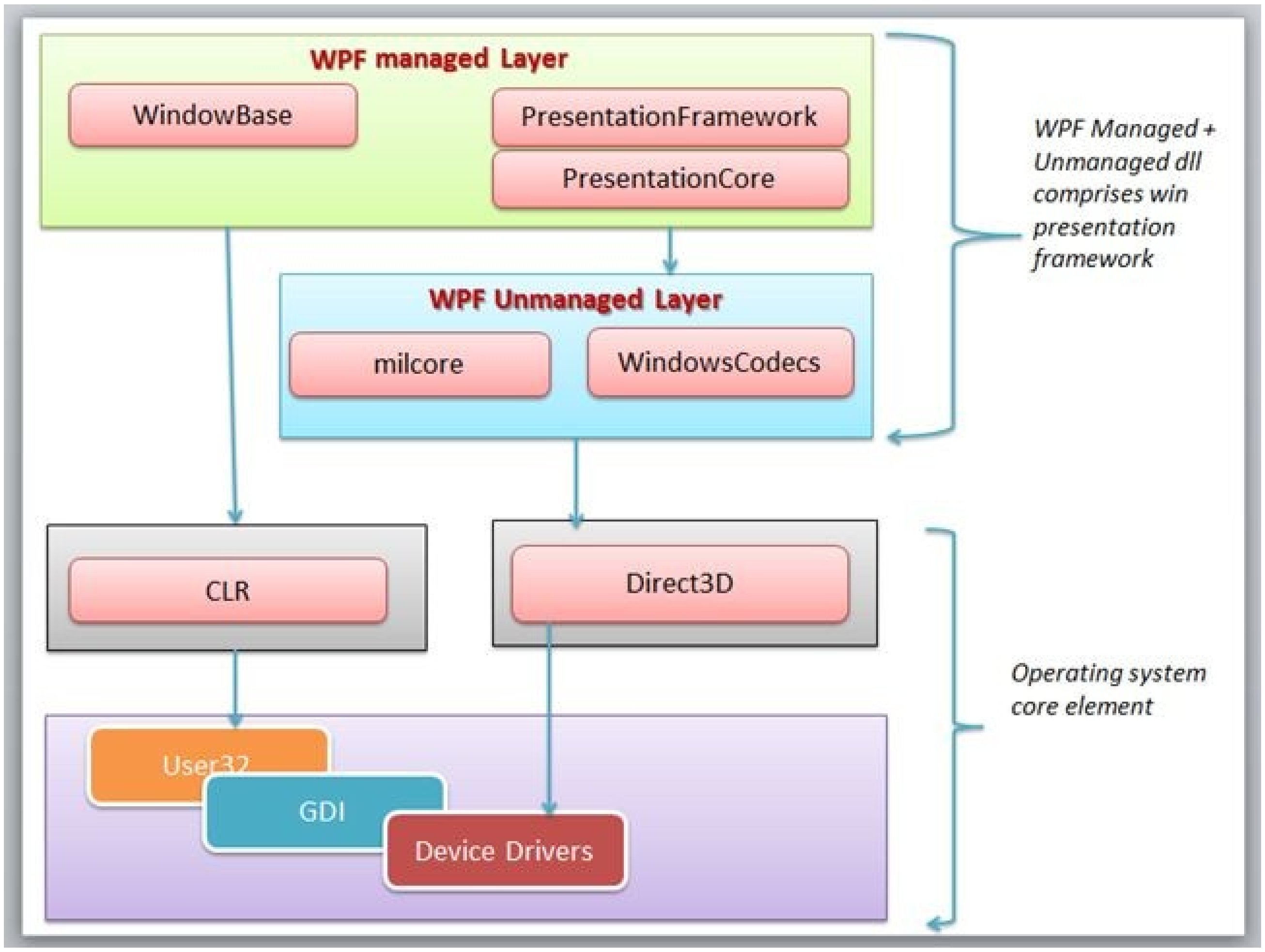
- Presentation Framework
- Presentation Core
- Milcore



The presentation framework and the presentation core have been written in managed code. Milcore is a part of unmanaged code which allows tight integration with DirectX (responsible for display and rendering). CLR makes the development process more productive by offering many features such as memory management, error handling, etc.

