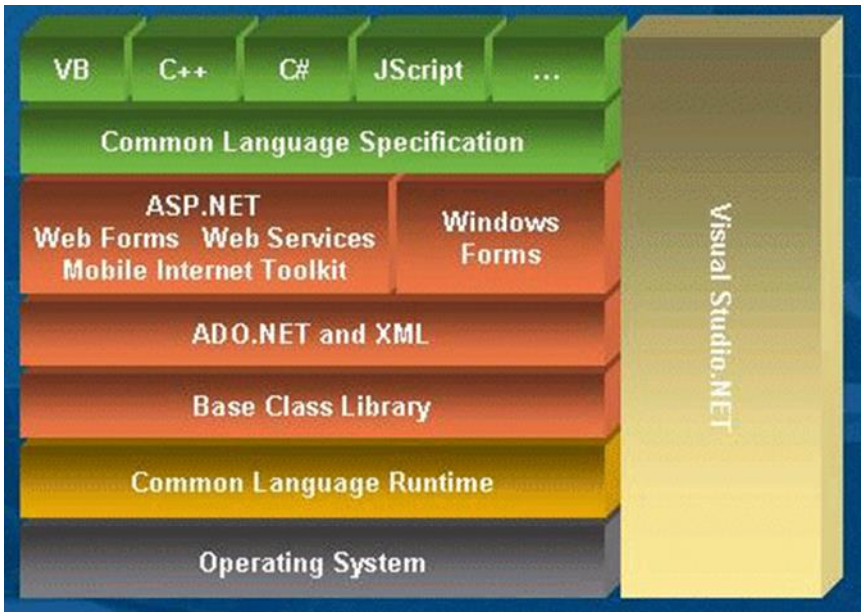
.Net Introduction

Explain .Net Framework Architecture

Definition: A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services.

- The .NET is an application development platform or framework.
- It is a Framework that supports Multiple Language and Cross language integration.
- IT has IDE (Integrated Development Environment).
- Developers can easily develop windows applications, web applications, web services, mobile applications, etc… using .NET framework.
- .NET framework SDK is free and includes command-line compilers for C++, C#, and VB.NET and various other utilities for development. Whereas most popular tool for .NET application development is visual studio and it costs few hundred dollars.
- First version of .NET was 1.0 and released in 2002 whereas latest version available in market is 4.5, released in 2012



Web Application & Windows Applications:

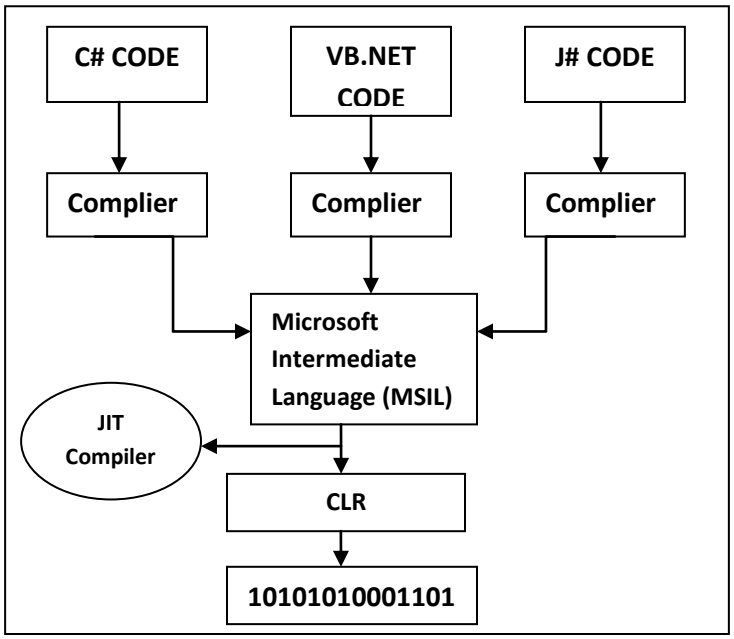
- ASP.NET is being used by .NET Framework to develop web applications & web services.
- With the help of ASP.NET we can develop more powerful and attractive web applications compare to ASP.
- We can develop web application by using the JQuery, JavaScript and Ajax.
- With the help of Windows Applications we can develop the different kind of software and ERP.

- With the help of .NET Framework we can do the Rapid Application Development (RAD) by using the Web forms and Windows Forms.
- We have to just drag and drop the controls on to the form, write the code related to the particular events, which is going to be performed when that particular event is executed.
- With the help of web services we can develop the light weighted distributed components.

ADO.NET and XML:

- ADO.NET extends the base classes and managing the data to the database and maintains the XML data manipulations.
- These all classes support database management with the help of Structured Query Language (SQL) and it also allows you to change the data.
- The .Net framework also supports the classes which will help you to maintain the XML data and perform searching and translations.

Common Language Runtime:



- The heart of the .Net framework is the Common Language Runtime (CLR).
- CLR manages the execution of the whole application.
- In the CLR, code is expressed as the byte code called the **MSIL** code (**MSIL = Microsoft Intermediate Language**).
- Developers using the CLR, write code in a language such as **C#** or **VB.NET.** At Compile time, a .NET Compiler converts such code into MSIL code.
- At runtime MSIL's just-in-time compiler converts the MSIL code to the Native code.
- JIT compiler converts the MSIL code in to the Native Code, which is related to the particular systems.
- Rather than converts all the code to the native code JIT compiler will converts the required amount of code to the Native code and saves the time and memory both.

Explain terms CLS, CTS, Jitter, IL, Namespace

CLS:

- A **Common Language Specification** (**CLS**) is a document that says how computer programs can be turned into bytecode.
- When several languages use the same bytecode, different parts of a program can be written in different languages. Microsoft uses a Common Language Specification for their .NET Framework.
- Common Language Specification (CLS) is a set of basic language features that .Net Languages needed to develop Applications and Services, which are compatible with the .Net Framework.
- When there is a situation to communicate Objects written in different .Net languages, those objects must expose the features that are common to all the languages.
- Common Language Specification (CLS) ensures complete interoperability among applications, regardless of the language used to create the application.
- Common Language Specification (CLS) defines a subset of Common Type System (CTS).
- Common Type System (CTS) describes a set of types that can use different .Net languages have in common, which ensure that objects written in different languages can interact with each other.

CTS (Common Type System):

- As .Net framework is language independent and support over 20 different programming languages, many programming language will write data types in their own programming languages.
- For example, an integer variable in C# is written as Int, where as in visual basic it is written as integer.
- Therefore in .Net framework you have single class called System.Int32 to interpret these variables. Similarly, for the ArrayList data type .Net framework has a common type called System.Collections.ArrayList.
- In .Net framework, System.Object is the common base type from where all the other types are derived.
- In Microsoft's .Net framework, the Common Type System(CTS) is a standard that specifies how type definitions and specific values of types are represented in computer memory.
- It is intended to allow programs written in different programming languages to easily share information.
- This system is called Common Type System, the types in .Net framework are the base on which .Net applications, component, and controls are built.
- Common type system in .Net framework defines how data types are going to be declared and managed in runtime.
- Purpose of CTS;
    1. Provides an object oriented mode that supports the complete implementation of many programming languages.
    2. Defines rules that languages must follow which helps ensure that objects written in different languages can interact with each other.
    3. For example Object1 is created in VB.Net and Object2 is created in C#.Net, so far cooperating for two objects.
    4. Languages supported by .Net can implement all or some common data types.

JITTER:

- There are three different Jitters can be used to convert the MSIL code into native code,
    1. Pre-JIT: Pre – JIT will compile an entire assembly into CPU-specific binary code. This compilation is done at install time, when the end user is least likely to notice that the assembly is being JIT-compiled.
    2. Econo-JIT: Econo – JIT compiles only those methods that are called at runtime. However these compiled methods are removed if the system begins to run out of memory.
    3. Normal-JIT: This Jitter is called at run time each time a method is invoked for the first time.

IL (Intermediate Language):

- MSIL or IL (Intermediate Language) is machine independent code generated by .NET framework after the compilation of program written in any language by you (as shown in the above figure).

Namespace

- A Namespace in Microsoft .Net is like containers of objects. They may contain unions, classes, structures, interfaces, enumerators and delegates.
- Main goal of using namespace in .Net is for creating a hierarchical organization of program. In this case a developer does not need to worry about the naming conflicts of classes, functions, variables etc., inside a project.
- In Microsoft .Net, every program is created with a default namespace. This default namespace is called as global namespace. But the program itself can declare any number of namespaces, each of them with a unique name.
- The advantage is that every namespace can contain any number of classes, functions, variables and also namespaces etc., whose names are unique only inside the namespace.
- The members with the same name can be created in some other namespace without any compiler complaints from Microsoft .Net.
- To declare namespace C# .Net has a reserved keyword **namespace**. If a new project is created in Visual Studio .NET it automatically adds some global namespaces. These namespaces can be different in different projects. But each of them should be placed under the base namespace System.
- The names space must be added and used through the **using operator**, if used in a different project. Please now have a look at the example of declaring some namespace:
  Using System;

Explain Managed and Unmanaged Code in .Net Framework

Managed Code:

- The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.
- It compiles to MSIL, not to the machine code that could run directly on your computer.

- The MSIL is kept in a file called an assembly, along with metadata that describes the classes, methods, and attributes of the code you've created.
- Now if we want to use that assembly on another pc then we have to just copy that assembly and used it.
- Managed code uses CLR which in turns looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on.

Unmanaged Code:

- The code, which is developed outside .NET, Framework is known as unmanaged code.
- Applications that do not run under the control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write.
- Unmanaged code can be unmanaged source code and unmanaged compile code.

Explain Assembly Structure. Write code for create shared assembly and store the assembly in GAC using tool.

- Assemblies are the basic building blocks required for any application to function in the .NET realm. They are partially compiled code libraries that form the fundamental unit of deployment, versioning, activation scoping, reuse, and security
- Typically, assemblies provide a collection of types and resources that work together to form a logical unit of functionality. They are the smallest deployable units of code in .NET. Compared to the executable files assemblies are far more reliable, more secure, and easy to manage.
- An assembly contains a lot more than the Microsoft Intermediate Language (MSIL) code that is compiled and run by the Common Language Runtime (CLR).
- Assembly contains four major parts
    1. Manifest- Additional information regarding assembly
    2. Type metadata:-Data about data or structure of classes
    3. MSIL code: - Containing business logics – Intermediate version of our program
    4. Set of Resource:- Information or resource of other assembly

Static Assembly & Dynamic Assembly

- Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on).
- Static assemblies are first stored on disk in portable executable (PE) files then run.
- Dynamic assemblies run directly from memory and are not saved to disk before execution.
- You can save dynamic assemblies to disk after they have executed.

Single File Assembly & Multi File Assembly

- A single file assembly contains all the required information (IL, Metadata, and Manifest) in a single package.
- A single file assembly does not implement version checking.

- A multi file assembly is spread in multiple files.
- A Multi file assembly is generally used for large applications with various types of resources
- A Multi file assembly implements version checking.

Private Assembly & Shared Assembly

- A Private assembly is used only by a single application.
- A Private assembly is stored in the application's directory or sub-directory.
- A shared assembly is normally used by more than one application.
- A shared assembly is stored in the global assembly cache (GAC) which is a repository of assemblies maintained by .NET runtime.
- The .NET Framework library is provided as shared assemblies.

A shared assembly is an assembly that resides in a centralized location known as the GAC (Global Assembly Cache) and that provides resources to multiple applications. If an assembly is shared then multiple copies will not be created even when used by multiple applications.

The GAC folder is under the Windows folder.
<drive>:\windows\assembly<--GACfolder
We can find all base class libraries under GAC.
We can only get a strong named assembly into the GAC.
GAC contains multiple assemblies and it is identified by PUBLIC KEY TOKEN.

How to generate a public key token:

We have a tool named strong name utility to do this, which is a command line tool and should be used from a command prompt as following.
Sn -k <file name>
E.g.:<drive>:\<folder>> sn -k key.snk
The above atatement generates a key value and writes it into "key.snk".
We can use sn or snk for the extension of key files.

Creating A shared assembly

- **Step 1:Generate a key file.** Open a VS command prompt. Go into your folder and generate a key file as: <drive>:\<folder> sn -k key.snk
- **Step 2:** create a project and associate a key file to it before compilation so that the generated assembly will be strong named.
  Open a new project of type class library and name it sAssembly; under class1 write the following:

```
Public string sayhello()
{
        Return "hello from shared assembly";
}
```

  To associate a key file we generated with the project, open the project properties and select the "signing" tab on the LHS which displays a CheckBox as "sign the assembly" select it that displays ComboBox below it

from it select browse and select key.snk from its physical location then compile the project using build which will generate assembly Assembly.dll that is strong named.

- **Step 3:copying the assembly into GAC**
  .Net provides a command line utility to be used as shown in the following:
  Gacutil -I | -u <assembly name> I:install u:uninstall
  Open a VS command prompt; go to the location where the Assembly.dll is present and write the following:
  <drive>:\<folder>\sAssembly\ sAssembly\\bin\Debug>gacutil -I Assembly.dll
- **Step 4:Testing**
  Open a new project add a reference to Assembly.dll and write the following code for the button click event.
  sAssembly.Classs1 obj=new sAssembly.Class1();
  Messagebox.Show(obj.sayhello());
  Run the project and verify under the bin/debug folder of the current project where we will not find a copy of the Assembly.dll as it is a shared assembly.

