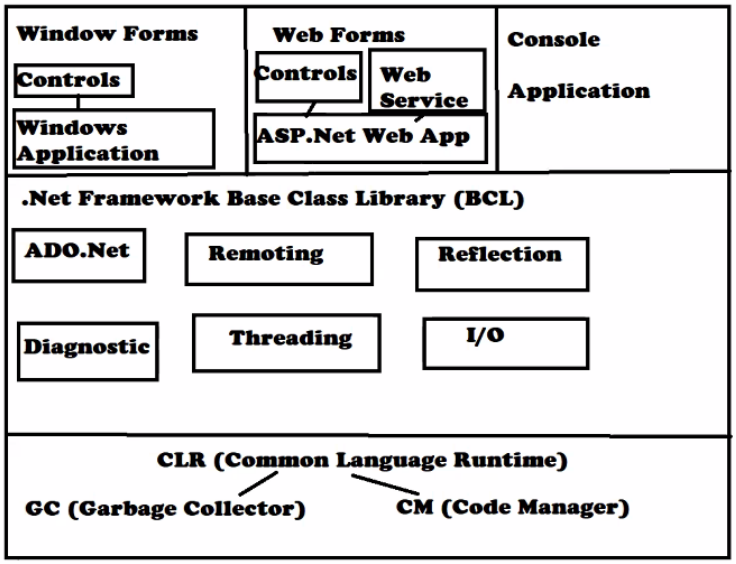
DOT NET

Dot net framework -

Dot net framework provides a set of base class libraries, which provides functions and features that can be used with any programming language such as visual basic, C#, J#, etc.



- **ASP -** Active server pages
- **ADO -** ActiveX data Object
- **Xamarin -** for mobile develop.

System - it contains fundamentals for programming such as datatype, console, array, etc.

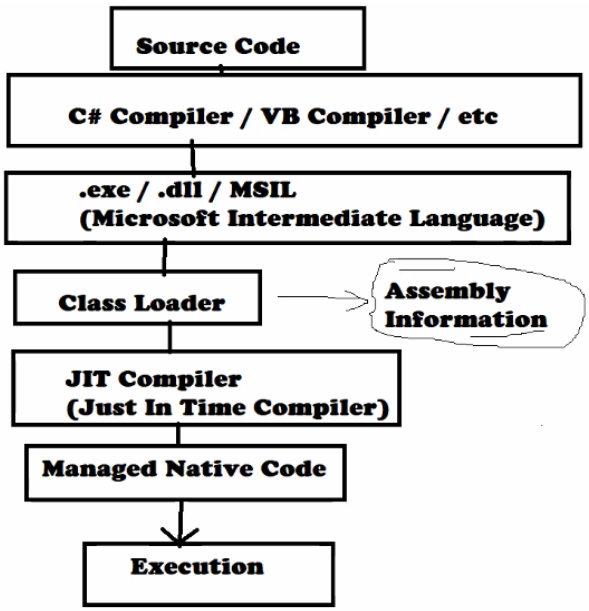
System is a class can be found in system.dll

System.Diagnostic - it provides trracing, logging, performance counter ,etc.

System.IO - It provides a connection to file system for reading and writing to data streams.

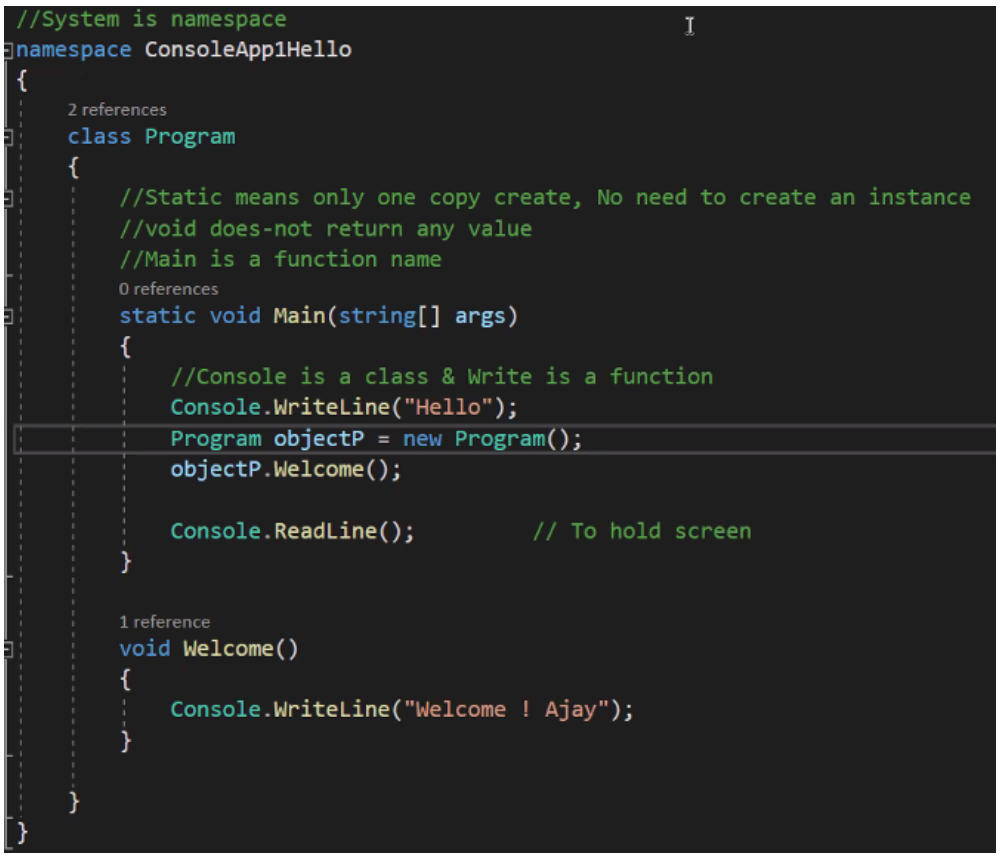
- **System.Threading -** it contains a method to manage the creation, synchronization, and pooling of program threads.
- **System.remoting -** for communication between objects which are not in same process.
- **ADO.Net -** it is an object-oriented set of libraries that allows you to interact with data sources, used to handle data access.
- **CLR -** common language runtime, provides an environment in which a program is executed, it activates the object, performs security checks on them, lay them out in the memory, executes them and garbage is collect. Code compilation in presence of CLR.
- **CTS -** (under CLR) a (common type system) is a rich type system, build into CLR, which supports the types and operations found in most programming languages. It provides for convert type, it consists of all types supported by .Net like bool, integer, string etc.

- **CLS** (under CLR)- (Common Language Specification) Is a set of constructs and constraints that serves as a guide for library writers and compiler writers.
    - All compilers user .Net will generate **LT** (Intermediate Language).
    - CLS is a set of rules or guidelines which if followed ensure that code written in one language can be used by another .Net language.



Namespace & Assembly - Namespace is a collection of different classes, whereas assembly is a basic building block of .Net framework. It is a logical group of functionality in a physical file. (Contains meta dada & .dll)

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

//System is namespace
namespace ConsoleApp1Hello
{
    0 references
    class Program
    {
        //Static means only one copy create, No need to create an instance
        //void does-not return any value
        //Main is a function name
        0 references
        static void Main(string[] args)
        {
            //Console is a class & Write is a function
            Console.Write("Hello");


            Console.ReadLine();         // To hold screen

        }
    }
}
```



```
//System is namespace
namespace ConsoleApp1Hello
{
    2 references
    class Program
    {
        //Static means only one copy create, No need to create an instance
        //void does-not return any value
        //Main is a function name
        0 references
        static void Main(string[] args)
        {
            //Console is a class & Write is a function
            Console.WriteLine("Hello");
            Program objectP = new Program();
            objectP.Welcome();

            Console.ReadLine();         // To hold screen
        }

        1 reference
        void Welcome()
        {
            Console.WriteLine("Welcome ! Ajay");
        }

    }
}
```

Types of JIT compiler -

1. **Pre JIT -** it complies with MSIL code in a single compilation. This is done at a time of deployment of an application.
2. **Econo JIT -** It complies only MSIL code of those methods that are called at runtime and removes them from memory after execution.
3. **Normal JIT -** it complies only MSIL code of those methods that are called at runtime and that converted native code is stored in cache (recalls many times).

#Access Specifier / Modifiers -

Access specifiers are the keywords used to define an accessibility level of all types and members.

1. **Public -** It is used to specify that access is not restricted.
2. **Private -** It is used to specify that access is limited to containing class or any type.

3. **Protected -** It is used to specify that access is limited to the containing type or types defined from the containing class. It is used only at the inheritance level.
    a. Eg - Inheritance - super class(parent) and sub class(child).
4. **Internal -** It is used to specify that access is limited to the current assembly.
    a. Available till current assembly. (current project).

Basics of VS -

- **To create new project -** Language (C#) and Project type (Console)
- **How to add a reference project -** Right-click on solution > add > New Project
- **How to add new class in a project** (ctr + shift + A)- Right-click on project name > add > New Itom > class.
- **Create dll -** Right click on solution > add > New Project > Select project - Class Library (.net framework).
    - Delete the existing class called Class1.cs > create new class.
    - At top build > Build solution (ctlr + shift + B);
    - to check for dll path - right-click on dll project file > Open file in file explorer.
    - The path will be like this - D:\Applications\VS_Projects\Practice\DLLTest\bin\Debug\net6.0 > dllFileName.dll
    - To add dll file in our project file - right-click on references under the project > add references > Browse > paste the path of dll file > add.

Add dll to current project.

Conditional Statements -

Press F10 to debug our code line by line.

If statement -

If statement is used to execute a block of code or statements when the defined condition is true.