The architecture of WPF is actually a multilayered architecture. It has mainly three layers, the WPF Managed Layer, the WPF Unmanaged Layer and the Core operating system element, basically these layers are a set of assemblies that rovide the entire framework.

The major components of WPF are illustrated in the following WPF diagram:



Now I will explain each layer one by one.

Managed Layer

The Presentation Framework, Presentation Core and Window Base are the three major components of the Managed Layer. These are the major code portions of WPF and plays a vital role in an overview of Windows Presentation Foundation. The public API exposed is only via this layer. The Major portion of the WPF is in managed code.

- PresentationFramework.dll: This section contains high-level features like application windows, panels, styles controls, layouts, content and so on that helps us to build our application. It also implements the end-user presentation features including data binding, time-dependencies, animations and many more.

- PresentationCore.dll: This is a low-level API exposed by WPF providing features for 2D, 3D, geometry and so on. The Presentation Core provides a managed wrapper for MIL and implements the core services for WPF such as UI Element and visual . The Visual System creates visual tree that contains applications Visual Elements and rendering instructions.

- WindowsBase.dll: It holds the more basic elements that are capable to be reused outside the WPF environment like Dispatcher objects and Dependency objects.

UnManaged Layer

- milCore.dll: The composition engine that renders the WPF application is a native component. It is called the Media Integration Layer (MIL) and resides in milCore.dll. The purpose of the milCore is to interface directly with DirectX and provide basic support for 2D and 3D surface. This section is unmanaged code because it acts as a bridge between WPF managed and the DirectX / User32 unmanaged API.

- WindowsCodecs.dll: WindowsCodecs is another low-level API for imaging support in WPF applications like image processing, image displaying and scaling and so on. It consists of a number of codecs that encode/decode images into vector graphics that would be rendered into a WPF screen.

Core operating System Layer (Kernel)

This layer has OS core components like User32, GDI, Device Drivers, Graphic cards and so on. These components are used by the application to access low-level APIs.

- DirectX: DirectX is the low-level API through which WPF renders all graphics. DirectX talks with drivers and renders the content.

- User32: User32 actually manages memory and process separation. It is the primary core API that every application uses. User32 decides which element will be placed where on the screen.

- GDI: GDI stands for Graphic Device Interface. GDI provides an expanded set of graphics primitives and a number of improvements in rendering quality.

- CLR: WPF leverages the full .NET Framework and executes on the Common Language Runtime (CLR).

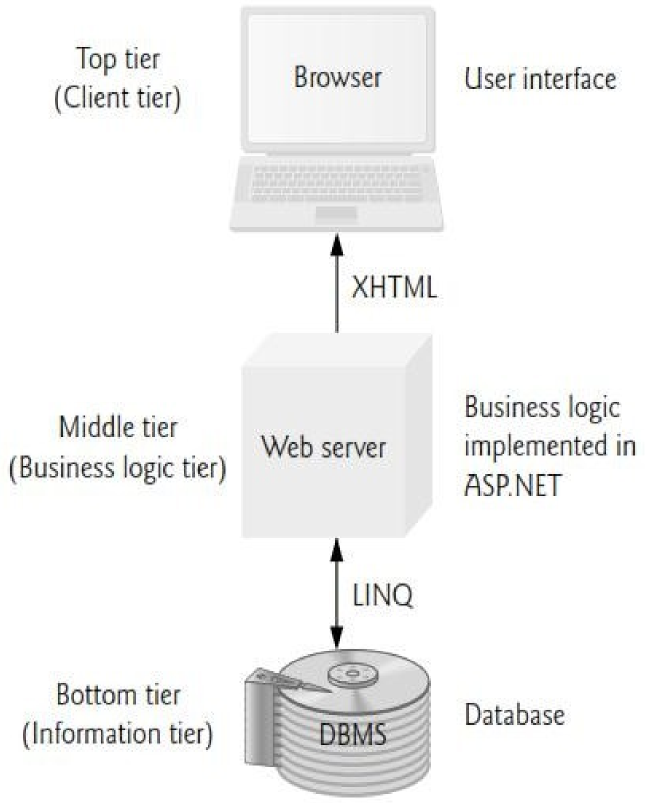
- Device Drivers: Device Drivers are specific to the operating system. Device Drivers are used by the applications to access low-level APIs.