Explain the concepts of Garbage Collection

- Garbage collection is automatic memory manager for the .Net framework.
- When we have a class that represents an object in the runtime that allocates a memory space in the heap memory.
- All the behavior of that objects can be done in the allotted memory in the heap.
- Once the activities related to that object is get finished then it will be there as unused space in the memory.
- When a variable defined, its gets a space in the memory and when the program control comes out of that function the scope of variable get ended, so the garbage collection acts on and memory will releases.
- Garbage collection prevents memory leaks during execution of programs.
- Garbage Collector is a low-priority process that manages the allocation and deallocation of memory for your application.

Finalization:

- Finalize provides implicit control by implementing the protected finalize method on an object. The garbage collector calls this method at some point after there are no longer any valid references to the object.

Dispose:

- Dispose is used to explicitly release resources before garbage collector frees the object. If programmer wants to explicitly release resources, dispose is used.
- To provide explicit control, implement Dispose method provided by IDisposable interface.
- Programmer calls the Dispose method while Finalize is invoked by the Garbage Collector.

VB.Net

What is VB.Net? Explain? Differentiate VB.Net and C#

- VB.Net is a simple, modern, object-oriented computer programming language developed by Microsoft.
- It is developed to combine the power of .NET Framework and the common language runtime with the productivity benefits that are the hallmark of Visual Basic.
- Everything in VB.NET is an object.

The following reasons make VB.Net a widely used professional language

1. Modern, General Purpose
2. Object Oriented
3. Component Oriented
4. Easy to learn
5. Structured Language
6. It produces efficient programs
7. Part of .Net Framework

Strong Programming Features of the VB.Net

1. Boolean Conditions
2. Automatic Garbage Collection
3. Standard Library
4. Assembly Versioning
5. Properties and Events
6. Delegates and Events Management
7. Easy-to-use Generics
8. Indexers
9. Conditional Compilation
10. Simple Multithreading

Difference between VB.Net and C#

Purpose	VB.NET	C#
Declare a named constant	Const	Const
Create a new object	New	New
Function/Method does not return a value	Sub	Void
Overload a function or method	Overloads	No keyword required for this
Refer to the current object	Me	This
Initialize a variable where it declared	Dim x as Long = 07;	int x=07;
Refer to a base class	MyBase	Base
Declare an interface	Interface	interface

Specify an interface to be implemented	Implements (statement)	class C1 : I1 (where C1 is class and I1 is interface name)
Derive a class from a base class	Inherits C2	Class C1 : C2 (Where C1 and C2 are classes name)
Override a method	Overrides	override
Declare an array	Dim a() as Long	int[] a = new int[5];
Initialize an array	Dim a() as Long = {3,6,9};	int[] a = new int[5]{1,5,8,3,2};
Comment Purpose	‘ sign will be used	// sign will be used

Explain DataTypes in VB.Net, how to declare variables in VB.Net?

Data Type	Storage Allocation	Value Range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point, 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	2 bytes	0 through 65,535 (unsigned)

Declaring variables

```
Dim number1 As Integer
Dim number2 As Integer

number1 = 3
number2 = 5
```

Here's a breakdown of the variable Declaration:

- **Dim:** Short for Dimension. It's a type of variable. You declare (or "tell" Visual Basic) that you are setting up a variable with this word. We'll meet other types of variables later, but for now just remember to start your variable declarations with Dim.
- **number1:** This is the cardboard box and the sticky label all in one. This is a variable. In other words, our storage area. After the Dim word, Visual Basic is looking for the name of your variable. You can call your variable almost anything you like, but there are a few reserved words that VB won't allow. It's good practice to give your variables a name appropriate to what is going in the variable.
- **As Integer:** We're telling Visual Basic that the variable is going to be a number (integer). Well meet alternatives to Integer later.
- **Number1 = 3:** The equals sign is not actually an equals sign. The = sign means assign a value of. In other words, here is where you put something in your variable. We're telling Visual Basic to assign a value of 3 to the variable called number1. Think back to the piece of paper going into the cardboard box. Well, this is the programming equivalent of writing a value on a piece of paper

Explain Program Structure of VB.Net with Sample Program.

- Let us look at the basic level of the VB.Net Program structure.
- A VB.Net Program basically consists of the following parts;
    1. Namespace declaration
    2. A class or module
    3. One to more procedures
    4. Variables
    5. The main procedure
    6. Statements and Expressions
    7. Comments

Sample VB.Net program

```
Imports System
Module Module1
   'This program will display Hello World
   Sub Main()
      Console.WriteLine("Hello World")
      Console.ReadKey()
   End Sub
End Module
```

- The first line of the program **Imports System** is used to include the System namespace in the program.
- The next line has a **Module** declaration, the module *Module1*. VB.Net is completely object oriented, so every program must contain a module of a class that contains the data and procedures that your program uses.
- Classes or Modules generally would contain more than one procedure. Procedures contain the executable code, or in other words, they define the behavior of the class. A procedure could be containing Function, Sub, Operator, Get, Set, AddHandler, Remove Handle, RaiseEvent.
- The next line('This program) will be ignored by the compiler and it has been put to add additional comments in the program.
- The next line defines the Main procedure, which is the entry point for all VB.Net programs. The Main procedure states what the module or class will do when executed.
- The Main procedure specifies its behavior with the statement
- **Console.WriteLine("Hello World")**
- *WriteLine* is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- The last line **Console.ReadKey()** is for the VS.NET Users. This will prevent the screen from running and closing quickly when the program is launched from Visual Studio .NET.

Explain the concept of Class and Object inVB.Net with Example. Also Explain Concept of Inheritance with Example in VB.Net. (Example: Create VB.Net console application to define shape class and derive circle and rectangle from it to demonstrate inheritance)

- A *class* is simply a representation of a type of *object*. It is the blueprint/ plan/ template that describe the details of an *object*.
- Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

- A class definition starts with the keyword **Class** followed by the class name; and the class body, ended by the End Class statement.

```
[accessmodifier] Class [classname ]
      Inherits [class_name]
      Implements [interface_name]
      [statements]
EndClass
```

- **accessmodifier** defines the access levels of the class, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- **Inherits** specifies the base class it is inheriting from.
- **Implements** specifies the interfaces the class is inheriting from.

```
Module Module1
    Class Test                          //Base Class
        Public Sub Circle()
            Console.WriteLine("Circle Method of Test Class")
        End Sub
    End Class
    Class Demo                          //Derived Class
        Inherits Test
        Public Sub rectangle()
            Console.WriteLine("Rectangle Method of Demo Class")
        End Sub
    End Class
    Sub Main()
        Dim d As Demo = New Demo()
        d.Circle()
        d.rectangle()
        Console.ReadKey()
    End Sub
End Module
```

- As shown in the example we had created one class called **"Test"** which contains one method/function, its name is **"Circle"** and we had also created another class called **"Demo"** which contains one method/function, its name is **"rectangle"**.
- One of the most important concepts in object-oriented programming is that of inheritance.
- Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the **base** class(Test Class), and the new class is referred to as the **derived** class(Demo Class).
- In our example Demo class inherits the Test class, now in the main method/function we had created object of the Demo class, now with the help of that object "d" we can called the method of the Test class and Demo class because of the inheritance.

Explain method overloading in VB.Net with Example. (Example: Create VB.Net Console Application to overload area method to find area of circle and rectangle)

- Overloading is basically when you have more than one method with the same name but a different number of or different types of parameters.
- To overload a method in VB.Net we have to declare that method firstly overridable, so after declaring particular method overridable, we can override that method.
- With the help of overloading method, we can define a method with the same name, but different number of arguments, so as we are passing the number of parameters at that time it will be decided which method is going to be called.
- Example of the method overloading is as shown below, in that we had created one class Area_Found, which contains method area which is overridable.

```
Module Module1
    Class Area_Found
        Public Overridable Sub area(ByVal r As Integer)
            Dim area As Double = r * r * 3.14
            Console.WriteLine(area)
        End Sub
        Public Sub area(ByVal l As Integer, ByVal b As Integer)
            Dim area As Double = l * b
            Console.WriteLine(area)
        End Sub
    End Class
    Sub Main()
        Dim obj As Area_Found = New Area_Found()
        obj.area(2)         //It will call the area method of circle
        obj.area(3, 4)      //It will call the area method of rectangle
        Console.ReadKey()
    End Sub
End Module
```

- When we are passing one parameter at that time it will call the area method to find the area of the circle, but when we are passing two parameters value then it will call the area method of the rectangle.

Explain different windows form controls in VB.Net

- VB.Net contains a very rich amount of the controls for the windows application.
- Every controls have three important elements which are Properties and Events
- **Properties** means all the controls can resized, customized and moved. A property is a value or characteristics held by the particular control.

- **An Event means** a signal that informs an application that something important has occurred.
- Some of the common Properties, Method and Events of all controls are as below.

Common Properties of all controls which are listed below:

Property Name	Description
Text	Display the name of the control on the windows forms
Name	Contains the name of the controls by which the control can be identified
Minimum Size	With the help of the this property we can manage the minimum size of the control
Maximum Size	With the help of the this property we can manage the maximum size of the control
Size	With the help of the this property we can manage the size of the control
Modifier	To handle the access level of control public, private, protected or friend
Font	To set the font style of that control
Anchor	To set anchor property of the control
TabIndex	To set the tab index of the particular control
Visible	Control should be visible or not visible we can set with this property

Common Events of all controls which are listed:

Event Name	Description
Click	When we click on the control and we have to execute any code that code will be written in this event
MouseClick	When mouse will be clicked on the control and we have to execute any code that code will be written in this event
DoubleClick	When we double click on the control and we have to execute any code that code will be written in this event
DragLeave	When we remove the mouse from the control and at that time if any code have to be executed that code will be written in this event
SizeChanged	When we changed the size of the control and at that time if any code have to be executed that code will be written in this event
StypeChanged	When we changed the style of the control and at that time if any code have to be executed that code will be written in this event

Some of them controls of windows forms are as following;

1. **Form Control**
   - The container for all the controls that make up the user interface.
   - **Properties**:

Property Name	Description
StartPosition	This property determines the initial position of the form when it's first displayed.
MinimizeBox	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
MaximizeBox	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.

   - **Events**:

Event Name	Description
Closed	Occurs before the form is closed.

Closing	Occurs when the form is closing
HelpButtonClicked	Occurs when the Help button is clicked.

2. Button Control

- It represents a Windows button control.
- **Properties**:

Property Name	Description
BackColor	Gets or sets the background color of the control.
DialogResult	Gets or sets a value that is returned to the parent form when the button is clicked. This is used while creating dialog boxes.
Image	Gets or sets the image that is displayed on a button control.

- **Events**:

Event Name	Description
GotFocus	Occurs when the control receives focus.
TextChanged	Occurs when the Text property value changes.
Validated	Occurs when the control is finished validating.

3. Label Control

- It represents a standard Windows label.
- **Properties**:

Property Name	Description
Autosize	Gets or sets a value specifying if the control should be automatically resized to display all its contents.
BorderStyle	Gets or sets the border style for the control.
TextAlign	Gets or sets the alignment of text in the label.

- **Events**:

Event Name	Description
Leave	Occurs when the input focus leaves the control.
LostFocus	Occurs when the control loses focus.
TextChanged	Occurs when the Text property value changes.

4. TextBox Control

- It represents a Windows text box control.
- **Properties**:

Property Name	Description
CharacterCasing	Gets or sets whether the TextBox control modifies the case of characters as they are typed.
Multiline	Gets or sets a value indicating whether this is a multiline TextBox control.
PasswordChar	Gets or sets the character used to mask characters of a password in a single-line TextBox control.

- **Events**:

Event Name	Description
TextAlignChanged	Occurs when the TextAlign property value changes.

5. RadioButton Control

- It enables the user to select a single option from a group of choices when paired with other RadioButton controls.
- **Properties**:

Property Name	Description
Appearance	Gets or sets a value determining the appearance of the radio button.
CheckAlign	Gets or sets the location of the check box portion of the radio button.
Checked	Gets or sets a value indicating whether the control is checked.

- **Events:**

Event Name	Description
AppearanceChanged	Occurs when the value of the Appearance property of the RadioButton control is changed.
CheckedChanged	Occurs when the value of the Checked property of the RadioButton control is changed.

6. Checkbox Control

- It represents a Windows CheckBox.
- **Properties:**

Property Name	Description
CheckAlign	Gets or sets the horizontal and vertical alignment of the check mark on the check box.
Checked	Gets or sets a value indicating whether the check box is selected.
ThreeState	Gets or sets a value indicating whether or not a check box should allow three check states rather than two.

- **Events:**

Event Name	Description
AppearanceChanged	Occurs when the value of the Appearance property of the check box is changed.
CheckedChanged	Occurs when the value of the Checked property of the CheckBox control is changed.
CheckStateChanged	Occurs when the value of the CheckState property of the CheckBox control is changed.

7. ComboBox Control

- It represents a Windows combo box control.
- **Properties:**

Property Name	Description

DataBindings	Gets the data bindings for the control.
DataSource	Gets or sets the data source for this ComboBox.
DropDownStyle	Gets or sets a value specifying the style of the combo box.