```
//Conditional Statements - If statement.
int x = 20, y = 15;
if (x >= 15)
{
    Console.WriteLine("x is greater than 15");
}
if (y <= 10)
{
    Console.WriteLine("y is less than 10");
}
Console.WriteLine("Press enter key to exit..");
Console.Read();
```

Else statement -

Else statement will be executed when if the condition fails to execute.

```
Console.Write("Enter First Number : ");
int x = Convert.ToInt32( Console.ReadLine());

Console.WriteLine("Enter Second Number : ");
int y = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter Third Number : ");
int z = Convert.ToInt32(Console.ReadLine());

if (x > y && x > z)
{
    Console.WriteLine("{0} is greater than {1} and {2}", x, y, z);
}
else if (y > z)
{
    Console.WriteLine("{1} is greater than {0} and {2}", x, y, z);
}
else
    Console.WriteLine("{2} is greater than {0} and {1}", x, y, z);

Console.WriteLine("Press enter key to exit..");
Console.Read();
```

Switch statement -

```
Console.Write("Enter First Number : ");
int x = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Second Number : ");
int y = Convert.ToInt32(Console.ReadLine());

Console.WriteLine();
Console.WriteLine("Switch-Case : Calculation Program == ");
Console.WriteLine("1 . Addition");
Console.WriteLine("2 . Substraction");
Console.WriteLine("3 . Multiplication");
Console.WriteLine("4 . Division");

Console.Write("Enter your choice : ");
int choice = Convert.ToInt32(Console.ReadLine());
```

```
switch (choice)
{
    case 1:
        Console.WriteLine("Addition is : " + (x + y));
        break;
    case 2:
        Console.WriteLine("Substraction is : " + (x - y));
        break;
    case 3:
        Console.WriteLine("Multiplication is : " + (x * y));
        break;
    case 4:
        Console.WriteLine("Division is : " + (x / y));
        break;
    default:
        Console.WriteLine("Invalid choice");
        break;
}

Console.WriteLine("Press enter key to exit..");
Console.Read();
```

Looping statements -

is used to execute a repetitive statement.

while loop -

it is used tp execute a block of statements until the specified condition return as true. In this condition will be checked first.

Whereas **do-while** lop is same as while loop just difference is that in this condition will be checked at last of the statement.

```
int number = 4;
int i = 1;
while (i <= 10)
{
    //Console.WriteLine(number + " * " + i + " = " + (number * i));

    Console.WriteLine("{0} * {1} = {2}", number, i, (number * i));
    i++;//i=i+1;//it will print the table of 4.
}
```

```
//do-while
int numForWhile = 5;
int i = 1;
do
{
    //Console.WriteLine(number + " * " + i + " = " + (number * i));
    Console.WriteLine("{0} * {1} = {2}", numForWhile, i, (numForWhile * i));
    i++;//i=i+1;
} while (i <= 10);//will print the table of 5.
```

For loop -

for loop is used to execute a statement repeatedly until the defined condition returns true. In this statement initialization, condition checking and increment/decrement are on single line of statement.

```
public void createTableFor()
{
    //for loop is used to execute a statement repeatedly.
    Console.Write("Enter any number to print table: ");
    int number = Convert.ToInt32(Console.ReadLine());
    for(int i = 1;i<=10;i++)//will print the table of number.
    {
        Console.WriteLine("{0} * {1} = {2}", number, i, (number * i));
    }
    Console.WriteLine("Press Enter Key To exit");
    Console.ReadLine();
}
```

For each loop -

for-each loop is useful to execute a statement repeatedly on a collection of objects. Such as array, list, etc.

```
1 reference
public void ForEachLoop()
{
    string[] studentNames = new string[3] { "sumit", "sam", "mat" };

    foreach(string name in studentNames)
    {
        Console.WriteLine(name);
    }//OP - sumit sam mat.
    Console.WriteLine("Press Enter Key To exit");
    Console.ReadLine();
}
```

Array -

Array is a collection of similar type of elements.

Call by value and call by reference -

There are 2 types of parameters that can be passed in function 1. Value Type & 2. Ref Type. means 2 ways of allocating space in memory.

- **Value Type -** A value type variable directly contains data in memory
- **Ref Type -** A ref type variable contains the memory address of a value.

When we create object of a particular class with new keyword, a space is created in the managed heap, that holds the reference of class.

```
static void Main(string[] args)
{
    int number = 6;
    /*//call by value.
    int result = Program.cube(number);
    Console.WriteLine(result);*///216

    //call by referance.
    Program.cube1(ref number);
    Console.WriteLine(number);//216
  //the value will be returned into number.

    Console.WriteLine("Press enter key to exit..");
    Console.Read();
}
0 references
public static int cube(int n)
{
    return n * n * n;
    //output will be returned
}
1 reference
public static void cube1(ref int n)
{
  n = n * n * n;
    //number local variable will be manipulated.
}
```

Out keyword -

Out keyword is used for the passing the arguments to methods as a reference type. It is generally used when a method has multiple values, **out** parameter doesn't pass the property.

- It is necessary to initialize the parameter before it passes to **out.**
- It is necessary to initialize the value of the parameter before returning to calling method.
- It is declared with **out** keyword.

```
0 references
static void Main(string[] args)
{
    int a, p;
    Program.Rectangle(4,5,out a,out p);
    Console.WriteLine("Area is : " + a);
    Console.WriteLine("Parameter is : " + p);

    Console.WriteLine("Press enter key to exit..");
    Console.Read();
}
1 reference
public static void Rectangle(int l, int w, out int area, out int parameter)
{
   /* we just received the values with l and w of a and p.
    and we are returning 2 arguments with out keyword.*/

    area = l * w;//have to return area and parameter (with out)
    parameter = 2 * (l + w);
}
```

Constructor -

Value initialize, no return type, always be **public**, same name of a class name. It get called automatically when object is created and initialized.

```
static string name;
1 reference
public Program()
{
    Console.WriteLine("Default Constructor");
    name = "Ram Kapoor";
}
0 references
static void Main(string[] args)
{
    Program pObject = new Program();//Constructor will be called.
    Console.WriteLine(Program.name);//Name string will be called.

    Console.WriteLine("Press enter key to exit..");
    Console.Read();
}
```

Parameterized constructor -

```
0 references
static void Main(String[] args)
{
    Program obj = new Program("sumit");

    Console.WriteLine("Press enter key to exit..");
    Console.Read();
```

```
static string name;
0 references
Program()       //default constructor
{
    Console.WriteLine("Default Constructor");
    name = "Ram Kapoor";
}

1 reference
Program(string username)        //parametrize constructor
{
    Console.WriteLine("parametrize Constructor");
    name = username;
    Console.WriteLine(name);
}
```

OP -

```
parametrize Constructor
sumit
Press enter key to exit..
```

Distructor -

Destructor is used in class to destroy the object or instance of class. Destructor will be called automatically whenever the class object become unreachable. Destructor same name as class name. Represented by(~)tilde sign. It is not accepting any parameter.

```
static string name;
0 references
Program()       //default constructor
{
    Console.WriteLine("Default Constructor");
    name = "Ram Kapoor";
}

~Program()  //Distructor.
{
    Console.WriteLine("Destrucor Called");
}

1 reference
Program(string username)        //parametrize constructor
{
    Console.WriteLine("parametrize Constructor");
    name = username;
    Console.WriteLine(name);
}
```

```
parametrize Constructor
sumit
Press enter key to exit..

Destrucor Called
Press any key to continue . . .
```

Distructor will be called automatically.

Inheritance -

Inheritance is used to inherit the properties from one class(base) to another (derived) class.

Get and Set (Accessor and Modifier) -

- Get Set accessor or modifier mostly usd for storing and retrieving value from the private field.
- The get accessor must return a value of property type where Set accessor returns void.
- Set accessor uses an implicit parameter called value.