5 Discuss in detail about multi-tier application architecture.

Web-based applications are multitier applications and also referred as n-tier applications.

- Multitier applications divide functionality into separate tiers (that is, logical groupings of functionality).
- Tiers can be located on the same computer, the tiers of web-based applications commonly reside on separate computers for security and scalability.

There are **3** tiers. They are:



i. Information Tier:

- The information tier also called the bottom
- It maintains the application's data.
- This tier typically stores data in a relational database management system.

Example: A retail store might have a database for storing product information, such as descriptions, prices and quantities in stock.
The same database also might contain customer information, such as user names, billing addresses and credit card numbers.

- This tier can contain multiple databases, which together comprise the data needed for an application.

ii.Business Logic:

- The middle tier acts as an intermediary between data in the information tier and the application's clients.
- The middle-tier **controller logic** processes client requests (such as requests to view a product catalog) and retrieves data from the database.
- The middle-tier **presentation logic** then processes data from the information tier and presents the content to the client.
- Web applications typically present data to clients as web pages.
- Business logic in the middle tier enforces business rules and ensures that data is reliable before the server application updates the database or presents the data to users.
- Business rules dictate how clients can and cannot access application data, and how applications

process data.

Example: A business rule in the middle tier of a retail store's web-based application might ensure that all product quantities remain positive.

A client request to set a negative quantity in the bottom tier's product information database would be rejected by the middle tier's business logic.

ii. Client Tier:

- The client tier, or top tier, is the application's user interface, which gathers input and displays output.
- Users interact directly with the application through the user interface (typically viewed in a web browser), keyboard and mouse.
- In response to user actions (Example: clicking a hyperlink), the client tier interacts with the middle tier to make requests and to retrieve data from the information tier.
- The client tier then displays to the user the data retrieved from the middle tier.
- The client tier never directly interacts with the information tier.

6 Explain the session management in ASP.NET using controls.

Cookies: Cookies provide you with a tool for personalizing web pages. A **cookie** is a piece of data stored by web browsers in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions.

Session tracking using the .NET class **HttpSessionState:**
If the user clicks the link for book recommendations, the information stored in the user's unique **HttpSessionState** object is read and used to form the list of recommendations. That can be done using **Session property.**

Session Property:

Every Web Form includes a user-specific **HttpSessionState** object, which is accessible through property **Session**
of class Page. We use this property to manipulate the current user's **HttpSessionState** object.

When a page is first requested, a unique HttpSessionState object is created by ASP.NET and assigned to the Page's **Session** property.

The session object is created from the HttpSessionState class, which defines a collection of session state

items. The **HttpSessionState class** has the following **properties**:

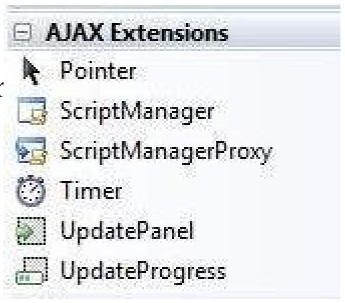
Properties	Description
SessionID	The unique session identifier.
Item(name)	The value of the session state item with the specified name. This is the default property of the HttpSessionState class.
Count	The number of items in the session state collection.
TimeOut	Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session.

The **HttpSessionState** class has the following **methods**:

Methods	Description
Add(name, value)	Adds an item to the session state collection.
Clear	Removes all the items from session state collection.
Remove(name)	Removes the specified item from the session state collection.
RemoveAll	Removes all keys and values from the session-state collection.
RemoveAt	Deletes an item at a specified index from the session-state collection.