- **Events:**

Event Name	Description
DropDown	Occurs when the drop-down portion of a combo box is displayed.
DropDownClosed	Occurs when the drop-down portion of a combo box is no longer visible.
SelectedIndexChanged	Occurs when the SelectedIndex property of a ComboBox has changed.

8. ListView Control

- It represents a Windows list view control, which displays a collection of items that can be displayed using one of four different views
- **Properties:**

Property Name	**Description**
Columns	Gets the collection of all column headers that appear in the control.
HeaderStyle	Gets or sets the column header style.
Items	Gets a collection containing all items in the control.

- **Events:**

Event Name	Description
ColumnClick	Occurs when a column header is clicked.
ItemCheck	Occurs when an item in the control is checked or unchecked.
SelectedIndexChanged	Occurs when the selected index is changed.

What is Dialog? Explain different dialogs with its usage. (Font Dialog, Save Dialog, Folder Browser Dialog etc…)

- There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc.
- All of these dialog box control classes inherit from the CommonDialog class.
- The ShowDialog method is used to display all the dialog box controls at run-time.
- The values of DialogResult enumeration are:

Abort	returns DialogResult.Abort value, when user clicks an Abort button.
Cancel	returns DialogResult.Cancel, when user clicks a Cancel button.
Ignore	returns DialogResult.Ignore, when user clicks an Ignore button.
No	returns DialogResult.No, when user clicks a No button.
None	returns nothing and the dialog box continues running.
OK	returns DialogResult.OK, when user clicks an OK button

Retry	returns DialogResult.Retry , when user clicks an Retry button
Yes	returns DialogResult.Yes, when user clicks an Yes button

- List of the DialogBox and its descriptions are as below.

DialogBox Name	Description
ColorDialog	It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
FontDialog	It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
OpenFileDialog	It prompts the user to open a file and allows the user to select a file to open.
SaveFileDialog	It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
PrintDialog	It lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

Explain Structure Error handling in VB.Net with Example.

- For Error Handling VB.Net provides Exception Handling, it has a predefined structure n how to use that and when to use exception handling is given below.
- Exceptions provide a way to transfer control from one part of a program to another.VB.Net exception handling is built upon four keywords: Try, Catch, Finally and Throw.
- **Try:**A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch:**A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
- **Finally:**The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
- **Throw:**A program throws an exception when a problem shows up. This is done using a Throw keyword.
- General structure of these keyword are as following;

```
Try
      [Statements]
Catch ex as [Exception Type]
      [Statements]
Catch ex1 as [Exception Type]
      [Statements]
Finally
      [Finally statements]
EndTry
```

- VB.Net provides a structured solution to the exception handling problems in the form of try and catch blocks.

- As shown in the example we had created one method which performs division, now it throws an error when user will perform division by 0.
- Example of the Error Handling in VB.Net is as shown below,

```
Module Module1
    Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
        Dim result As Integer
        Try
            result = num1 \ num2
        Catch e As DivideByZeroException
            Console.WriteLine("Exception caught: {0}", e)
        Finally
            Console.WriteLine("Result: {0}", result)
        End Try
    End Sub
    Sub Main()
        division(25, 0)
        Console.ReadKey()
    End Sub
```

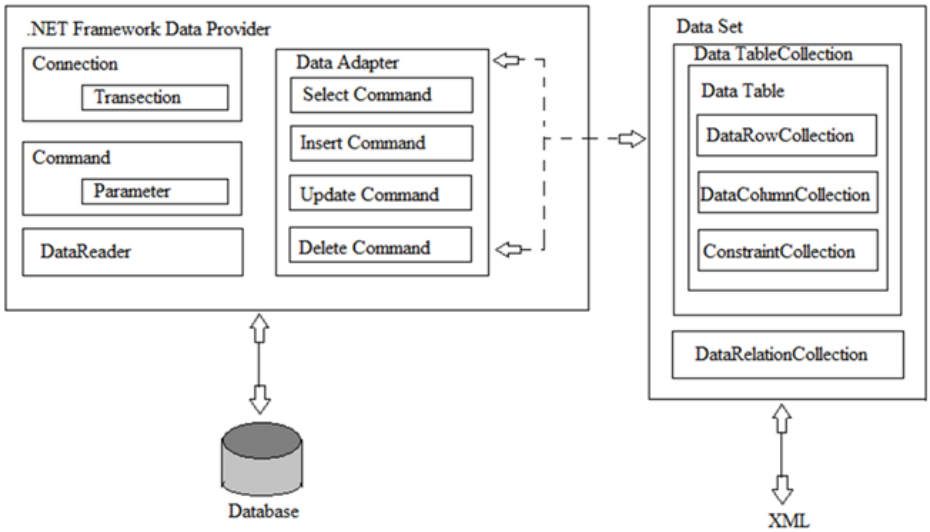
- We had created one method division which takes two parameters and in that method we had perform exception handling.
- While calling method we are passing divider's values is '0', so error will be handle by the inbuilt error handling function DivideByZeroException.
- Finally block will contains the result which we have to display at the end of the error handling.

ADO.Net

Compare classic ADO and ADO.Net. Also explain advantages of ADO.Net compare to classic ADO.

ADO	ADO.Net
It is a COM based Library	It is a CLR based Library
ADO works in the connected mode to access data	ADO.Net works in the disconnected mode to access data
Locking features is available	Locking features is not available
Data is stored in Binary Format	Data is stores in XML
XML integration is not possible	XML integration is possible
It uses RecordSet to store the data from datasource	It uses Dataset to store the data from datasource
Using classic ADO, you can obtain information from one table or set of tables through join. You cannot fetch records from multiple tables independently	Dataset object of ADO.Net includes collection of DataTable wherein each DataTable will contain records fetched from a particular table. Hence multiple table records are maintained independently
Firewall might prevent execution of Classic ADO	ADO.Net has firewall proof and its execution will never be interrupted
Classic ADO architecture includes client side cursor and server side cursor	ADO.Net architecture doesn't include such cursors
You cannot send multiple transaction using a single connection instance	You can send multiple transaction using a single connection instance

Explain ADO.Net Architecture with figure.



The ADO.NET architecture has two main parts:

- Data provider(connected Objects or Connection oriented objects)
- Data Set (Disconnected objects or connectionless objects)

Data Provider

- The .NET framework Data Provider are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
- .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results.

- The Data Provider has four core objects:
  - Connection:
    - The Connection objects provider connectivity to a data source.
  - Command:
    - The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.

Data Reader

- The Data Reader provides a high-performance stream of data from the data source.

Data Adapter

- The Data Adapter provides the bridge between the Data Set object and the data source.
- The Data Adapter uses command object to execute SQL commands at the data source to both load the Data Set with data and reconcile changes that were made to the data in the dataset back to the data source.
- The following lists the data providers that are included in the .NET framework.

Data Provider	Description
SQL Server	○ Provides data access for Microsoft SQL server. ○ Uses the System.Data.SqlClient namespace.
OLEDB	○ For data sources exposed by using OLEDB. ○ Uses the System.Data.OleDb namespace.
ODBC	○ For data sources exposed by using ODBC. ○ Uses the System.Data.Odbc namespace.
Oracle	○ For Oracle data sources. ○ Uses the System.Data.OracleClient namespace.

Data Set

- The dataset object is central to supporting disconnected, distributed data scenarios with ADO.NET.
- The dataset is a memory-resident representation of data that provides consistent relational programming model regardless of the data source.
- The dataset represents a complete set of data, including related tables, constraints, and relationship among the table.
- The dataset has two major objects:

1. **The Data Table Collection:**
   - The Data table collection contains all the data table objects in a dataset.
   - A Data table is defined in the System.Data namespace and represents a single table of memory-resident data.
   - It contains a collection of columns represented by a data column collection, and constraints represented by a constraint collection, which together define the schema of the table.
2. **The Data Relation Collection:**
   - A relationship represented by the Data relation object, associated rows in one Data table with rows in another Data table.
   - A relationship is analogous to a join path that might exist between primary and foreign key columns in a relational database.
   - A data relation identifies matching columns in two tables of a dataset.
   - The essential element of a data relation are:
     - The name of the relationship
     - The name of the tables being related
     - The related column in each table
   - Relationship can be built with more than one column per table by specifying an array of Data Column objects as the key columns.
   - When you add a relationship to the data relation collection, you can optionally add a Unique key constraint to enforce integrity constraints when changes are made to related column values.

Explain different ADO.Net objects with Example.

- ADO.Net is designed to help developers work efficiently with multi tier databases, across internet.
- ADO.Net object model consists of two key components as follows:
  1. Connected Model
  2. Disconnected Model

Connected Model:

- **Connection:**
  - The Connection object is the first component of ADO.NET. The connection object opens a connection to your data source.
  - Connection object helps in accessing and manipulating a database. Database transactions are also dependent upon the Connection object.
- **Command:**
  - The Command object is used to perform action on the data source. Command object can execute stored procedures and T-SQL commands.
  - You can execute SQL queries to return data in a DataSet or a DataReader object. Command object performs the standard Select, Insert, Delete and Update T-SQL operations.
- **Data Reader:**

- The DataReader is built as a way to retrieve and examine the rows returned in response to your query as quickly as possible.
- The data returned by a DataReader is always read only. This class was built to be a lightweight forward only, read only, way to run through data quickly
- DataReader object works in connected model.

- **Data Adapter:**
  - The DataAdapter takes the results of a database query from a Command object and pushes them into a DataSet using the DataAdapter.Fill() method.
  - Additionally the DataAdapter.Update() method will negotiate any changes to a DataSet back to the original data source.
- Now as shown in the below example we had created one windows application to get the records of all braches with its branch code.
- In our example we had created object of the connection, command, data reader and data adapter.
- After click on the button, message box will be pop up and in that message box branch name and branch code will be displayed.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string connetionString = null;
            SqlConnection sqlCnn;
            SqlCommand sqlCmd ;
            SqlDataAdapter adapter = new SqlDataAdapter();
            DataSet ds = new DataSet();
            int i = 0;
            string sql = null;
            connetionString = @"Data Source=MYPC\SQLExpress;Initial
Catalog=AddressBook;Integrated Security=True";
            sql = "Select * from Branch_Detail";
            sqlCnn = new SqlConnection(connetionString);
            try
            {
                sqlCnn.Open();
                sqlCmd = new SqlCommand(sql, sqlCnn);
                adapter.SelectCommand = sqlCmd;
```

```
                adapter.Fill(ds);
                 for (i = 0; i <= ds.Tables[0].Rows.Count - 1; i++)
                 {
                     MessageBox.Show(ds.Tables[0].Rows[i].ItemArray[0] + " -- " +
ds.Tables[0].Rows[i].ItemArray[1]);
                 }
                 adapter.Dispose();
                 sqlCmd.Dispose();
                 sqlCnn.Close();
             }
             catch (Exception ex)
             {
                 MessageBox.Show("Can not open connection ! ");
             }
         }
     }
}
```

Disconnected Model

Dataset

- The DataSet Object is the parent object to most of the other objects in the System.Data namespace.
- The dataset is a disconnected, in-memory representation of data.
- DataSet also contains several methods for reading and writing XML, as well as merging other DataSets, DataTables and DataRows.

DataTable

- DataTable stores a table of information, typically retrieved from a data source.
- DataTable allows you to examine the actual rows of a DataSet through rows and columns collections.
- DataTable also stores metatable information such as the primary key and constraints.

DataRows

- The DataRow class permits you to reference a specific row of data in a DataTable.
- This is the class that permits you to edit, accept, or reject changes to individual DataColumns of the row.

DataColumns:

- DataColumns is the building block of the DataTable. A number of such objects make up a table.
- Each DataColumn object has a DataType property that determines the kind of data that the column is holding data table.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace DataTable_Datarow_DataColumn_Example
{
    class Program
```

```
{
        static void Main(string[] args)
        {
            string strcon = @"Data Source=ANS-pc\SQLExpress;Initial
Catalog=St;Integrated Security=True";
            SqlConnection con = new SqlConnection(strcon);
            con.Open();
            string strquery = "select * from Faculty";
            SqlCommand cmd = new SqlCommand(strquery, con);
            cmd.CommandType = CommandType.Text;
            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataTable dt = new DataTable();
            sda.Fill(dt);
            Console.WriteLine("ID    Name     Initial");
            foreach (DataRow dr in dt.Rows)
            {
                foreach (DataColumn dc in dt.Columns)
                {
                    Console.Write(dr[dc]);
                    Console.Write(" ");
                }
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

- As shown in our example we had created one console application, in that console application we had created object of the datatable, datarow, and datacolumn.
- With the help of these objects we can manage all these data easily and can be manipulated easily.
- All of these object and its method will be worked in the disconnected model, it will not be worked in the connected model.

Explain Typed Dataset.

- A typed dataset is very much similar to a normal dataset. But the only difference is that the schema is already present for the same.
- Hence any mismatch in the column will generate compile time errors rather than runtime error as in the case of normal dataset.
- Also accessing the column value is much easier than the normal dataset as the column definition will be available in the schema.
- The basic program or schema as shown below:

```
//Create DataAdapter
SqlDataAdapter daEmp = new SqlDataAdapter("SELECT empno,empname,empaddress FROM
EMPLOYEE",conn);
//Create a DataSet Object
EmployeeDS dsEmp = new EmployeeDS ();
//Fill the DataSet
daEmp.Fill(dsEmp,"EMPLOYEE");
//Let us print first row and first column of the table
Console.Write(dsEmp.EMPLOYEE[0].empno.ToString());
//Assign a value to the first column
dsEmp.EMPLOYEE[0].empno = "12345";//This will generate compile time error.
```

How to create a Typed DataSet??

1. Open VS .Net IDE and Click on File --> New --> Project and Select Console Application.
2. Enter name for the project. Say TypedDataSetTest.
3. Right click on the solution and click on Add--> Add New Item will show a dialog box.
4. Select DataSet from templates pane, give the name (Say TypedDs.xsd) and click on Open. This will add file by name TypedDs.xsd to the solution.
5. Click on the Server Explorer browse to the database and drop the table on the TypedDs.xsd file.
6. This dataset can be used in the same manner as the normal dataset to get the data.

C#.Net

Explain Constructor and Destructors in c#.

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type.
- A default constructor does not have any parameter.
- If you need a constructor can have parameters. Such constructors are called parameterized constructors. This technique helps you to assign initial value to an object at the time of its creation.
- A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Destructors cannot be inherited or overloaded.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Constructor
{
    class cons {
        public  cons() {
            int a = 10;
            int b = 20;
            int c = a * b;
            Console.WriteLine("No Parameter:s {0}",c);
        }
        public cons(int a, int b) {
            int c = a * b;
            Console.WriteLine("With Parameter: {0}",c);
        }
       ~cons() {
            Console.WriteLine("Destructor Called...!!!");
            Console.ReadKey();
        }
    }

    class Program {
        static void Main(string[] args)  {
            cons c1 = new cons();
            cons c2 = new cons(3, 4);
        }
    } }
```

- As shown in the example we had created two constructor, first constructor with no parameters and the second constructor with two parameters.
- Now when object of the class will be destroyed then destructor will be called as shown in example.