```
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        /*string[] str = new string[3] { "A", "B", "C" };
        Console.WriteLine(str.Length);*/

        PropertyEx obj = new PropertyEx();//object
        obj.PropertyName = "Sumit";
        obj.print();//calling print method

        Console.WriteLine("Property value " + obj.PropertyName);
        Console.WriteLine("Press enter key to exit..");
        Console.Read();
    }
}
```

```
class PropertyEx
{
    private static string name;

    2 references
    public string PropertyName //property.
    {
        get { return name; }
        set { name = value; }
    }
    1 reference
    public void print()
    {
        Console.WriteLine("Private Value " + name);
    }
}
```

```
Private Value Sumit
Property value Sumit
Press enter key to exit..
```

Inheritance -

Inheritance is used to inherit the properties from the base (parent) class to the derived (child) class.

Types of inheritance -

1. **Single level inheritance -** One base class and one derived class.
2. **Multi-level inheritance -** class can inherit from one base class and this chain process is goes on.
3. **Multiple inheritance -** Not achievable because of ambiguity error.
4. **Hierarchical inheritance -** one base class and more than one derived class.

1. Inheritance Example single level - base class -

```
public class Student
{
    public string Name;//Data member
    public int RollNumber;

    1 reference
    public void GetStudent(int studentId)
    {
        RollNumber = studentId;//assigning to Rollnumber
        Console.WriteLine("Base class");
        Console.WriteLine("Sir Name is : " + Name);
        Console.WriteLine("Student Roll Number is : "+RollNumber);
    }
}
```

Child class -

```
public class StudentDetails : Student//inherited base class.
{
    public string city;//data member
    1 reference
    public void GetCity()
    {
        Console.WriteLine("Derived class class");
        Console.WriteLine("Student City : " +city);
    }
}
```

Main class -

```
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        StudentDetails derivedobj = new StudentDetails();
        derivedobj.Name = "Ajay Sir";
        derivedobj.city = "Indore";
        derivedobj.GetStudent(1011);//calling base class methods.
        derivedobj.GetCity();
    }
```

OP -

```
Base class
Sir Name is : Ajay Sir
Student Roll Number is : 1011
Derived class class
Student City : Indore
Press any key to continue . . .
```

2. Multi-Level Inheritance - base class

```
public class A
{
    public string Name;

public A()
{
    Console.WriteLine("Base Class Const A");
}

    public void GetName()
    {
        Console.WriteLine("Base Class A");
        Console.WriteLine("Student name : " + Name);
    }
}
```

Child class B -

```
public class B : A
{
    public int RollNumber;

    public B()
    {
    Console.WriteLine("Derived Class Const B");
    }

    public void GetRollNumber()
    {
        Console.WriteLine("Derived Class B");
        Console.WriteLine("Student Roll Number : " + RollNumber);
    }
}
```

Child class C -

```
public class C : B
{

    public C()
    {
        Console.WriteLine("Derived Class Const C");
    }
    public string City;

    public void GetCity()
    {
        Console.WriteLine("Student City : " + City);
    }
}
```

Main method -

```
namespace MultiLevelInherit
{

    internal class Program
    {

        static void Main(string[] args)
        {
            C cObject = new C();
            cObject.Name = "Ajay Sharma";
            cObject.RollNumber = 1011;
            cObject.City = "Indore";
            cObject.GetName();
            cObject.GetRollNumber();
            cObject.GetCity();

            Console.WriteLine("Press enter key to exit...");
            Console.Read();
        }
    }
}
```

OP -

```
Base Class Const A
Derived Class Const B
Derived Class Const C
Base Class A
Student name : Ajay Sharma
Derived Class B
Student Roll Number : 1011
Student City : Indore
Press enter key to exit...
```

Method overriding -

It means overrid a base class method in derived class. (means same name of method used in base and child class). It is achieved using override and virtual keyword.

- Virtual keyword - used when method overriding. Used with base class method.
- Override keyword - used when method overriding. Used with child class method.
    - Both keywords are used to call base method implicitly.

base ( ) Keyword -

```
namespace MethodOverriding
{
    0 references
    internal class Program
    {
        3 references
        public class A
        {
            4 references
            public virtual void GetName()
            {
                Console.WriteLine("Base class : Username ");
            }
        }
        2 references
        public class B : A
        {
            4 references
            public override void GetName()
            {
                base.GetName();
                Console.WriteLine("Derived class : Username ");
            }
        }
        0 references
```

```
        0 references
        static void Main(string[] args)
        {
            A obj = new A();
            obj.GetName();

            B bObj = new B();
            bObj.GetName();
        }
    }
```

```
Base class : Username
Base class : Username
Derived class : Username
Press any key to continue . .
```

Interface -

It is same as class but it only contains declaration of methods. C#not supports multiple inheritance, we can achieve it by interface. We can define interface by interface keyword. Only public method are allowed to declare, no other access specifiers allowed. It cannot be instantiated direclty.

Interface Name and Interface City -

We can only declare methods in interface and define methods in inherited class. In this ex class User.

```
namespace InterfaceExample
{
    1 reference
    internal interface Name
    {
        2 references
        void GetName(string name);
    }
}
```

```
namespace InterfaceExample
{
    1 reference
    internal interface City
    {
        2 references
        void GetCity(string city, string pincode);
    }
}
```

Creating a class to inherit both the interface -

```
namespace InterfaceExample
{
    2 references
    public class User : Name, City
    {
        2 references
        public void GetName(string name)
        {
            Console.WriteLine("Username : {0}", name);
        }

        2 references
        public void GetCity(string city, string pincode)
        {
            Console.WriteLine("City : {0}, Pincode : {1}", city, pincode);
        }
    }
```

Main Class -

```
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            User uObject = new User(); //Created obj of user class.
            uObject.GetName("Amar");
            uObject.GetCity("Indore","455001");

            Console.WriteLine("Press enter key to exit..");
            Console.Read();
        }
    }// OP - Username : Amar
          // City : Indore, Pincode : 455001
}
```

Abstract -

Abstract is a keyword. It is used to define classes, methods which is used to implement / override in derived (child) class.

Those methods we define as abstract or include an abstract class must be implement by class derived from abstract class.

Abstract methods are internally treated as virtual methods, that's why need to be use override method.

Abstract keyword ex -

```
abstract class Hello
{
    1 reference
    public void HelloWorld()
    {
        Console.WriteLine("Welcome");
    }
    //Abstract Class -
    2 references
    abstract public void FindArea(int x, int y);
}
2 references
class Rectangle : Hello //can't write public here.
{
    2 references
    public override void FindArea(int x, int y)
    {
        HelloWorld();
        Console.WriteLine("Area is : " + (x * y));
    }
}
```

```
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        Rectangle rectangle = new Rectangle();
        rectangle.FindArea(5, 6);
    }
}
```

```
Welcome
Area is : 30
Press any key to continue
```

Sealed keyword -

It is used to stop inheriting any specific class from other classes. For this we need to use **sealed** keyword**.**

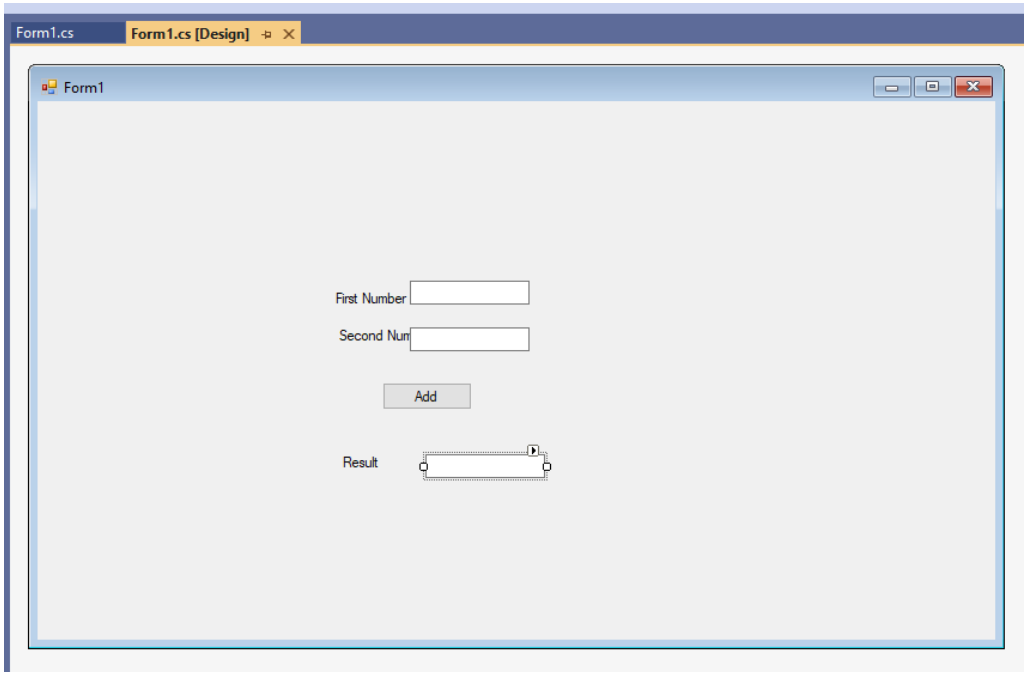
```
namespace Sealed_Keyword
{
    1 reference
    public sealed class Student
    {
        0 references
        public void GetName()
        {
            Console.WriteLine("Hello Ajay");
        }
    }
    2 references
    public class StudentCity : Student //error.
    { //error if we inherit sealed class.
        1 reference
        public void GetCity()
        {
            Console.WriteLine("City : Indore");
        }
    }
    0 references
```

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        StudentCity city = new StudentCity();
        city.GetCity();
    }
}
```

Error we cannot inherit a class having sealed keyword.