7 Explain the controls from AJAX control toolkit.

1. AJAX Control ToolKit:



The control toolbox in the Visual Studio IDE contains a group of contr

called the “AJAX Extensions”.

The ScriptManager Control:

The ScriptManager control is the most important control and must be present on the page for other controls to work.

Syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control.

The **ScriptManager** control takes care of the client-side script for all the server side controls.

The UpdatePanel Control:

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface.

When a control inside it triggers a post back, the UpdatePanel interferes to initiate the post asynchronously and update just that portion of the page.

Example:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
                <asp:Button ID="btnpartial" runat="server"
                        onclick="btnpartial_Click" Text="Partial PostBack"/>
                <br />
                <br />
                <asp:Label ID="lblpartial" runat="server"></asp:Label>
         </ContentTemplate>
</asp:UpdatePanel>
```

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

UpdateMode	ChildrenAsTriggers	Effect

Always	False	Illegal parameters.
Always	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back.
Conditional	False	UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.
Conditional	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

The UpdateProgress Control:

The **UpdateProgress** control provides a sort of feedback on the browser while one or more update panel controls are being updated.

Example: While a user logs in or waits for server response while performing some database oriented job. It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

Syntax:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
         DynamicLayout="true" AssociatedUpdatePanelID="UpdatePanel1" >

      <ProgressTemplate>
         Loading...
      </ProgressTemplate>

</asp:UpdateProgress>
```

The above snippet shows a simple message within the **ProgressTemplate** tag.

However, it could be an image or other relevant controls. The **UpdateProgress** control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

Properties	Description
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.
Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.
DynamicLayout	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

Methods	Description
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

i. Setting the **Triggers** property of the **UpdatePanel control:**

ii. Placing a **timercontrol** directly inside the **UpdatePanel** to act as a **child control trigger. A single timer can be the trigger for multiple UpdatePanels.**

8 Describe the different validation controls With Example in ASP.NET.

Validation control or **Validator**, which determines whether the data in another web control is in the proper format.

- **Validators** provide a mechanism for validating user input on the client.
- When the page is sent to the client, the validator is *converted into JavaScript* that performs the validation in the client web browser.
- JavaScript is a scripting language that executed on the client. Unfortunately, some client browsers might not support scripting or the user might disable it.

For this reason, you should ***always perform validation*** on the **server**. ***ASP.NET validation controls*** can function on the **client**, on the **server** or both.

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET that listed below:

The below table describes the controls and their work:

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern

CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

All these validation control classes are inherited from the **BaseValidator** class hence they inherit its properties and methods that are ControlToValidate, Display, EnableClientScript, Enabled, Text, isValid, and validate() method.

RequiredFieldValidator Control:

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

Syntax:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
   runat="server" ControlToValidate ="ddlcandidate"
   ErrorMessage="Please choose a candidate"
   InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

RangeValidator Control:

The RangeValidator control verifies that the input value falls within a predetermined range. It has **three** specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

Syntax:

```
<asp:RangeValidator ID="rvclass" runat="server"
     ControlToValidate="txtclass"
     ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
     MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

CompareValidator Control:

The CompareValidator control compares a value in one control with a fixed value or a value in another control. It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
        ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

Syntax:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
        ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

RegularExpressionValidator:

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for

Regular expressions: \b, \n, \,\f, \r, \v, \t

Metacharacters: ., [abcd], [^abcd], [a-zA-Z], \w, \W, \s, \S, \d, \D

Quantifiers: *, +, ?, {n}, {n, },

{n,m}

```
<asp:RegularExpressionValidator ID="string" runat="server"
        ErrorMessage="string" ValidationExpression="string"
        ValidationGroup="string">
</asp:RegularExpressionValidator>
```

CustomValidator:

The **CustomValidator** control allows writing application specific custom validation routines for both the client side and the server side validation.

- The **client side validation** is accomplished through the **ClientValidationFunction property**. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.
- The **server side validation** routine must be called from the control's **ServerValidate** eventhandler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

Syntax:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
        ClientValidationFunction=.cvf_func.
        ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

ValidationSummary:

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation. The following **two mutually inclusive properties** list out the error message:

- ShowSummary : shows the error messages in specified format.

ShowMessageBox : shows the error messages in a separate window.

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode = "BulletList" ShowSummary = "true"
    HeaderText="Errors:" />
```