Explain function overloading and operator overloading in c#.

- The process of creating more than one method in a class with same name or creating a method in derived class with same name as a method in base class is called as method overloading.
- In C# no need to use any keyword while overloading a method either in same class or in derived class.
- While overloading methods, a rule to follow is the overloaded methods must differ either in number of arguments they take or the data type of at least one argument.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Method_Overlaoding
{
    class Base_Class {
        public int sum(int a, int b)
        {
            return a + b;
        }
        public float sum(int a, float b)
        {
            return a + b;
        }
    }

    class Derived_Class : Base_Class   {
        public int sum(int a, int b, int c)
        {
            return a + b + c;
        }
    }
    class Program {
        static void Main(string[] args)
        {
            Derived_Class obj = new Derived_Class();
            Console.WriteLine(obj.sum(10, 20));
            Console.WriteLine(obj.sum(10,30.23f));
            Console.WriteLine(obj.sum(10, 20, 30));
            Console.ReadKey();
        }
    }}
```

- As shown in our example we had created two classes Base_Class and Derived_Class, now in our main method we had created object of the Derived_Class.
- Now with the help of that object we are calling method with different number of arguments, while passing the arguments if two methods have same number of arguments but datatype of arguments are different then while passing the values we have to tell the compiler, that which type of the argument you are passing, as shown in our example **"10,30.23f",** here we are telling the compiler that which method have the second parameter as a float that method we have to call.

Operator Overloading:

- Overloaded operators are functions with special names the keyword **operator** followed by the symbol for the operator being defined.
- Like any other function, an overloaded operator has a return type and a parameter list.
- Example shows implements the addition operator (+) for a user-defined class Box. It adds the attributes of two Box objects and returns the resultant Box object.

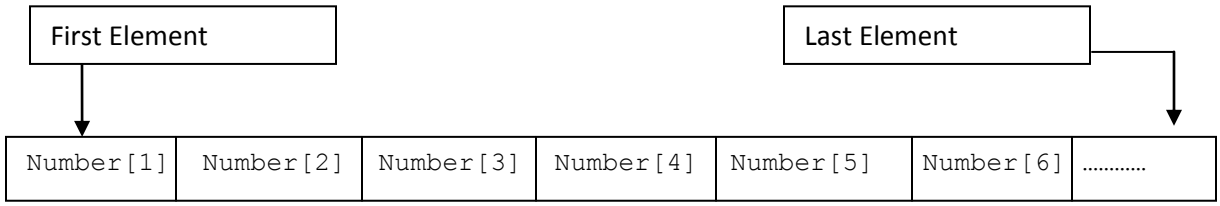
```
using System;
namespace OperatorOvlApplication {
   class Box {
      private double length;      // Length of a box
      private double breadth;     // Breadth of a box
      private double height;      // Height of a box
      public double getVolume()  {
         return length * breadth * height;}
      public void setLength( double len ) {
         length = len;      }
      public void setBreadth( double bre ){
         breadth = bre;     }
      public void setHeight( double hei ) {
         height = hei;      }
      // Overload + operator to add two Box objects.
      public static Box operator+ (Box b, Box c)
      {
         Box box = new Box();
         box.length = b.length + c.length;
         box.breadth = b.breadth + c.breadth;
         box.height = b.height + c.height;
         return box;
      }
   }
class Tester
{
      static void Main(string[] args)
      {
         Box Box1 = new Box();       // Declare Box1 of type Box
         Box Box2 = new Box();       // Declare Box2 of type Box
         Box Box3 = new Box();       // Declare Box3 of type Box
         double volume = 0.0;        // Store the volume of a box here
        // box 1 specification
         Box1.setLength(6.0);
         Box1.setBreadth(7.0);
         Box1.setHeight(5.0);
```

```
// box 2 specification
          Box2.setLength(12.0);
          Box2.setBreadth(13.0);
          Box2.setHeight(10.0);

          // volume of box 1
          volume = Box1.getVolume();
          Console.WriteLine("Volume of Box1 : {0}", volume);

          // volume of box 2
          volume = Box2.getVolume();
          Console.WriteLine("Volume of Box2 : {0}", volume);

          // Add two object as follows:
          Box3 = Box1 + Box2;

          // volume of box 3
          volume = Box3.getVolume();
          Console.WriteLine("Volume of Box3 : {0}", volume);
          Console.ReadKey();
       }
    }
}
```

Explain Array and Array List in .Net with Example. Explain various methods of Array List

Array:

- An array stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- A specific element in an array is accessed by an index.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.



- One dimensional array can be declared as
- double [ ] sample = {23.45,56.78,78.90};
- int [] marks = new int[5] {86,78,89,56,34};
- int [] marks = new int[] {86,78,89,56,34};
- Multi-dimensional arrays are also called rectangular array.
- The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays.
- A two dimensional array can be thought of as a table which will have x number of rows and y number of columns.
- Example of multi dimensional array is as following;

```
class program
    {
         public   static void Main(string[] args)
        {
            int[,] sample = new int[2,3] { {1,2,3},{4,5,6} };
            int i, j;
            for (i = 0; i < 2; i++)
            { for (j = 0; j < 3; j++ )
                {  Console.WriteLine("sample{0} ,{1} = {2} ", i, j,
sample[i,j]); }}
      Console.ReadKey();}}
```

ArrayList:

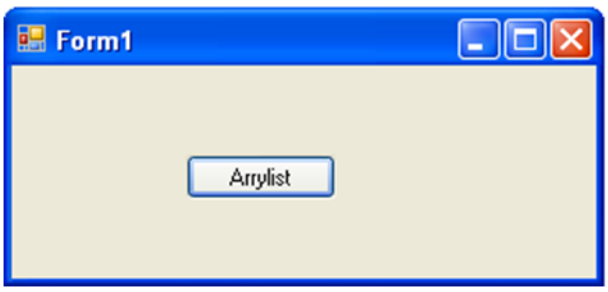
- Represents an array of objects whose size is dynamically increased as required.
- Basic operation for ArrayList:
- Add: Add an Item in an ArrayList
- Insert: Insert an Item in a specified position in an ArrayList
- Remove: Remove an Item from ArrayList
- Remove At: remove an item from a specified position
- Sort: Sort Items in an ArrayList
- **Methods of the ArrayList:**
  - **Add:**
    - How to add an Item in an ArrayList ?
    - Syntax: ArrayList.add (object)
    - Object: The Item to be add the ArrayList

- ArrayList arr;
- arr.Add ("Item1");

- **Insert:**
  - How to Insert an Item in an Array List ?
  - Syntax: ArrayList.insert (index, object)
  - Index: The position of the item in an ArrayList
  - object : The Item to be add the ArrayList
  - ArrayList arr;
  - arr.Insert(3, "Item3");
- **Remove:**
  - How to remove an item from arrayList ?
  - Syntax : ArrayList.Remove(object)
  - object : The Item to be add the ArrayList
  - arr.Remove("item2");
- **Remove At:**
  - How to remove an item in a specified position from an ArrayList ?
  - Syntax : ArrayList.RemoveAt(index)
  - index : the position of an item to remove from an ArrayList
  - ItemList.RemoveAt(2)
- **Sort:**
  - How to sort ArrayList ?
  - Syntax : ArrayList.Sort()

Code: ArayList

- Step-1 Design a Win form with simple Button Control



- Double click on the buttion ArrayList which is shown in the figure, now click event of the button we have to write the following code,

```
using System.Collections;

private void btnArraylist_Click(object sender, EventArgs e){
            int i = 0;
            ArrayList ItemList = new ArrayList();
            ItemList.Add("Item4");
            ItemList.Add("Item5");
            ItemList.Add("Item2");
            ItemList.Add("Item1");
            ItemList.Add("Item3");
            MessageBox.Show ("Shows Added Items");
```

```
for (i = 0; i<= ItemList.Count - 1; i++)
            {
                MessageBox.Show(ItemList[i].ToString());
            }
            //insert an item
            ItemList.Insert(3, "Item6");
            //sort itemms in an arraylist
            ItemList.Sort();
            //remove an item
            ItemList.Remove("Item1");
            //remove item from a specified index
            ItemList.RemoveAt(3);

            MessageBox.Show("Shows final Items the ArrayList");
            for (i = 0; i<= ItemList.Count - 1; i++)
            {
                MessageBox.Show(ItemList[i].ToString());
            }
        }
```

Explain Concept in c# (Modifiers, Properties and Indexers, Attributes and Reflection)

Modifiers

PUBLIC	The item is visible to any other code, no restrictions on the use of a class declared as Public.
PRIVATE	The item is visible only inside the type to which it belongs, members in class A that are marked private are accessible only to methods of class A.
PROTECTED	The item is visible only to any derived type, members in class A that are marked protected are accessible to methods of class A and also to methods of classes derived from class A.
INTERNAL	The item is visible only within its containing assembly, members in class A that are marked internal are accessible to methods of any class in A's assembly.
INTERNAL PROTECTED	The item is visible to any code within its containing assembly and also to any code inside a derived type, members in class A that are marked protected internal are accessible to

	methods of class A, to methods of classes derived from class A, and also to any class in A's assembly.

Properties and Indexers (What is Property? What is indexer? Differentiate Property & Indexer. Explain it with Suitable Example.)

- **Properties** are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called Fields.
- Properties are an extension of fields and are accessed using the same syntax.
- They use accessors through which the values of the private fields can be read, written or manipulated.
- Properties do not name the storage locations. Instead, they have accessors that read, write, or compute their values.
- For example, let us have a class named Student, with private fields for age, name and code. We cannot directly access these fields from outside the class scope, but we can have properties for accessing these private fields.
- **An indexer** allows an object to be indexed like an array.
- When you define an indexer for a class, this class behaves like a virtual array.
- You can then access the instance of this class using the array access operator ([ ]).
- Declaration of behavior of an indexer is to some extent similar to a property. Like properties, you use get and set accessors for defining an indexer.
- However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance.
- In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.
- Defining a property involves providing a property name. Indexers are not defined with names, but with **this** keyword, which refers to the object instance.

Properties	**Indexer**
Identified by its name.	Identified by its signature.
Accessed through a simple name or a member access.	Accessed through an element access.
Can be a static or an instance member.	Must be an instance member.
A get accessor of a property has no parameters.	A get accessor of an indexer has the same formal parameter list as the indexer.
A set accessor of a property contains the implicit value parameter.	A set accessor of an indexer has the same formal parameter list as the indexer, in addition to the value parameter.

- Example of the properties and indexer are as following,

(Properties Example)

```
namespace Properties_Example
{
    class Property_Examle    {
        private string sub_name;
        private int sub_code;
        public string  prt_sub_name     {
            get { return sub_name; }
            set { sub_name = value; }   }
        public int prt_sub_code  //Property Declared for sub_code
        {
            set { sub_code = value; }
                  get { return sub_code; }    }

       public void WriteData()
        {
            Console.WriteLine("Subject Name:{0}",sub_name);
            Console.WriteLine("Subject Code:{0}\n",sub_code);
        }
    }
       class Program
      {
        static void Main(string[] args)
        {
              Property_Examle p = new Property_Examle();
             p.prt_sub_name = "IAP";
             p.prt_sub_code = 2350701;
             p.WriteData();
             p.prt_sub_name = "Java";
             p.prt_sub_code = 2350703;
             p.WriteData();
              Console.ReadKey();
        }}}
```

(Indexer Example)

```
namespace Indexers
{
    class Indexers_Example
    {
        private string[] sub_name = new string[5];
        public string this[int index]
        {
            get { return sub_name[index]; }
            set { sub_name[index] = value; }
        }}
    class Program
    {
        static void Main(string[] args)
        {
            Indexers_Example ie = new Indexers_Example();
            ie[0] = "IAP";
            ie[1] = "Java";
            ie[2] = "MCCP";
            ie[3] = "Project-I";
            ie[4]= "Computer Networks";
            Console.WriteLine("Subject of 5th C.E.");
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Subject-{0}: {1} ",i+1,ie[i]);
            }
            Console.ReadKey(); }}}
```

Attributes and Reflection

- An **attribute** is a declarative tag that is used to convey information to runtime about the behaviors of various elements like classes, methods, structures, enumerators, assemblies etc., in your program.
- You can add declarative information to a program by using an attribute. A declarative tag is depicted by square ([ ]) brackets placed above the element it is used for.
- Attributes are used for adding metadata, such as compiler instruction and other information such as comments, description, methods and classes to a program.
- **Reflection** objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in the System.Reflection namespace.
- The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values and objects to the application.
- Uses of **Reflection:**
  - It allows view attribute information at runtime.
  - It allows examining various types in an assembly and instantiate these types.
  - It allows late binding to methods and properties
  - It allows creating new types at runtime and then performs some tasks using those types.

Explain Reflection API in C# with Example.

- Reflection is the ability of a managed code to read its own metadata for the purpose of finding assemblies, modules and type information at runtime.
- In other words, reflection provides objects that encapsulate assemblies, modules and types.
- By using Reflection in C#, one is able to find out details of an object, method, and create objects and invoke methods at runtime.