Windows Forms App -

```
namespace WindowsFormsAppCalculation
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }

        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        1 reference
        private void button1_Click(object sender, EventArgs e)
        {
            int fn = Convert.ToInt32(textBox1.Text);
            int sn = Convert.ToInt32(textBox2.Text);
            int responce = fn + sn;
            textBox3.Text = responce.ToString();//to string to convert into int

        }

        1 reference
        private void label3_Click(object sender, EventArgs e)
        {

        }
    }
}
```



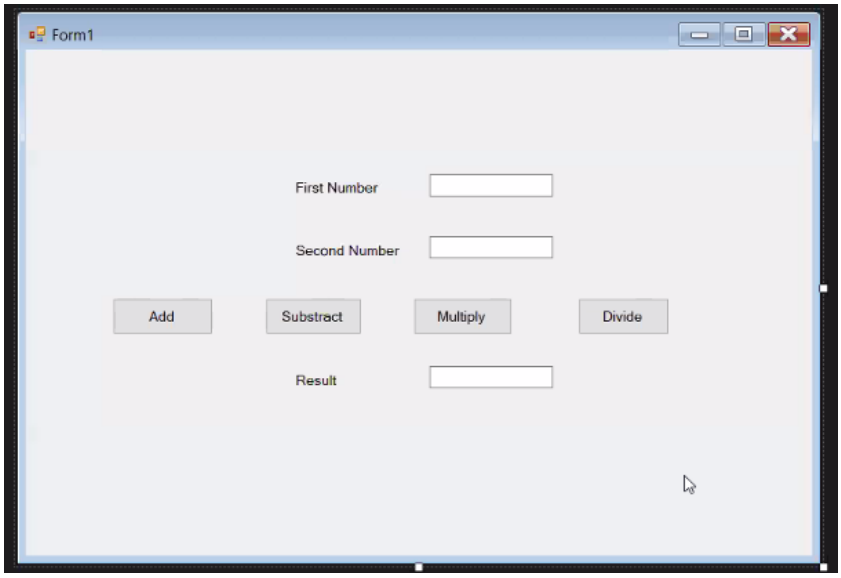
Form 2 -

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

    private void button1_Click(object sender, EventArgs e)
    {
        int fn = Convert.ToInt32(textBox1.Text);
        int sn = Convert.ToInt32(textBox2.Text);
        int response = fn + sn;
        textBox3.Text = response.ToString();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        int fn = Convert.ToInt32(textBox1.Text);
        int sn = Convert.ToInt32(textBox2.Text);
        int response = fn - sn;
        textBox3.Text = response.ToString();
    }
```

```
private void button3_Click(object sender, EventArgs e)
{
    int response = Convert.ToInt32(textBox1.Text) * Convert.ToInt32(textBox2.Text);
    textBox3.Text = response.ToString();
}
```

(local variable) int response

```
private void button4_Click(object sender, EventArgs e)
{
    textBox3.Text = (Convert.ToInt32(textBox1.Text) / Convert.ToInt32(textBox2.Text)).ToString();
}
```



Exception Handling -

Try catch block -

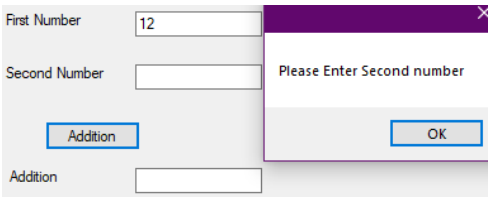
- try – A try block is used to encapsulate a region of code. If any code throws an exception within that try block, the exception will be handled by the corresponding catch.
- catch – When an exception occurs, the Catch block of code is executed. This is where you are able to handle the exception, log it, or ignore it.

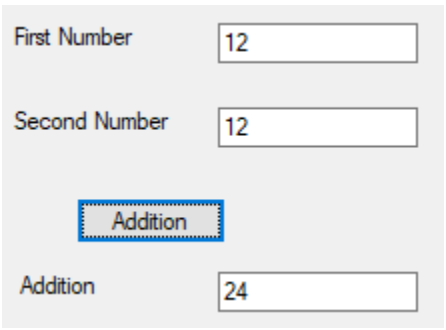
Creating a method to validate the error -

```
public partial class Form1 : Form
{
    //Exeption Handling
    3 references
    public string ValidateNumbers()
    {
        if (textBox1.Text == "")
            return "Please Enter first number";
        else if (textBox2.Text == "")
            return "Please Enter Second number";
        else
            return "success";
    }
    1 reference
    public Form1()
    {
        InitializeComponent();
    }
    1 reference
```

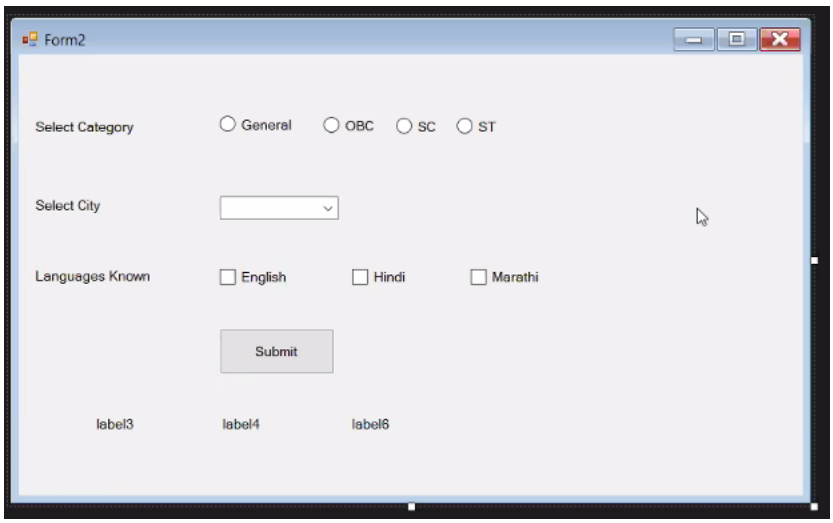
Assigning the above method into validateRespone and printing into textbox -

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        string validateResponce = ValidateNumbers();

        if (validateResponce == "success")
        {
            textBox3.Text = (Convert.ToInt32(textBox1.Text) + Convert.ToInt32(textBox2.Text)).ToString();
        }
        else
        {
            MessageBox.Show(validateResponce);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```





Form 2 -

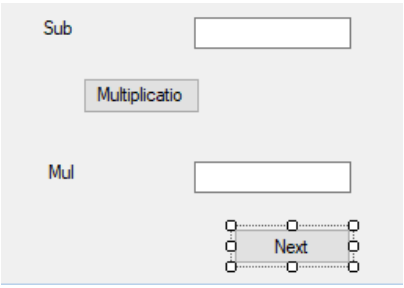


```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    label4.Text = "";
    label5.Text = "";
    label6.Text = "";

    if (radioButton1.Checked)
        label4.Text = radioButton1.Text;
    else if (radioButton2.Checked)
        label4.Text = radioButton2.Text;
    else if (radioButton3.Checked)
        label4.Text = radioButton3.Text;
    else if (radioButton4.Checked)
        label4.Text = radioButton4.Text;
    else
        MessageBox.Show("Please Select Category");
```

```
string languageSelected = "";
if (checkBox1.Checked)
    languageSelected = checkBox1.Text;
if (checkBox2.Checked)
    languageSelected += (languageSelected == "") ? checkBox2.Text : ", " + checkBox2.Text;
if (checkBox3.Checked)
    languageSelected += (languageSelected == "") ? checkBox3.Text : ", " + checkBox3.Text;

label6.Text = languageSelected;


if (checkBox1.Checked== false && checkBox2.Checked == false && checkBox3.Checked == false)
{
    label6.Text = "";
    MessageBox.Show("Please Select Language");
}
```

To go to form2 from form1 -

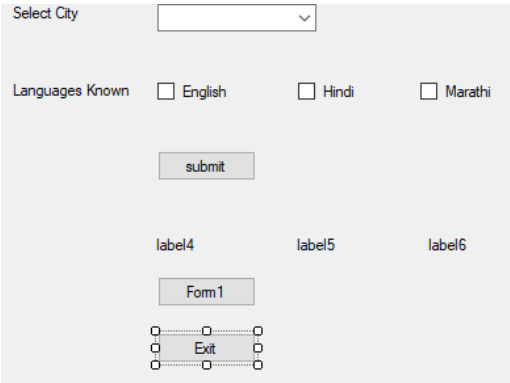
Adding one extra button at form1.