9Wriwrite a program for addition AND Subtraction of two numbers from text boxes and display the result on label in the form using button click event.

```
 TextBox TextBox1= new TextBox();
TextBox TextBox2= new TextBox();
Label  Label1= new Label();
Label  Label2= new Label();
Label  Label3= new Label();

Button Button1= new Button();
this.controls.add(TextBox1);
this.controls.add(TextBox2);
this.controls.add(Label1);
this.controls.add(Label2);
this.controls.add(Label3);


Int a=Convert.ToInt32(TextBox1.Text);
Int b=Convert.ToInt32(TextBox2.Text);

Int sum=a+b;
Int sub=a-b;
Int mul=a*b;
Label1.Text=Convert.ToString(sum);
Label2.Text=Convert.ToString(sub);

Label3.Text=Convert.ToString(mul);
```

10 Explain how custom Exceptions will be ceated in C# program with suitable example.

The **ApplicationException** is thrown by a user program, not by the common language runtime. If you are designing an application that needs to create its **own exceptions**.

To create your own exception class, here are some important recommendations:

- Give a meaningful name to your Exception class and end it with Exception.
- Throw the most specific exception possible.
- Give meaningful messages.
- Do use InnerExceptions.
- When wrong arguments are passed, throw an ArgumentException or a subclass of it, if necessary.

Building Custom Exceptions, Take Two:

Set the parent's Message property via an **incoming constructor parameter**. Here, you are simply *passing the parameter* to the **base class constructor.**

With this design, a custom exception class is a uniquely named class deriving from

System.ApplicationException.

```
class MyException : ApplicationException
      { public MyException(){ . . . }
      public MyException(String Msg): base(Msg){. . .
      }
```

```
using System;
namespace Chapter5_Examples {
  class MyException : ApplicationException { public
    MyException(String Msg): base(Msg){
      Console.WriteLine("Pass the message up to the base class");
    }
  }
  class CusException1{
    static void Main(string[] args){ try{
         throw new MyException("This is My Exception");
      }
      catch(MyException ex){ Console.WriteLine(ex.Message);  }
      Console.ReadLine();
    }
  }
}
```

Building Custom Exceptions, Take Three:

To build a correct and proper custom exception class, which requires to follow the below points:

- Derives from Exception /ApplicationException

- Is marked with the [System.Serializable] attribute
- Defines a default constructor
- Defines a constructor that sets the inherited Message property
- Defines a constructor to handle “InnerExceptions” and the serialization of your type

However, to finalize our examination of building custom exceptions, here is the final iteration of **CustomException:**

```
[Serializable]
public class CustomException : Exception{
    public CustomException() : base() { }
    public CustomException(string message): base(message) { }

    public CustomException(string format, params object[] args)
        : base(string.Format(format, args)) { }

    public CustomException(string message, Exception
             innerException): base(message, innerException) { }

    public CustomException(string format, Exception

                          innerException, params object[] args)
        : base(string.Format(format, args), innerException) { }

    protected CustomException(SerializationInfo info,
           StreamingContext context) : base(info, context) { }
}
```

1. Throw exception without message: `throw new CustomException()`
2. Throw exception with simple message: `throw new CustomException(message)`
3. Throw exception with message format and parameters
   ```
   throw new CustomException("Exception with parameter
                                  value '{0}'", param)
   ```
4. Throw exception with simple message and inner exception
   ```
   throw new CustomException(message, innerException)
   ```
5. Throw exception with message format and inner exception. Note that, the variable length params are always floating.
   ```
   throw new CustomException("Exception with parameter
                  value '{0}'", innerException, param)
   ```
6. The last flavor of custom exception constructor is used during exception serialization/deserialization.