```
using System;
using System.Reflection;
namespace Reflection_API
{
    public class MyClass
    {
        public virtual int AddNumb(int numb1, int numb2)
        {
            int result = numb1 + numb2;
            return result;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("\nReflection.MethodInfo");
            MyClass myClassObj = new MyClass();
            Type myTypeObj = myClassObj.GetType();
            MethodInfo myMethodInfo = myTypeObj.GetMethod("AddNumb");
            object[] mParam = new object[] { 5, 10 };
            Console.Write("\nFirst method - " + myTypeObj.FullName + " returns " +
                                    myMethodInfo.Invoke(myClassObj, mParam) + "\n");
            Console.ReadKey();
        }
    }
}
```

Unsafe code in c#

- The **unsafe code** or the unmanaged code is a code block that uses a pointer variable.
- A pointer is a variable whose value is the address of another variable i.e., the direct address of the memory location
- Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- The general form of a pointer variable declaration is:
- `type *var-name;`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnsafeCode_Examle
{
    class Unsafe_Example
    {
        public void UnsafeMethod()
        {
            unsafe
            {
                int x = 10;
                int* ptr1 = &x;
                Console.WriteLine((int)ptr1);
                Console.WriteLine(*ptr1);
            }
        }
    }
    class Program
    {
         static  void Main(string[] args)
        {
            {
                Unsafe_Example obj = new Unsafe_Example();
                obj.UnsafeMethod();
                Console.ReadKey();
            }
        }
    }
}
```

- Now to after writing this code, the code will not directly run because visual studio doesn't allow to run the unsafe application.
- So to execute this application, go to the Solution Explorere→ Right Click on the Solution Name →Go to the Properties →Build → Check the check box Allow unsafe code.
- Now you can run this application without any error.

Explain Events & Delegates with Example in c#

- A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
- Delegates allow methods to be passed as parameters.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- A delegate can refer to a method, which have the same signature as that of the delegate.

- Once a delegate type has been declared, a delegate object must be created with the new keyword and be associated with a particular method.
- When creating a delegate, the argument passed to the new expression is written like a method call, but without the arguments to the method.
- **Events** are basically a user action like key press, clicks, mouse movements etc., or some occurrence like system generated notifications.
- The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class.
- The class containing the event is used to publish the event. This is called the **publisher** class
- Some other class that accepts this event is called the **subscriber** class.
- A **publisher** is an object that contains the definition of the event and the delegate.
- A **subscriber** is an object that accepts the event and provides an event handler.
- To declare an event inside a class, first a delegate type for the event must be declared.
    - `public delegate void SimpleDelegate();`
- Next, the event itself is declared, using the **event** keyword:
    - `public static event SimpleDelegatae Simply;`

```
namespace Event_With_Delegates
{
    delegate void SimpleDelegate();
    class EventExample
    {
        public static event SimpleDelegate Simply;
        private static void Hello()
        {
            Console.WriteLine("It's Event with Delegates Example");
        }
        private static void Hello1()
        {
            Console.WriteLine("It's Delegates Example");
        }
        static void Main(string[] args)
        {
            Simply += new SimpleDelegate(Hello);
            Simply +=new SimpleDelegate(Hello1);
            Simply();
            Console.ReadKey(); } }
```

Explain Stream Reader & Stream Writer class for File Stream.

- File handling is an unmanaged resource in your application system. It is outside your application domain (unmanaged resource). It is not managed by CLR.
- Data is stored in two ways, persistent and non-persistent manner.

- When you open a file for reading or writing, it becomes stream.

Stream:

- Stream is a sequence of bytes traveling from a source to a destination over a communication path.
- The two basic streams are input and output streams. Input stream is used to read and output stream is used to write.
- The **System.IO** namespace includes various classes for file handling.
- The parent class of file processing is stream. Stream is an abstract class, which is used as the parent of the classes that actually implement the necessary operations.
- The primary support of a file as an object is provided by a .NET Framework class called File. This static class is equipped with various types of (static) methods to create, save, open, copy, move, delete, or check the existence of a file.

StreamWriter Class:

- The StreamWriter class in inherited from the abstract class TextWriter. The TextWriter class represents a writer, which can write a series of characters.
- The following table describes some of the methods used by StreamWriter class.

Close	Closes the current StreamWriter object and the underlying stream
Flush	Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream
Write	Writes to the stream
WriteLine	Writes data specified by the overloaded parameters, followed by end of line

```
using System;
using System.Text;
using System.IO;
namespace FileWriting_SW
{
   class Program
   {
      class FileWrite
      {
         public void WriteData()
         {
            FileStream fs= new FileStream("c:\\test.txt", FileMode.Append, FileAccess.Write);
```

```
StreamWriter sw = new StreamWriter(fs);
            Console.WriteLine("Enter the text which you want to write to the file");
            string str = Console.ReadLine();
            sw.WriteLine(str);
            sw.Flush();
            sw.Close();
            fs.Close();
        }
    }
    static void Main(string[] args)
    {
        FileWrite wr = new FileWrite();
        wr.WriteData();
    }
  }
}
```

StreamReader Class:

- The StreamReader class is inherited from the abstract class TextReader. The TextReader class represents a reader, which can read series of characters.
- The following table describes some methods of the StreamReader class.

Close	Closes the object of StreamReader class and the underlying stream, and release any system resources associated with the reader
Peek	Returns the next available character but doesn't consume it
Read	Reads the next character or the next set of characters from the stream
ReadLine	Reads a line of characters from the current stream and returns data as a string
Seek	Allows the read/write position to be moved to any position with the file

```
using System;
using System.IO;
namespace FileReading_SR
{
   class Program
   {
      class FileRead
      {
         public void ReadData()
         {
            FileStream fs = new FileStream("c:\\test.txt", FileMode.Open, FileAccess.Read);
```

```
            StreamReader sr = new StreamReader(fs);
            Console.WriteLine("Program to show content of test file");
            sr.BaseStream.Seek(0, SeekOrigin.Begin);
            string str = sr.ReadLine();
            while (str != null)
            {
                Console.WriteLine(str);
                str = sr.ReadLine();
            }
            Console.ReadLine();
            sr.Close();
            fs.Close();
        }
    }
    static void Main(string[] args)
    {
        FileRead wr = new FileRead();
        wr.ReadData();
    }
}
```

What is Exception in c#? Explain Exception handling in c# with Example.

- An exception is a problem that arises during unsafe code executes of a program.
- Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: **try**, **catch**, **finally** and **throw**.
- **try:** A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception
- **finally:** The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw:** A program throws an exception when a problem shows up. This is done using a throw keyword.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code.
- The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class.
- Some of the exception classes derived from the System.Exception class are the System.ApplicationException and System.SystemException classes.

```
using System;
namespace ErrorHandlingApplication
{
    class DivNumbers
    {
        int result;
        DivNumbers()
        {
            result = 0;
        }
        public void division(int num1, int num2)
        {
            try
            {
                result = num1 / num2;
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception caught: {0}", e);
            }
            finally
            {
                Console.WriteLine("Result: {0}", result);
            }

        }
        static void Main(string[] args)
        {
            DivNumbers d = new DivNumbers();
            d.division(25, 0);
            Console.ReadKey();
        }
    }}
```

What is GDI+? Give example showing use of GDI+?

- The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers.
- GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes.
- It is not directly responsible for drawing windows, menus, etc.; that task is reserved for the user subsystem, which resides in user32.dll and is built atop GDI.
- **Graphic Classes' Methods:**

DrawArc	Draws an arc from the specified ellipse.
DrawBezier	Draws a cubic bezier curve.
DrawBeziers	Draws a series of cubic Bezier curves.
DrawClosedCurve	Draws a closed curve defined by an array of points.
DrawCurve	Draws a curve defined by an array of points.
DrawEllipse	Draws an ellipse.
DrawImage	Draws an image.
DrawLine	Draws a line.
DrawPath	Draws the lines and curves defined by a GraphicsPath.
DrawPie	Draws the outline of a pie section.
DrawPolygon	Draws the outline of a polygon.
DrawRectangle	Draws the outline of a rectangle.
DrawString	Draws a string.
FillEllipse	Fills the interior of an ellipse defined by a bounding rectangle.
FillPath	Fills the interior of a path.
FillPie	Fills the interior of a pie section.
FillPolygon	Fills the interior of a polygon defined by an array of points.
FillRectangle	Fills the interior of a rectangle with a Brush.
FillRectangles	Fills the interiors of a series of rectangles with a Brush.
FillRegion	Fills the interior of a Region.

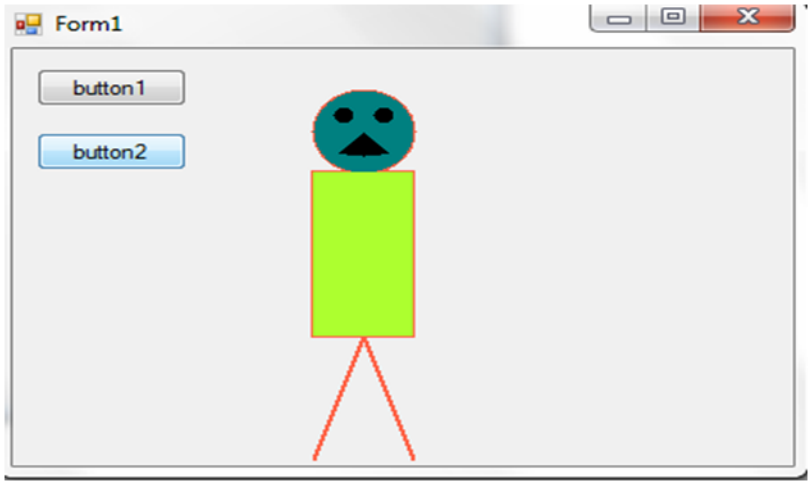
- **Graphic's Object:**

Brush	Used to fill enclosed surfaces with patterns,colors, or bitmaps.
Pen	Used to draw lines and polygons, including rectangles, arcs, and pies
Font	Used to describe the font to be used to render text
Color	Used to describe the color used to render a particular object. In GDI+ color can be alpha blended
Brush	Used to fill enclosed surfaces with patterns,colors, or bitmaps.