```
1 reference
private void button4_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    form2.Show();//to show form2.
    this.Hide();//to hide form1, not close.
}
```

After clicking on Next form 2 will open –

Form2 -



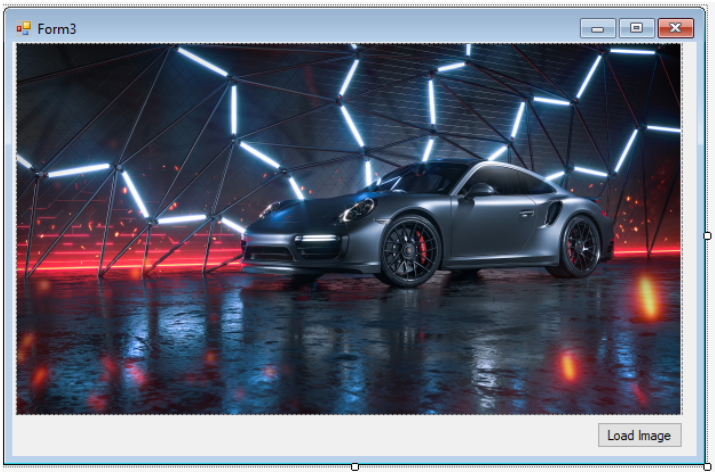
Added two buttons in the form 2 - form1 button will lead to form 1 and Exit will exit the running programm.

```
1 reference
private void button3_Click(object sender, EventArgs e)
{
    Form1 form1 = new Form1();
    form1.Show();
    this.Hide();
}
```

To exit -

```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

Adding image and loading image - Form 3 -



To load the image > picture box > properties > appearance > image > then select the image.

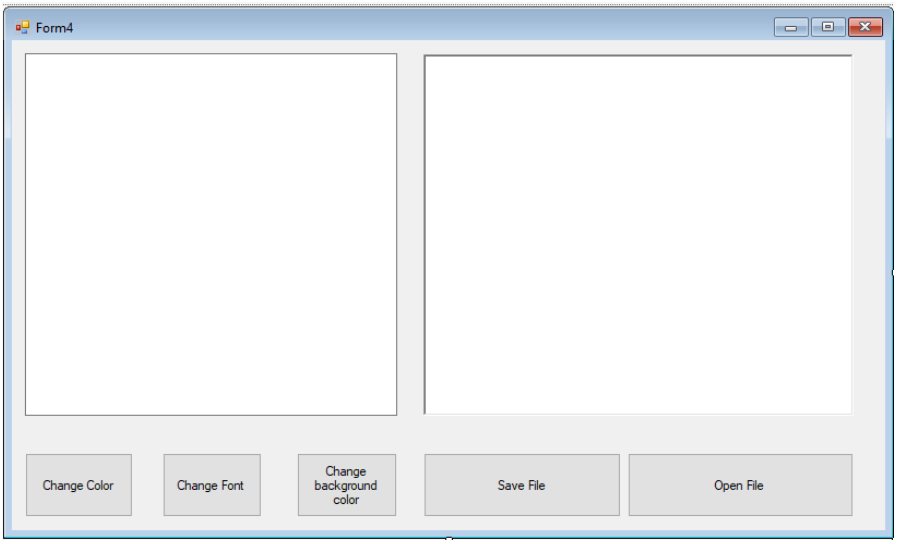
For loading the image with button -

```
private void button1_Click(object sender, EventArgs e)
{
    string imagePath = "C:\\Users\\sumit\\OneDrive\\Desktop\\jewellery2.png";
    pictureBox1.Image = Image.FromFile(imagePath);
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
}
```

To change the background color - double click on form3.

```
private void Form3_Load(object sender, EventArgs e)
{
    //MessageBox.Show("Welcome to Form3");
    this.BackColor = Color.LightBlue;
}
```

Form 4 -



Added - colorDialog, FontDialog, SaceFileDialog, opeFileDialog. And buttons.

Change color button -

```
private void button1_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
        textBox1.ForeColor = colorDialog1.Color;
}
```

Change font -

```
private void button2_Click(object sender, EventArgs e)
{
    /*fontDialog1.ShowDialog();*/
    if (fontDialog1.ShowDialog() == DialogResult.OK)
    textBox1.Font = fontDialog1.Font;
}
```

Change background -

```
private void button3_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
        textBox1.BackColor = colorDialog1.Color;
}
```

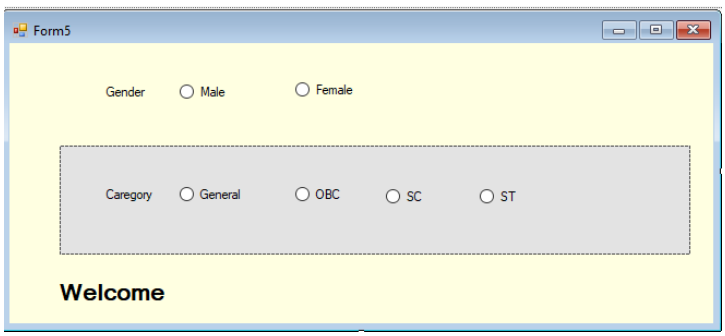
Save file -

```
private void button4_Click(object sender, EventArgs e)
{
    saveFileDialog1.ShowDialog();
    string fileName = saveFileDialog1.FileName;
    richTextBox1.SaveFile(fileName, RichTextBoxStreamType.PlainText);
}
```

Load File -

```
private void button5_Click(object sender, EventArgs e)
{
    openFileDialog1.ShowDialog();
    string filename = openFileDialog1.FileName;
    richTextBox1.LoadFile(filename, RichTextBoxStreamType.PlainText);
}
```

Form 5 -

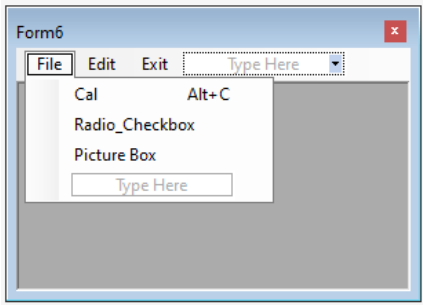


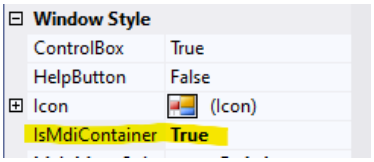
With the time on welcome and we can select multiple radio buttons with the panel.

```
private void timer1_Tick(object sender, EventArgs e)
{
    //this.BackColor = Color.LightSkyBlue;
    string colorName = this.BackColor.Name;
    if(colorName == Color.LightSkyBlue.Name)
    {
        this.BackColor = Color.LightSeaGreen;
    }
    else
    {
        this.BackColor= Color.LightSkyBlue;
    }
}
```

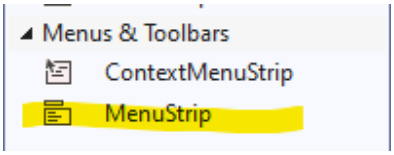
```
private void timer2_Tick(object sender, EventArgs e)
{
    label3.Left += label3.Left + 2;
}
```

Form 6 - with mdiContainer - true





To add menu buttons -



Under File -

```
private void calToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form1 form1 = new Form1();
    form1.MdiParent = this;
    form1.Show();
}
```

Added calculator from form1 and form from form5.

```
1 reference
private void dialogBoxToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form5 form5 = new Form5();
    form5.MdiParent = this;
    form5.Show();
}
```

```
private void pictureBoxToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form3 form3 = new Form3();
    form3.MdiParent = this;
    form3.Show();
}
```

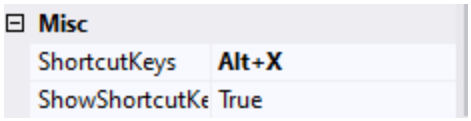
Added form3 features.

```
private void radioCheckboxToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    form2.MdiParent = this;
    form2.Show();
}
```

Added form2 featurs.

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

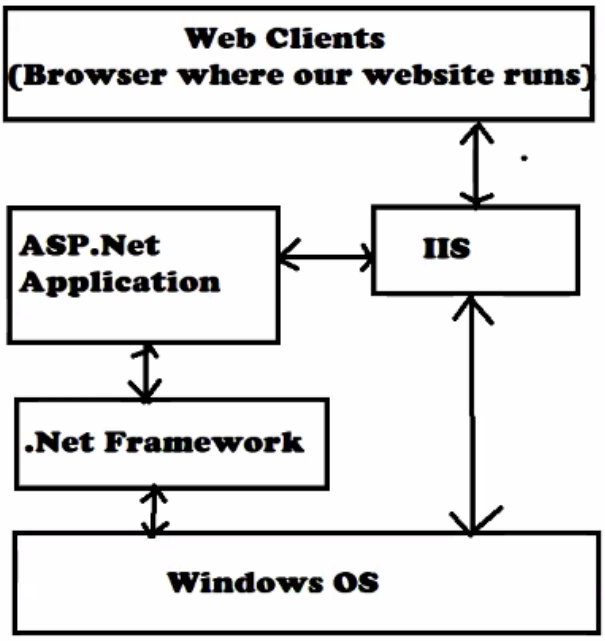
To exit the application.



Added shortcut to exit key.

ASP .NET -

ASP - Active Server Pages **IIS - Internet Information Services**



ASP.Net 3-tier architecture
- To make application more understandable
- Easy to maintain, easy to modify and we can maintain a good look of architecture
- It has 2 files : .cs (code behind file) & .aspx (html view)

It contains 3 layer :
1. Application or Presentation layer - Our aspx pages, & UI port

2. Business Access or Business logic layer -
It do all types of calculations & perform any type of work related with data like insert data, retrieve data & validating the data.

3. Data Access Layer - Database queries.

File -> new ➔ project ➔ ASP.NET web application(.NET Framework). After Empty project name ➔ click on create button. Select empty from list and from right side select MVC checkbox. And click on create button.

From solution explorer ➔ right click on controller folder, from the list go to add ➔ controller.

Now, form the popun select MVC5 controller - empty click on add button, give controller name, don't change "controller" only change default name then click on add.

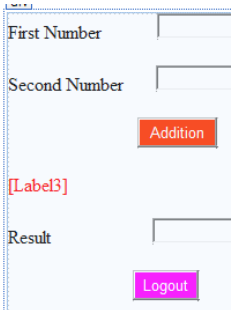
- If we want to create a view page - right-click on method name and from the open list click "Add view".
- If we want to create model (class) then right-click on the model folder and select class from the list. Give class name and click add.

To set as start page - Right click on project > set as start page.

Form 2 -


```
protected void Button1_Click(object sender, EventArgs e)
{
    Session["UserID"] = TextBox1.Text;
    Response.Redirect("WebForm1.aspx");
}
```

Form 1 -




```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["UserID"] == null)//it will validate the Text is empty or not.
        Response.Redirect("WebForm2.aspx");
    else if (Session["UserID"].ToString() == "" || Session["UserID"].ToString() == " ")
        Response.Redirect("Webform2.aspx");
}
```

For addition.

```
protected void Button1_Click(object sender, EventArgs e)
{
    try
    {
        TextBox3.Text = (Convert.ToInt32(TextBox1.Text) + Convert.ToInt32(TextBox2.Text)).ToString();
    }
    catch(Exception ex)
    {
      // Label3.Text = "ERROR";
       Label3.Text = ex.Message;
    }
}
```

For logout button.

```
protected void Button2_Click(object sender, EventArgs e)
{
    Session.Clear(); // it will clear the session.
    Response.Redirect("WebForm2.aspx");//redirect to form2
}
```

Xml -

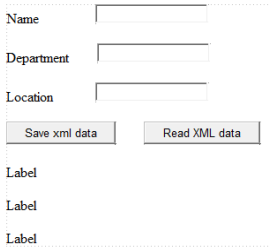
Xml - eXtensible Markup Language.

- It is designed to transport and store data, whereas HTML is used to display data and design web page.
- XML tags are not predefined. We must define our own tags.
- XML is designed to be self-descriptive.
- XML tags are case sensitive . eg <message>this is correct </message>.
- XML element can have attributes, just like html eg - <file type=”gif”> computer.gif</file>
    - Here file is an element and type is an attribute and gif is an attribute value.
- XML gives developers the power to deliver structured data from a wide variety of applications to windows or web-based presentations.

Some XML classes are -

1. XmlTextReader - it provides fast, non-cached, forward-only read access to xml data.
2. XmlTextWriter - It provides a fast, forward way of generating XML.

Namespace - using system.XML.



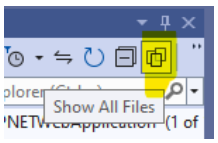

```
protected void Button1_Click(object sender, EventArgs e) //Save button.
{
    XmlTextWriter xWriter = new XmlTextWriter(Server.MapPath("Employee.xml"), Encoding.UTF8);
    xWriter.WriteStartDocument();//xWriter is an object of XmlWriter class. to write
    xWriter.WriteStartElement("EmployeeDetails");//A tag will be created named EmpolyeeDetails.
    //And file will be saved as Employee.xml.
    xWriter.WriteElementString("Name", TextBox1.Text);
    xWriter.WriteElementString("Department", TextBox2.Text);
    xWriter.WriteElementString("Location", TextBox3.Text);

    xWriter.WriteEndElement();
    xWriter.WriteEndDocument();
    xWriter.Close();

    Label4.Text = "Data Saved Successfully";
    TextBox1.Text = "";
    TextBox2.Text = "";
    TextBox3.Text = "";
}
```

```
protected void Button2_Click(object sender, EventArgs e)
{
    string name = "";
    string department = "";
    string location = "";
    string Elementname = "";

    XmlTextReader xReader = new XmlTextReader(Server.MapPath("Employee.xml"));

    while (xReader.Read())
    {
        if(xReader.NodeType == XmlNodeType.Element)
        {
            Elementname = xReader.Name;
        }
        else if (xReader.NodeType == XmlNodeType.Text)
        {
            if(Elementname == "Name")
            {
                name = xReader.Value;
            }
            if (Elementname == "Department")
            {
                department = xReader.Value;
            }
            if (Elementname == "Location")
            {
                location = xReader.Value;
            }
        }
    }
```

After a while loop -

```
xReader.Close();
Label4.Text = name;
Label5.Text = department;
Label6.Text = location;

TextBox1.Text = name;
TextBox2.Text = department;
TextBox3.Text = location;
```

To show all files -



Web Service -

It is a communication platform between two different or same platform applications that allows you to use their web method. This means, we can create a web service in any language and can be called to same or different language application. Web service always starts with [WebMethod] attribute means it is a web method. Its extension is .asmx (Active server Method Files).

WSDL (Web Service Description Language) - It is an XML-based language for describing web services and how to access then. WSDL is an XML document. It specifies the location of the service and the operation of the service exposes.

<definations>

<types>**data type definitions**</types>
<message>**defination of the data being communicated**</message>
<port type>**set of operations**</port type>
<dinding>**protocol and data format**</binding>

</definations>

SOAP - (Simple Object Access Protocol) is a remote function call that invokes methods and executes them on the remote machine and translates the object communication into XML format. SOAP is the way by which method calls are translated into XML format and sent via HTTP.

Delegate -

Use with consoleApp .NET framework.

Delegate allows the programmers to encapsulate a reference to a method inside a delegate object. we need "delegate" keyword to declare delegate method. Delegate is used to implement the concept of pointer.

```
namespace Delegate_ConsoleApp
{
    delegate void FirstDelegate();
    0 references
    internal class Program
    {
        1 reference
        public static void HelloWorld()
        {
            Console.WriteLine("HelloWorld Delegate");
        }
        0 references
        static void Main(string[] args)
        {
            //Here we are passing reference of method
            FirstDelegate f1 = new FirstDelegate(HelloWorld);
            f1(); // Now delegate object is treat as a method.
            Console.WriteLine("End...");
        }
    }
}
```

User-Defined Delegate -

```
HelloWorld Delegate
End...
Press any key to continue .
```

Multicast Delegate -

We can pass multiple methods with one object of the delegate.

```
delegate void FirstDelegate(int x, int y);
0 references
internal class Program
{
    2 references
    public static void Add(int a, int b)
    {
        Console.WriteLine("Addition is "+(a+b));
    }
    1 reference
    public static void Mul(int a, int b)
    {
        Console.WriteLine("Multiplication is " + (a * b));
    }
    0 references
    public static void Sub(int a, int b)
    {
        Console.WriteLine("Substraction is " + (a - b));
    }
    0 references
    static void Main(string[] args)
    {
        FirstDelegate f1 = new FirstDelegate(Add);

        f1 += Mul; //using multiple methods.
        f1 += Add;
        f1(10, 5);//Passing 10, 5 to methods.
        Console.WriteLine("End...");
    }
}
```

```
Addition is 15
Multiplication is 50
Addition is 15
End...
```

Built-in delegate -

Func -

Func is a built-in delegate. It is used to hold the reference of one or more methods with same signature of the delegate object. It must include one output parameter for the return type.