- Example of the GDI+ is as following,
    - Step-1 Open the visual studio, click on the new windows application with C# language,
    - Step-2 After that drag and drop two button on the windows forms.
    - Step-3 Double click on the button1 and write the code which is shown in the example
    - Step-4 Double click on the button1 and write the code which is shown in the example
    - Step-5 now run the application.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace GDI_
{
    public partial class Form1 : Form
    {

       public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Graphics g = this.CreateGraphics();
            Pen p = new Pen(Color.Tomato,2);
            g.DrawEllipse(p,150,25,50,50);
            g.DrawEllipse(p, 160, 35, 10, 10);
            g.DrawEllipse(p, 180, 35, 10, 10);
            g.DrawArc(p, 150, 35, 50, 50, 45, 90);
            g.DrawRectangle(p, 150, 75, 50, 100);
            g.DrawLine(p, 175, 175, 150, 250);
            g.DrawLine(p, 175, 175, 200, 250);
        }
        private void button2_Click(object sender, EventArgs e)
        {
            Graphics g = this.CreateGraphics();
            SolidBrush sb = new SolidBrush(Color.Teal);
            g.FillEllipse(sb, 150, 25, 50, 50);
            sb.Color = Color.Black;
            g.FillEllipse(sb, 160, 35, 10, 10);
            g.FillEllipse(sb, 180, 35, 10, 10);
            sb.Color = Color.Black;
            g.FillPie(sb, 150, 35, 50, 30, 45, 90);
            sb.Color = Color.GreenYellow;
            g.FillRectangle(sb, 150, 75, 50, 100);
        }
    }
}
```

Output



ASP.Net

Introduction to ASP.Net:

- Asp.net Refers as Active Server Page .net which is server side object oriented programming language.
- Asp.net is language of Microsoft.net framework to develop web applications.
- Benefits using Asp.net:

Easier OOP Language	Quick Drag & Drop control	Code Separation
Easier database operations	Version Compatibility	High security
Easy app. Development	Highly Integrated IDE	Etc

- Asp.net uses to develop dynamic web pages, websites, web services and web applications.
- Asp.net is built on the CLR (Common Language Runtime) that allows user to write Asp.net program using any .net framework supported languages like C#, VB, Visual C++, J# etc.
- Asp.net also Supports ADO.Net (ActiveX Data Objects.Net) that helps to connect and work with data stored in database.
- Asp.net also provides Web services through which Same Service can be applicable to more than one website. E.g. weather, cricket live score etc.
- Asp.net provides everything Readymade such as Rich controls, Validation Controls, data bound controls, AJAX controls for developing easier, fast and highly dynamic web applications.

Directory Structure of Asp.net:

Bin	App_Code	App_GlobalResources	App_LocalResources
App_Data	App_Themes	App_Browsers	App_Webreferences

Differentiate ASP and ASP.Net

	ASP.Net	**ASP**
Language	OOP language	Scripting Language
Inheritance	Supports	Doesn't Support
Code behind files	Separate file for code and data both	Single file for code and data
Configuration files	Machine.config, Web.Config	No Configuration files
Custom Controls	Supports	Doesn't Support
Database Language	ADO.Net with XML integration	Simple ADO with limited functionalities

Explain ASP.Net Webpage life cycle & Events

When a user request for Asp.net page, the page gone through many stages that is call its life cycle.

Various stages in ASP.Net Web Page

Stages	What is done in Stages?
Page Request	Asp.Net checks that request of the page is new or old, if new then it compiles and executes that page and if request is old, cached copy will be returned.
Start	Request And Response properties are set and using IsPostBack property new or old request can be identified.
Page Initialization	In page, each control is initialized. Themes will be loaded. If requested page is old then post back data is not loaded.
Load	If current request is old one, the property of control is loaded with data from view and control state.
Postback Event Handling	If request for the page is old one, then any event handler can be invoked.
Validation	After loading the page, the validation controls are used to invoke Validate method for error free access.
Rendering	All controls on the page are saved. During rendering, render method for each control on page and writes it to output stream object of page response property.
Unload	Request and response properties of the page are unloaded and cleaning performed if required.

Asp.Net Pages Life Cycle Events

Events	Working Of Events
PreInit	First event of Asp.net Page Life cycle. It checks whether the page is processed for the first time or not. It also creates and recreates dynamic controls, sets master page and themes. It sets and gets profile property.
Init	This event is raised after initialization of all controls and applying skin properties and initialize or read control properties.
Load	This event is called to set control properties and establish database connections. Page class calls OnLoad event of web page after that calls event for each child control, until page is loaded.

Explain various Server Controls of ASP.Net

Asp.net provides many built in server controls that can be easily drag and drop to the web pages.

Controls:

Text Box	Check Box	Radio Button	Label
Button	Hyper Link	Image	Etc.

Common Properties for all controls:

ID	Runat	Text	CssClass
Height	Width	Auto post back	Visible

- System.Web.UI.WebControls
- Server side Controls are useless without runat="server" Property.

- **Text Box:**
  - This control is used to take input from user into a default string format.

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack="true" height="10px"
width="10px" visible="true">
</asp:TextBox>
```

  - Example:

Enter your Name….

- **Check Box:**
  - This control is used to select multiple values from a list.

```
<asp:CheckBox ID="CheckBox1" runat="server" AutoPostBack="true"  visible="true">
<asp:CheckBox>
```

  - Example:

Apple ☐ Banana ☐ Grapes ☐ Guava ☐

- **Radio Button:**
  - This control is used to select Single value from a list.

```
<asp:RadioButton ID="RadioButton1" runat="server" AutoPostBack="True" Visible="True"
Group="one">
</asp:RadioButton>

<asp:RadioButton ID="RadioButton1" runat="server" AutoPostBack="True" Visible="True"
Group="one">
</asp:RadioButton>
```

  - Example:

Male ◯ Female ◯

- **Label:**
  - This Control is used to display information on the web Page.

```
<asp:Label ID="Label1" runat="server" Visible="True" Text="Hi This Is Darshan !!!">
<asp:Label>
```

  - Example:

    Hi this Is Darshan !!!

- **Button:**
  - This control provides firing of various events at a single click.
  - Button Provides page navigation as well as event firing.

```
<asp:Button ID="Buttton1" runat="server" Text="Click Me" OnClick="Button1_Click" />
```

  - Example:

    Click Me

- **HyperLink:**
  - This control provides easy navigation between various Pages.
  - NavigateUrl = address of destination page

```
<asp:HyperLink ID="HyperLink1" runat="Server" NavigateUrl="Home.aspx" Text="Home Page"></asp:HyperLink>
```

  - Example:

    Home → navigate to Home page on single Click

- **Image:**
  - This Control is used to display images on the web page.
  - ImageUrl = Location of the image in application folder or in computer.
  - AlternateText= alternate text if image is not found

```
<asp:Image ID="Img" runat="server" ImageUrl="~/images/ab.jpg" AlternateText="Jerry"/>
```

  - Example:

Explain various Validation Controls of ASP.Net

- Asp.Net provides various validation controls for error free access for the web page. This validation Controls validates the input for controls. So that any mismatching and faulty data cannot be inserted to the data storage.
- Various Validation controls are:
  - Required Field Validator
  - Range Validator
  - Regular Expression Validator
  - Compare Validator
  - Custom Validator
  - Validation Summary

Common Properties

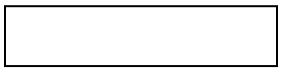
ID	Runat	ControlToValidate	ErrorMessage

Required Field Validator

- It provides validation for emptiness of any control.
- This Validator is applied to such controls where input is necessary.

```
<asp:TextBox ID=”TextBox1” runat=”server” AutoPostBack=”true” height=”10px” width=”10px”
visible=”true”>
</asp:TextBox >

<asp:RequiredFieldValidator ID=”rfv1” runat=”server” ControlToValidate=”TextBox1”
ErrorMessage=”Text Box must not empty”></ asp:RequiredFieldValidator>
```

- Example:



Text Box must not empty

Range Validator

- It provides validation for range of an input by setting minimum value and maximum value.
- This Validator is used when input is taken between certain type of values.
- E.g. Salary must between 10000 and 50000.
- Special Attributes:
  - MinValue
  - MaxValue

```
<asp:TextBox ID=”TextBox1” runat=”server” AutoPostBack=”true” height=”10px” width=”10px”
visible=”true”>
</asp:TextBox >

<asp:RangeValidator ID=”rv1” runat=”server” ControlToValidate=”TextBox1” ErrorMessage=”Salary
between 10000 to 50000” MinValue=”10000” MaxValue=”50000”> </asp:RangeValidator >
```

Example

9999	salary between 10000 to 50000

Regular Expression Validator

- It provides complex find of validation.
- Such as email validation.
- Validation can be done using regular expressions to validate special inputs.
- Special attributes:
    - Validation expression

```
<asp:TextBox ID=”TextBox1” runat=”server” AutoPostBack=”true” height=”10px” width=”10px”
visible=”true”>
</asp:TextBox >

<asp:RegularExpressionValidator ID=”rev1” runat=”server” ControlToValidate=”TextBox1”
ErrorMessage=”Email is not Valid”> </asp:RegularexpressionValidator >
```

Example

abc@yahoo_in	Email is not valid

Compare Validator:

- It compares two controls of same data type.
- Validation can be done on two controls.
- E.g. Comparison of password and re-password
- Special attribute:
    - Control to Compare

```
<asp:TextBox ID=”TextBox1” runat=”server” AutoPostBack=”true” height=”10px” width=”10px”
visible=”true”>
</asp:TextBox >

<asp:TextBox ID=”TextBox2” runat=”server” AutoPostBack=”true” height=”10px” width=”10px”
visible=”true”>
</asp:TextBox >

<asp:CompareValidator ID=”rv1” runat=”server” ControlToValidate=”TextBox2”
```

```
ControlToCompare="TextBox2"  ErrorMessage="Value doesn't match" > </asp:CompareValidator >
```

Example:

abc@yahoo_in	abc@yahooin	Value doesn't match

Custom validator:

- This validator validates the input as per user's preferences.
- It is having a function that calls the validation logic.
- E.g. birthdate validation, user must be above 18 years etc.
- Special Attributes
    - ClientValidationFunction
    - Funaction written in code behind file

```
<asp:TextBox ID="TextBox2" runat="server" AutoPostBack="true" height="10px" width="10px"
visible="true">
</asp:TextBox >

<asp:CustomValidator ID="cv1" runat="server" ControlToValidate="TextBox2"
ClientValidationFunction="checkAge"  ErrorMessage="Age must be above 18" >
</asp:CustomValidator>
```

Example:

22/10/1999	Age must above 18

Validation Summary:

- Validation summary provides all the validation errors as a list at one time at the end.
- Special attributes:
    - ShowSummary (Boolean)

Explain various List Controls in ASP.Net

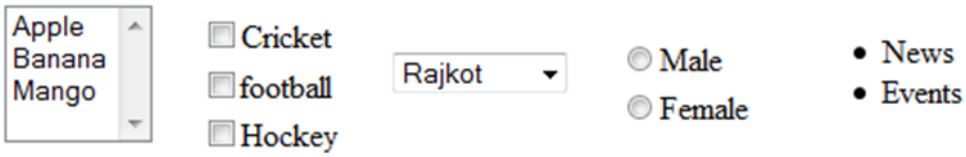
- Asp.Net provides many built in List controls to show the data in the form of list.
- User can select one or more value from the list.
- List controls are many and they are used as per user's requirements.

Controls	Functions	Examples
ListBox	Provides all the data in a box format and user can select one and more data.	List of fruits
CheckBoxList	Provides a checkbox with each data and user can select more than one item.	Hobbies
DropDownList	Provides a list of items with only one item visible at a time and user can select single or multiple items.	Country List
RadioButtonList	Provides a list from which only one item can be select.	Gender
Bulleted List	Used to show Important Highlighted data.	Imp. Points

Let us take a simple example to explain all controls at once.

Example

```
<asp:ListBox ID="ListBox1" runat="server">
   <asp:ListItem>Apple</asp:ListItem>
   <asp:ListItem>Banana</asp:ListItem>
   <asp:ListItem>Mango</asp:ListItem>
</asp:ListBox>

<asp:CheckBoxList ID="CheckBoxList1" runat="server">
   <asp:ListItem>Cricket</asp:ListItem>
   <asp:ListItem>football</asp:ListItem>
   <asp:ListItem>Hockey</asp:ListItem>
</asp:CheckBoxList>

<asp:DropDownList ID="DropDownList1" runat="server">
   <asp:ListItem>Rajkot</asp:ListItem>
   <asp:ListItem>hadala</asp:ListItem>
   <asp:ListItem>Junagadh</asp:ListItem>
</asp:DropDownList>

<asp:RadioButtonList ID="RadioButtonList1" runat="server">
   <asp:ListItem>Male</asp:ListItem>
   <asp:ListItem>Female</asp:ListItem>
</asp:RadioButtonList>

<asp:BulletedList ID="BulletedList1" runat="server">
   <asp:ListItem>News</asp:ListItem>
   <asp:ListItem>Events</asp:ListItem>
</asp:BulletedList>
```



Explain various Data Controls of ASP.Net

- Data controls are used to display records from database in a repeated manner.
- The results can be customized according to the query style and user preferences.
- Custom edit and paging and select record type functionalities are ready made available.
- Just drag and drop data controls and display the records according to your need.
- Some Controls are mentioned and explained below:

Grid View, Data List, Repeater Control

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True"
AutoGenerateColumns="False" DataKeyNames="Id" DataSourceID="SqlDataSource1">
        <Columns>
          <asp:BoundField DataField="Id" HeaderText="Id" InsertVisible="False"
            ReadOnly="True" SortExpression="Id" />
          <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
          <asp:BoundField DataField="College" HeaderText="College"
            SortExpression="College" />
        </Columns>
      </asp:GridView>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
        SelectCommand="SELECT * FROM [Student]"></asp:SqlDataSource><br />

<asp:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource2">
      <HeaderTemplate>
        <table border=1 width="150px">
          <tr><td>Id</td>
          <td>Name</td>
          <td>College</td></tr>    </table>
      </HeaderTemplate>
      <ItemTemplate>
        <table border=1 width="150px">
          <tr><td><asp:Label ID="id" Text='<%#Eval("Id") %>' runat="server">
</asp:Label></td>
          <td><asp:Label ID="Name" Text='<%#Eval("Name") %>' runat="server">
</asp:Label></td>
          <td><asp:Label ID="College" Text='<%#Eval("College") %>' runat="server">
```

```
</asp:Label></td></tr>
        </table>
    </ItemTemplate>
</asp:Repeater>
    <asp:SqlDataSource ID="SqlDataSource2" runat="server"
      ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
      SelectCommand="SELECT * FROM [Student]"></asp:SqlDataSource>