Func <int, int, int> ➜ first two int are for parameters and the last int is an O/P parameter, for storing the result which is returned by the method. O/P parameter is must include.

```
namespace BuiltInDelegate
{
    0 references
    internal class Program
    {
        1 reference
        public static int Add(int a, int b)
        {
            return a + b;
        }
        1 reference
        public static int Mul(int a, int b)
        {
            return a * b;
        }
        0 references
        static void Main(string[] args)
        {
            int responce = 0;
            Func<int, int, int> operation = Add;
            responce = operation(10, 20);
            Console.WriteLine("Addition is "+responce);

            operation = Mul;
            responce = operation(5, 5);
            Console.WriteLine("Multiplication is " +responce);
        }
    }
}
```

```
Addition is 30
Multiplication is 25
Press any key to continue
```

Action -

Action is a built-in delegate. It is used to hold the reference of one or more methods with same signature of the delegate object. The only difference is that the Action delegate will not return anything.

```
internal class Program
{
    1 reference
    public static void Add(int a, int b)
    {
        Console.WriteLine("Addition is " +(a+b));
    }
    1 reference
    public static void Mul(int a, int b)
    {
        Console.WriteLine("Multiplication is " + (a * b));
    }
    0 references
    static void Main(string[] args)
    {
        Action<int, int> operation = Add;
        operation(45, 5);
        operation = Mul;
        operation(5, 5);
    }
}
```

```
Addition is 30
Multiplication is 25
Press any key to conti
```

Predicate Delegate -

The predicate is the delegate like Func and Action delegates. It represents a method containing a set of criteria and checks whether the passed parameter meets those criteria. A predicate delegate methods must take one input parameter and return a boolean - true or false.

```
namespace BuiltInDelegate
{
    0 references
    internal class Program
    {
        1 reference
        public static bool IsUpperCase(String str)
        {
            return str.Equals(str.ToUpper());
        }
        0 references
        static void Main(string[] args)
        {
            Predicate<string> isUpper = IsUpperCase;
            bool result = isUpper("SUMIT");
            Console.WriteLine(result);
        }
    }
}
```

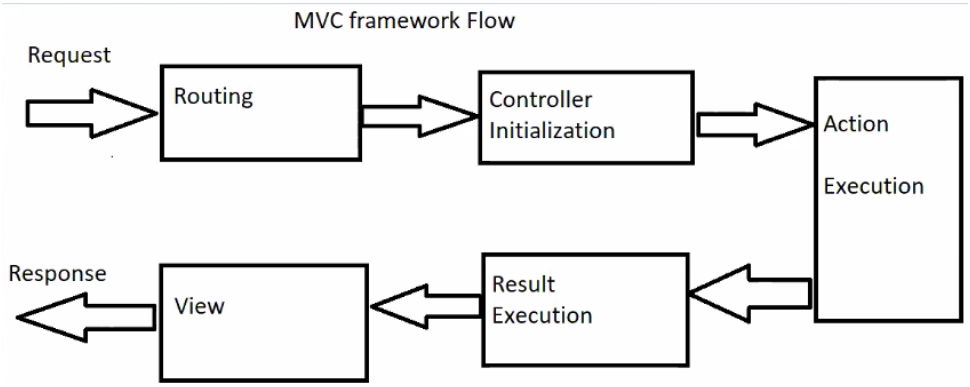
```
True
Press any
```

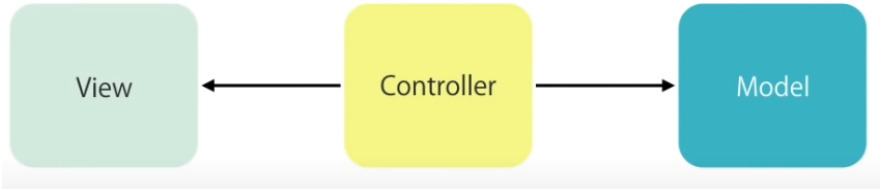
False is letters are not in uppercase.

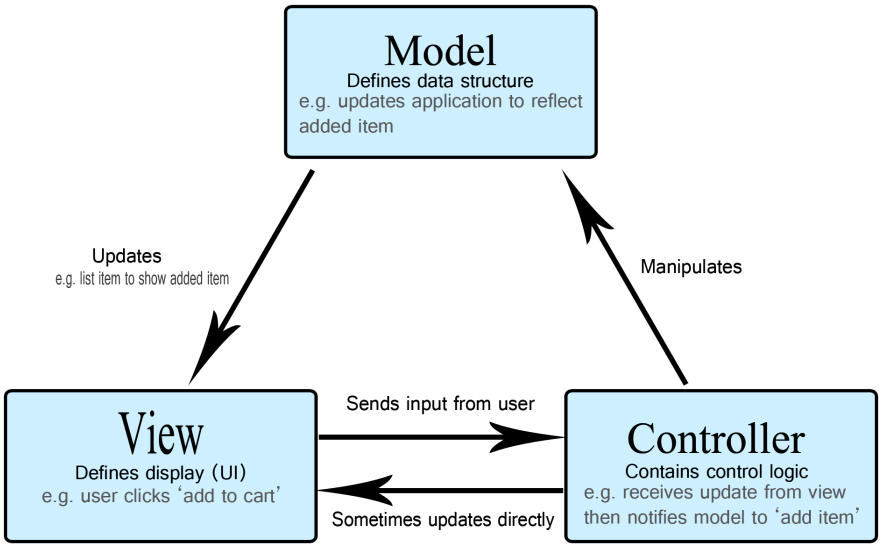
MVC (Model View Controller) -

MVC is a design/ architectural pattern separates the user interface of an application into 3 main parts -

1. **Controller -** It is a set of classes that handles the communication from a user and also used to communicate between classes in the model and view.
2. **Model -** A set of classes that describes the data we are working with and also our business logic.
3. **View -** Defines the application's UI (User Interface) will be displayed, it is a pure HTML.







Advantages -

1. It iis faster than normal we forms.
2. Using html helpers (html functions for designing) we can create html controls programatically.
3. HTML helpers are used in view page to render html content, mostly it is a method that returns a string.
4. Routing enables us to degine url pattern that maps to the define url pattern that maps to the required handlers.

HTML Helpers -

1. To create text box - @HTML.Textboxfor(m => m.Name) m is an object
2. To create rows and columns - @HTML.TextArea(m => m.Address, 5 , 20, new { }) – 5 is row, 20 is columns.
3. For password ... – @HTML.PasswordFor(m => m.Password).
4. For true or false – @HTML.CheckboxFor (m => m.IsApproved). Will return boolean.
5. For radio button – @HTML.RadiobuttonFor(m ⇒ m.Gender, "Male")
    a. @HTML.RadiobuttonFor(m ⇒ m.Gender, "Female")
6. For drop down list –

a. @HTML.FropdownListFor(m => m.Category, new SelectList (new [ ] {“General”, “OBC”,”SC”, “ST”}))

b. For Action –

i. @HTML.ActionLink(Display Name, method name, controller name). –ex ⬇

ii. @HTML.ActionLink(“List of employees”,”index”, “Employee”).’