<asp:DataList ID="DataList1" runat="server" DataKeyField="Id"
      DataSourceID="SqlDataSource3" RepeatDirection="Horizontal">
<ItemTemplate>
Id:<asp:Label ID="Id" runat="server" Text='<%# Eval("Id") %>' /><br />
Name:<asp:Label ID="Name" runat="server" Text='<%# Eval("Name") %>' />
College:<asp:Label ID="College" runat="server" Text='<%# Eval("College") %>' />  </ItemTemplate>
</asp:DataList>
    <asp:SqlDataSource ID="SqlDataSource3" runat="server" ConnectionString="<%$
ConnectionStrings:ConnectionString %>"
      SelectCommand="SELECT * FROM [Student]"></asp:SqlDataSource>
```

All the attributes are explained in previous example.

- The Output can be like this:

Id	Name	College
1	Jay	RKU
2	Maulik	DIET
3	Jatin	Marwadi

Id	Name	College
1	Jay	RKU
2	Maulik	DIET
3	Jatin	Marwadi

Id: 1 Id: 2 Id: 3
Name: Jay Name: Maulik Name: Jatin
College: RKU College: DIET College: Marwadi

Explain various Rich server Controls in ASP.Net

- Rich server controls are used to deploy rich applications in very easy manner.
- Such as to rotate advertisements in webpage.
- Rich customized online calendar control.

Ad rotator

- Ad rotator controls are used to rotate advertisements on web screen using XML.
- XML file make the control more effective.

Example Ad rotator

```
<asp:AdRotator ID="AdRotator1"   runat="server" AdvertisementFile ="~/ads.xml"
            onadcreated="AdRotator1_AdCreated" />

//Ads.xml file//

<Advertisements>
<Ad>
<ImageUrl>rose1.jpg</ImageUrl>
<NavigateUrl>http://www.1800flowers.com</NavigateUrl>
<AlternateText> Order flowers, roses, gifts and more </AlternateText>
<Impressions>20</Impressions>
<Keyword>flowers</Keyword>
</Ad>

<Ad>
<ImageUrl>rose2.jpg</ImageUrl>
<NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
<AlternateText>Order roses and flowers</AlternateText>
<Impressions>20</Impressions>
<Keyword>gifts</Keyword>
</Ad>

<Ad>
<ImageUrl>rose3.jpg</ImageUrl>
<NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
<AlternateText>Send flowers to Russia</AlternateText>
<Impressions>20</Impressions>
<Keyword>russia</Keyword>
</Ad>
</Advertisements>
```

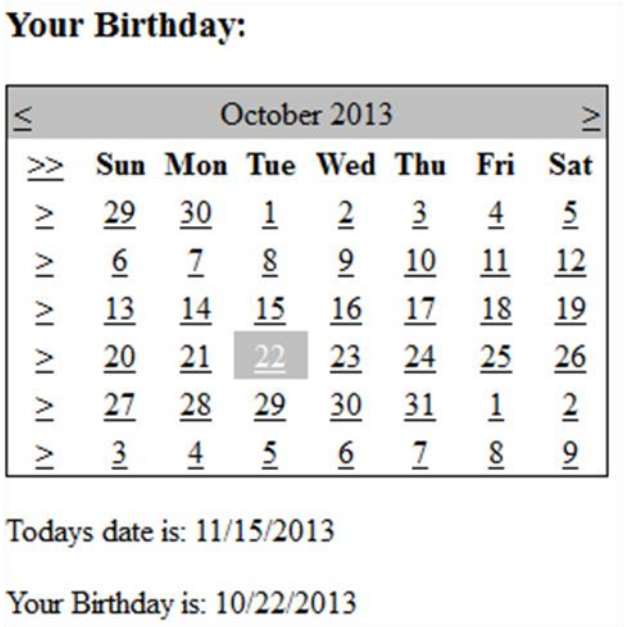
Calendar Control

- Calendar controls are used to maintain online calendar.
- Customized calendar is developed with reminders, birthdays etc.

Example Calendar Control:

```
<h3> Your Birthday:</h3>
<asp:Calendar ID="Cal1" runat="server" SelectionMode="DayWeekMonth"
          onselectionchanged="Calendar1_SelectionChanged">
</asp:Calendar>
</div>
<p>Todays date is:   <asp:Label ID="lblday" runat="server"></asp:Label></p>
<p>Your Birthday is:   <asp:Label ID="lblbday" runat="server"></asp:Label> </p>

//Code Behind file//
```

```
protected void Calendar1_SelectionChanged(object sender,  EventArgs e)
{
   lblday.Text = Calendar1.TodaysDate.ToShortDateString();
   lblbday.Text = Calendar1.SelectedDate.ToShortDateString();
}
```



Explain various State management policies in ASP.Net

- You can preserve the state of the application either at the server or the client end.
- The state of a web application helps you to store the runtime changes that have been made to the web application.
- For example, a user selects and saves some products in the shopping cart of an online shopping websites.
- For that you have to store it to state, otherwise the changes are discarded.
- There are various methods for storing state information.
    - Hidden fields
    - Cookies
    - Query Strings
    - Application state
    - Session State
    - Profile Properties

Page-Level State [View State]

- In Asp.Net, the ViewState property is used to store the page-level state of a web page, the ViewState property the state information of a single user, as long as the user is working with the page.
- The ViewState property is used when a form is submitted and presented second time to a user, as it retains the information entered in the form's controls first time.
- The VIEWSTATE property at each page is initialized and stored in ViewState.
- The string is assigned to the Value attribute of VIEWSTATE field and is sent as a part of the web page.

```
ViewState["User-definedKey"] = user-definedValue;
Response.Write(ViewState["user-DefinedKey"]);
```

Cookies

- Cookie is a small amount of data that is stored as a text file on client system or in client browser's session.
- It stores site specific information of a user, such as user name and inputs.
- Cookies can be created temporarily with the specific expiration time.
- Cookies can be created permanent, called as persistent cookies.
- When a page is requested, the client sends information in form of cookies along request information.
- The server reads cookie and extract value.

```
HttpCookie kukie;
If (Request.Cookies[txtName.text] == null)
Kukie = new HttpCookie(txtName.Text, txtValue.Text);
Kukie.values.Add(txtvalname.Text, txtvalvalue.Text);
Kukie.Expires = System.DateTime.Now.AddDays(1);
Response.AppendCookie(Kukie);
```

ASP.Net Session State

- Session is defined as the period of time, in which a user interacts with a web application.
- Session state is a collection of data, which are related to a session and stored on a server.
- Session state acts as memory in the form of a hash table with key-value pair.
- Each user is given unique session ID when the session begins.
- Session state can be implemented in the following Modes:

In-Process

- In this mode, session state memory is kept within the Asp.Net process. This mode applies to the web applications that are hosted on a single server.

Out-of-Process

- In this mode, you get performance of reading from memory and the reliability of a separate process that manages the state for all servers.

SQL Server

- Here reliability of data within a web application with help of database.

Configuring Session state

- Following parameters with example coding:

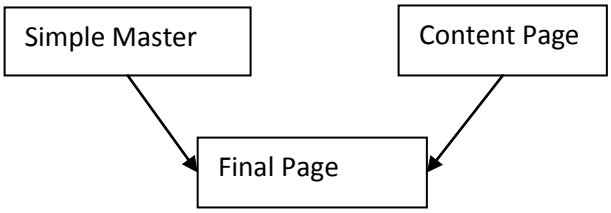
```
<configuration>
  <sessionState
    Mode = "inproc"
    Cookieless="false"
    Timeout="20"
    Sqlconnectionstring= "data source=127.0.0.1;user id = <userid>; password = <Password>"
    Server="127.0.0.1"
    Port="42424"/>
</configuration>
```

Explain the concepts of Master Page and Nested Master Page in ASP.Net

Master page

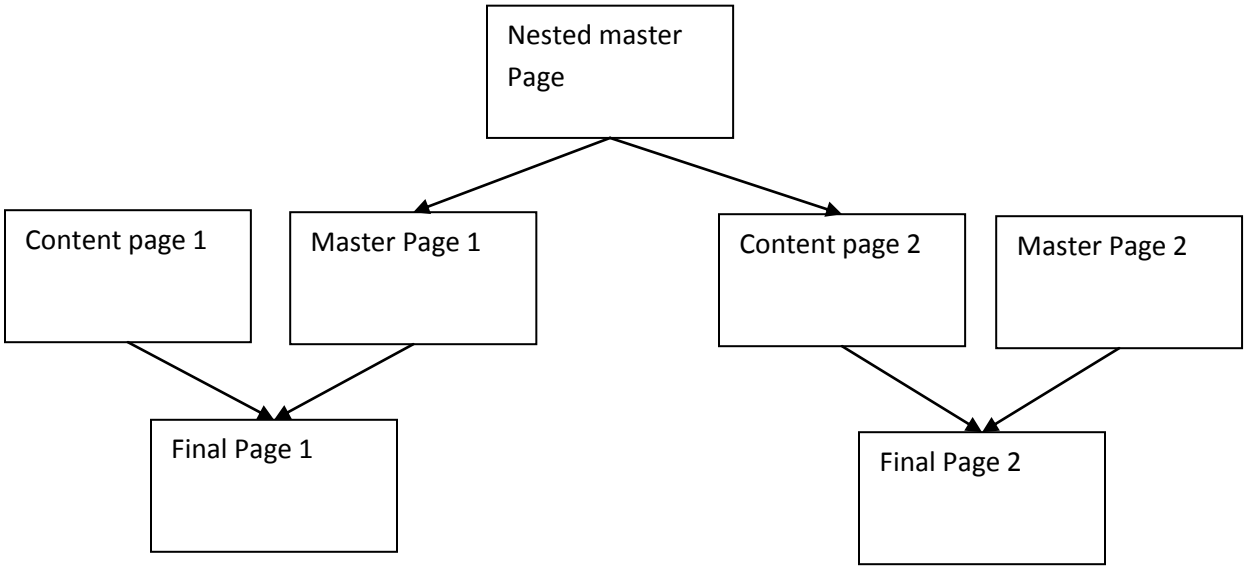
- Master pages enable the template based development of a website.
- Master defines the overall layout of a web application and all pages are derived from master page so that they have a common look.
- Master page having .master extension.
- Define master page using @Master directive on top a web page.
- The Content of master pages is fixed for all derived pages. To place the changing data for every pages master page provide <ContentPlaceHolder> where each page hold its local content.
- <asp:ContentPlaceHolder ID="cph" runat="server">
- <%@ Page language="C#" MasterPageFile="~/masterpage.master" Title= "contentPage">
- Major characteristics of a master page:
- Defining common properties of a website, such as header, footer, banners, navigation menus and other elements that can be accessed by the content pages.
- Allowing single or multiple content pages to access single or multiple master pages.
- Displaying the content of each content page in content place holder.
    - Master pages are of two types:

Simple Master Page



Nested Master Page:

- Here you can merge the master page and content page.
- Simply, Main master page → another master page for some selected pages that is called nested master page.
- It is used to give common and distinguish look from master page for some special module.



Explain the ASP.Net Themes with Skin file

- Themes in Asp.Net enable you to define the style properties and the change the appearance of a Web Page.
- They provides User friendly interface with similar all over look with optimized code.
- You can set theme in master page directive in master page as well in web.config also.
- Theme can be defined as collection of element that is common look and feel for whole application.
- The Key Features are as follows:

- Providing flexibility to easily change the appearance of all web pages just by making changes in few template files with minimum possible efforts.
- Providing various options to customize a web page. E.g. in Orkut, you can change website appearance as per your requirement.

- Themes contain three elements:

Skin

- It is having .skin extension that contains tags, property settings and formatting options for server side controls. Such as Label, TextBox and Button.
- Skin is divided into following two groups:

Default:

- Default skin is applied automatically to all the controls of the same type in web page, when theme is applied to the page.
- SkinID attributes is used to determine whether the applied skin is default or not. If SkinID is not present in current tag means it is a default skin.

Named:

- It is a customized skin that is applied to controls when theme is applied.
- It has SkinID property which allows different styles to different controls.

CSS

- Refers to a mechanism of adding fonts, colors, styles and behavior to web pages.
- CSS is used to provide the visual inheritance to all the web pages in the website.
- It is an Asp.Net file with .css extension and applies a style sheet as a part of theme in a web page.
- It should include in App_theme folder.

Image

- Helps the web page making more attractive, interactive and user friendly.
- In addition to skins and CSS, themes can also include other resource file as script files or sound files.

Web Services and XML

What is web Service in .Net? Write a Program to create Web Service and another program to consume same web service. (Take simple web method to add two integer numbers)

- A web service is an entity that you can develop to provide particular functionality over the internet.
- For example weather information, live score of cricket, simple and compound interest formula.
- For such purposes you make a web service for each. It will useful not only for one website but also for many other websites those are providing same facilities.
- Developer just needs to consume the web services and then its required function is working.
- So, it is better to develop a web service common to all instead of writing the same business logic for all websites.
- And moreover web service provides common functionalities and results for all websites.
- And web service irrespective from any programming language as it is based on XML so can be useful to all language based applications.
- Web Service messages are formatted as XML, a standard way for communication between two incompatible system. And this message is sent via HTTP, so that they can reach to any machine on the internet without being blocked by firewall.

Take an example to demonstrate web service

- Right click on project →Add new item → select web services → OK

```
public class WebService1 : System.Web.Services.WebService
{
   [WebMethod]
   public string HelloWorld()   { return "Hello World"; }
   [WebMethod]
   public int sum(int a, int b)
   {  return a+b;  } }
```

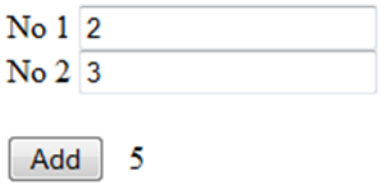
- Now you have to **add web reference** to achieve web services working for any application.
  - Web services for solution only
  - Web services for local machine
  - Web services over internet

Consume web service

- Make an object of web service and you can call any method within that web service.

```
public partial class counter : System.Web.UI.Page
  {
     protected void Page_Load(object sender, EventArgs e){}
     protected void btnAdd_Click(object sender, EventArgs e)
     {
        WebService1 intsum = new WebService1();
        int a = Convert.ToInt32(txt1.Text);
        int b = Convert.ToInt32(txt2.Text);
        lblSum.Text=intsum.sum(a,b).ToString();
     }

Designer Page:

No 1<asp:TextBox ID="txt1" runat="server"></asp:TextBox><br />
No 2<asp:TextBox ID="txt2" runat="server"></asp:TextBox><br /><br />
<asp:Button ID="btnAdd" runat="server" Text="Add" onclick="btnAdd_Click" />  
<asp:Label ID="lblSum" runat="server"></asp:Label>
```

Output Would be:



Advanced .Net Concepts

AJAX controls with Update Panel

- AJAX is asynchronous Java Script and XML
- AJAX is not a programming language for developing websites.
- It is a collection of technologies to refresh the highly dynamic page according to page update approach not according to page replacement approach.
- AJAX controls are very useful because in a highly loaded graphical web page to refresh the content of small control, we have to reload the whole page, it took too much time.
- As well as tedious for user to use such web sites.
- But AJAX controls are capable that they only update the data of required contents only. Not the whole page.
- So page is quick updated.
- And easy to use graphically loaded web pages for highly dynamic interactions.

Script manager Control

- It helps to implement AJAX functionalities in Asp.Net website.
- On which page you want AJAX functionalities you have to add Script manager control.
- It is responsible for managing client scripts for AJAX enabled website.
- Without Script Manager Control AJAX functionalities are useless.

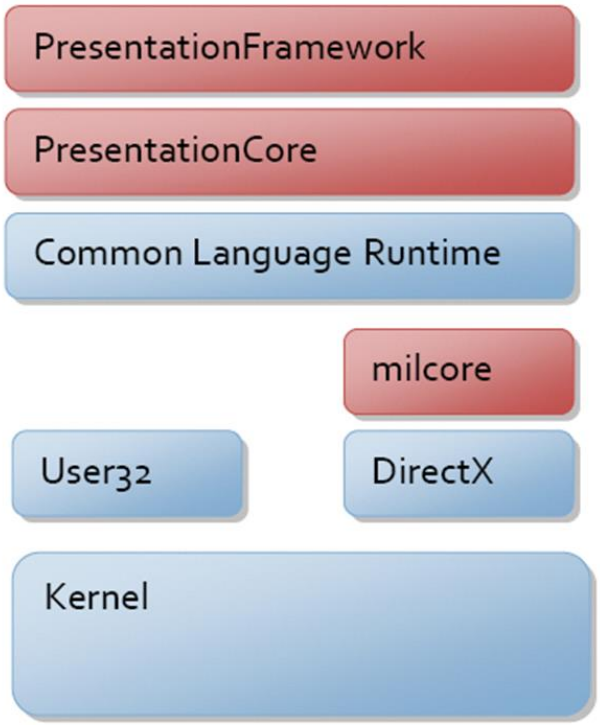
Update Panel Control

```
<form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server" />
   <asp:UpdatePanel runat="server" id="UpdatePanel" updatemode="Conditional">
   <Triggers>
   <asp:AsyncPostBackTrigger controlid="UpdateButton2" eventname="Click" />
   </Triggers>
   <ContentTemplate>
   <asp:Label runat="server" id="DateTimeLabel1" />
    <asp:Button runat="server" id="UpdateButton1" onclick="UpdateButton_Click" text="Update" />
        </ContentTemplate>
    </asp:UpdatePanel>
    <asp:UpdatePanel runat="server" id="UpdatePanel1" updatemode="Conditional">
     <ContentTemplate>
     <asp:Label runat="server" id="DateTimeLabel2" />
     <asp:Button runat="server" id="UpdateButton2" onclick="UpdateButton_Click" text="Update" />
        </ContentTemplate>
     </asp:UpdatePanel>
  </form>
```

```
Code Behind File
protected void UpdateButton_Click(object sender, EventArgs e)
{
   DateTimeLabel1.Text = DateTime.Now.ToString();
   DateTimeLabel2.Text = DateTime.Now.ToString();}
```

Explain WPF with its features

- Windows Presentation Foundation (WPF) is a means for the programmer to create Windows applications possessing rich user interfaces and graphics which the classic .NET Windows applications lack. The initial version of WPF was released as a part of .NET 3.0 and it was really like a preview of WPF itself. The actual version of WPF was released as a part of .NET Framework 3.5.
- High level architecture of WPF:



Features of WPF

- **Declarative Programming**
  WPF application paves the way for the developers to define the thick client UI in a declarative way, which was never supported by the traditional .NET windows forms. Tasks like defining a template for a control, creating a control hierarchy and similar work would be much easier if it is done in a declarative fashion. In

WPF declarative programming was made possible with the introduction of Extensible Application Markup Language (XAML). It can be compared to the HTML part in a web user interface.

- **Independent of screen resolution**
  This is a neat feature of WPF. What I mean by independency of screen resolution is, the WPF user interface will look better even on a screen with low resolution. It uses DirectX components whereas the Windows Forms applications make use of the User 32 components of a machine. WPF framework has the Media Integration Layer (MIL) in order to talk to the DirectX components. The direct components impose a vector based graphics on the WPF user interface. The below screen shot will show you the difference between the Windows forms UI and WPF UI at lower resolutions of the screen.
- **Control Inside a control**
  WPF allows you to provide not only the text but it also allows you to define a control as a content of another basic control like a Button. This feature is truly astonishing fact for the developers and this lets the world know, what the power of WPF is when it comes to user interfaces. You can have a Text Box inside a Button for example.
- **Control transforms**
  - WPF contains a handful of 2D transforms which will enable you to change the size, position, rotation angle and also allows skewing. Control transforms can be performed in two waysLayoutTransform and RenderTransform.
  - Rotate transform
  - Scale transform
  - Skew transform
  - Translate transform
  - Matrix transformation
- **Control Templates**
  What if the user wants to change the shape of a button, this will definitely sound weird for .NET developers. Now it can be done in WPF by defining the control template. For example you can declare a Button on your WPF window and can change its shape to elliptical.
- **2D, 3D, graphics, animation and Media**
  This is a vast topic in WPF, in my opinion these features make WPF a much more unique technology which mixes both controls and graphics to be tightly coupled. I will be touching some key features under this topic.

Explain WCF and simple Example

- Windows Communication Foundation (WCF) is a framework for building service-oriented applications.
- Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application.
- An endpoint can be a client of a service that requests data from a service endpoint.

- The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:
    - A secure service to process business transactions.
    - A service that supplies current data to others, such as a traffic report or other monitoring service.
    - A chat service that allows two people to communicate or exchange data in real time.
    - A dashboard application that polls one or more services for data and presents it in a logical presentation.
    - Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
    - A Silver light application to poll a service for the latest data feeds.
    - In summary, WCF is designed to offer a manageable approach to creating Web services and Web service clients.

Features of WCF

- Service Orientation
- Interoperability
- Multi message patterns
- Service Metadata and data contracts
- Security
- Durable messages
- Transactions
- AJAX and REST support
- Extensibility
- **Steps to follow:**

In Visual Studio New Project → WCF → WCF Service library
Give name as myLib and click OK.
Add Service1.vb and Iservice1.vb Files to project.

Now Right Click on project → Add reference
Select System.ServiceModel

Then open Service1.vb

```
Imports System
Imports System.ServiceModel
Public Class Service1
    Implements IService1
    Public Function GetName(ByVal value As String) As String Implements IService1.GetData
      Console.WriteLine("Hello {0}", value)
      Return String.Format("Hello {0}", value)
   End Function
 End Class
```

Then open Iservice1.vb

```
<ServiceContract()>
Public Interface IService1

    <OperationContract()>
    Function GetData(ByVal value As String) As String

End Interface
```

Now host and run service

File → Add new Project
Select Console Application Name → WcHost
Add Module1.vb
Right click on WcHost and Add reference.

```
Imports System
Imports System.ServiceModel
Imports System.ServiceModel.Description
Imports myLib.Service1

Module Module1

    Sub Main()
        ' Step 1 Create a URI to serve as the base address
        Dim baseAddress As New Uri("http://localhost:8000/SayHelloService")

        ' Step 2 Create a ServiceHost instance
        Dim selfHost As New ServiceHost(GetType(myLib.Service1), baseAddress)
        Try

            ' Step 3 Add a service endpoint
            ' Add a service endpoint
            selfHost.AddServiceEndpoint(GetType(myLib.IService1), _
            New WSHttpBinding(), "HelloService")

            ' Step 4 Enable metadata exchange.
            Dim smb As New ServiceMetadataBehavior()
            smb.HttpGetEnabled = True
            selfHost.Description.Behaviors.Add(smb)

            ' Step 5 Start the service
            selfHost.Open()
            Console.WriteLine("The service is ready.")
            Console.WriteLine("Press <ENTER> to terminate service.")
            Console.WriteLine()
            Console.ReadLine()
```

```
        ' Close the ServiceHostBase to shutdown the service.
        selfHost.Close()
      Catch ce As CommunicationException
        Console.WriteLine("An exception occurred: {0}", ce.Message)
        selfHost.Abort()
      End Try
    End Sub
End Module
```

Build WcHost.

Create console application → WcClient
Add Reference→ System. ServiceModel

Add Service reference →

Put "http://localhost:8000/SayHelloService" in address.
Now run wcHost.exe, you can locate it in ..\wcHost\bin\Debug folder.
When wcHost runs, then click go in add service reference window and wait.

Select Service 1 and Click Ok

Configure WCF Client

Open WcClient app.config from solution explorer.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section defines the logging configuration for My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
          <!-- Uncomment the below section to write to the Application Event Log -->
          <!--<add name="EventLog"/>-->
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
```

```
            type="Microsoft.VisualBasic.Logging.FileLogTraceListener, Microsoft.VisualBasic, Version=8.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL"
            initializeData="FileLogWriter"/>
        <!-- Uncomment the below section and replace APPLICATION_NAME with the name of your application to
write to the Application Event Log -->
        <!--<add name="EventLog" type="System.Diagnostics.EventLogTraceListener"
initializeData="APPLICATION_NAME"/> -->
      </sharedListeners>
   </system.diagnostics>
   <system.serviceModel>
      <bindings>
         <wsHttpBinding>
            <binding name="WSHttpBinding_IService1" />
         </wsHttpBinding>
      </bindings>
      <client>
         <endpoint address="http://localhost:8000/SayHelloService/HelloService"
            binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_IService1"
            contract="ServiceReference1.IService1" name="WSHttpBinding_IService1">
            <identity>
               <userPrincipalName value="ERP001\Manager1" />
            </identity>
         </endpoint>
      </client>
   </system.serviceModel>
</configuration>
```

In Moduel1.vb

```
Imports System
Imports System.ServiceModel
Imports wcClient.ServiceReference1
Module Module1
   Dim Client As New Service1Client
   Sub Main()
      Console.WriteLine("Client Window")
      Console.WriteLine("{0}", Client.GetData(Console.ReadLine()))
      ' Step 3: Closing the client gracefully closes the connection and cleans up resources.
      Client.Close()
      Console.WriteLine()
      Console.WriteLine("Press <ENTER> to terminate client.")
      Console.ReadLine()
   End Sub
End Module
```

Build And Run WcClient

Silverlight in ASP.Net

- Silver light is a web based technology that allows web designers and developers to stretch the boundaries of web application development.
- It is an integration of the rich user interface of desktop applications and the transparency of other web languages, such as html and java script.
- Silver light allows you to develop web applications that contain high-fidelity multimedia content and eye catching visual effects.
- The primary components of silver light include HTML, Java script, XAML and .net Framework.
- While developing silver light applications you will be using the combination of varied features of these technologies.
- It has two main features:
    - .net framework for silver light
    - Core presentation framework

.Net framework for silver light

- Silver light comes with a smaller version of .net framework.
- The .net frame work in silver light component also enables you to easily access data and services at remote locations through WCF.
- In addition, you can use HTTP objects, cross-domain HTTP requests, Really simple Syndication (RSS) feeds, JSON and SOAP services in Silver light application.

Core Presentation Framework

- It is a component in Silver light that offers the features and services for designing the UI of silver light applications through controls, styles, templates, 2-d vectors graphics, animations and multimedia.

- This component also allows you to incorporate user input and interactivity through various input devices.
- Moreover you can also manage the digital rights of multimedia contents used in silver light application through this component.

Silver light controls

- The controls in silver light correspond to different XAML elements.
- E.g Border, button, Canvas, Content Control, Data Grid, data Picker, Image etc.