7. @usin(Html.beginForm(“Index”,”Employee”))

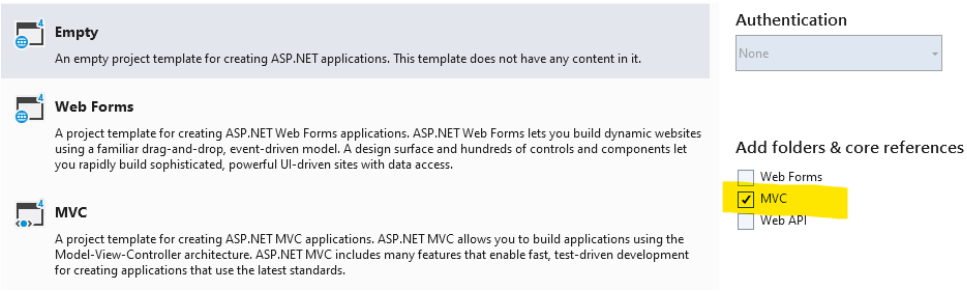
{

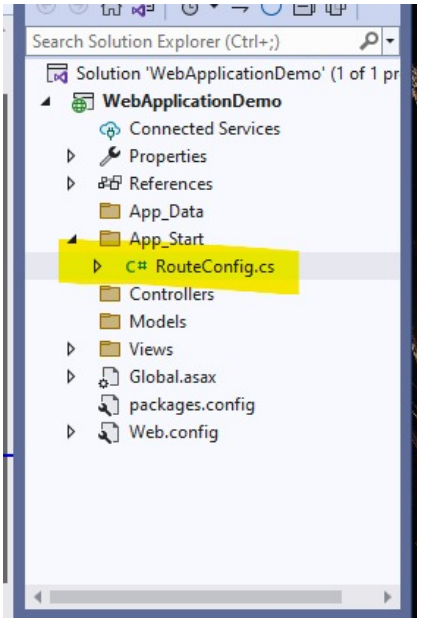
@HTML.TextboxFor(m => m.Name)

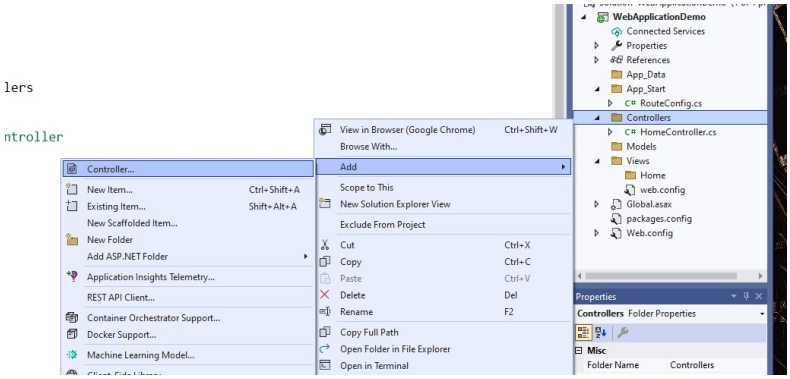
<input type = “submit” value= “Save”/>

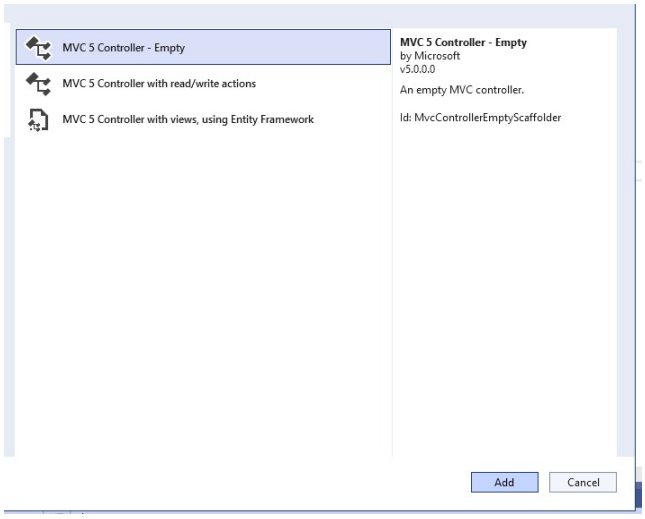
}

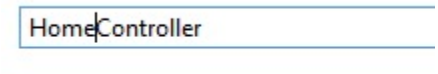
Create a new ASP.NET Web Application



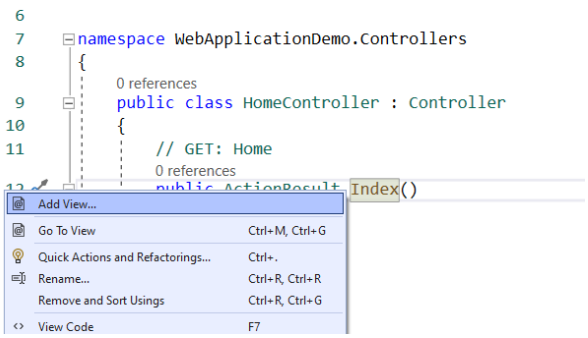




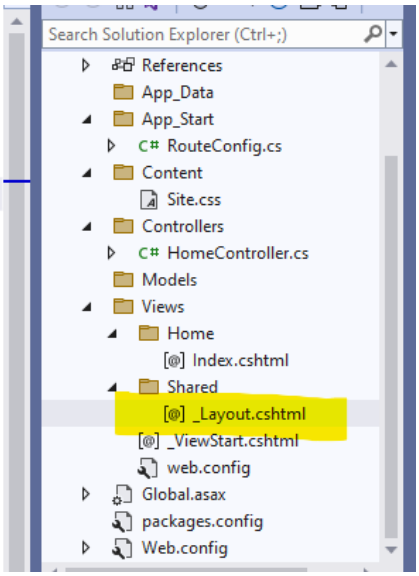




To add view -



To open layout -



Creating a web page -

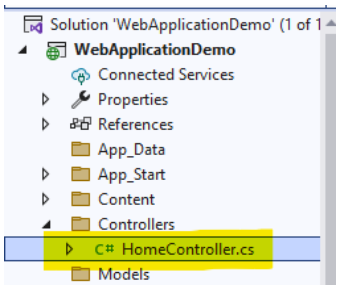
- Under Layout.cshtml.

```
@Html.ActionLink("CDAC Home Page", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
@Html.ActionLink("Contact Us", "ContactUs", "Home", new { area = "" }, new { @class = "navbar-brand" })
@Html.ActionLink("About Us", "AboutUs", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

Under footer -

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET CDAC Demo Application</p>
    </footer>
</div>
```

- Goto Controler - Created methods for Index, Contactus, AboutUs -




```
// GET: Home
0 references
public ActionResult Index()
{
    return View();
}
0 references
public ActionResult ContactUs()
{
    return View();
}
0 references
public ActionResult AboutUs()
{
    return View();
}
```

Add view for adding and creating a view -

Index -

```
<h2>Index</h2>
<p>MCV DEMO CLASS</p>
<p>.Net Class Topics</p>
<ol>
    <li>Object</li>
    <li>Class</li>
    <li>Inheritance</li>
</ol>
```

Contact -

```
<h2>ContactUs</h2>
<ol>
    <li>Name</li>
    <li>Surname</li>
    <li>Pune M.H</li>
    <li>Contact US End</li>
</ol>
```

AboutUs-

```
<h2>AboutUs</h2>
<p>Dot Net Batch 2022 Demo Application</p>
<ul>
    <li>Sumit - <span style="color:red"><b>Programmer</b></span></li>
    <li>Bhavsar - <span style="color:cornflowerblue"><b>Designer</b></span></li>

</ul>
```

Final Web page - home

CDAC Home Page Contact Us About Us

Index

MCV DEMO CLASS

.Net Class Topics

1. Object
2. Class
3. Inheritance

© 2022 - My ASP.NET CDAC Demo Application

- ContactUS

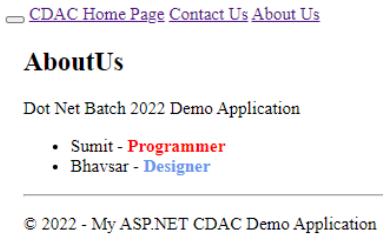
CDAC Home Page Contact Us About Us

ContactUs

1. Name
2. Surname
3. Pune M.H
4. Contact US End

© 2022 - My ASP.NET CDAC Demo Application

AboutUs -



ViewData-

ViewData is used to store data as an object. It is used for passing values from controller to view. It is available only for the current request, it will be destroyed on redirection. While fetching data it needs to be type cast.

Added Student data for studentController -

```
</button>
@Html.ActionLink("CDAC Home Page", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
@Html.ActionLink("Contact Us", "ContactUs", "Home", new { area = "" }, new { @class = "navbar-brand" })
@Html.ActionLink("About Us", "AboutUs", "Home", new { area = "" }, new { @class = "navbar-brand" })
@Html.ActionLink("Student data", "Index", "Student", new { area = "" }, new { @class = "navbar-brand" })
```

```
public class HomeController : Controller
{
    // GET: Home
    0 references
    public ActionResult Index()
    {
        List<string> Employee = new List<string>();
        Employee.Add("Sumit");
        Employee.Add("Tushar");
        Employee.Add("Mat");

        ViewData["EmployeeList"]= Employee;

        ViewData["CDAC-Message"] = "Hello World";
        return View();
    }
```

Created under index of homeController.cs -

```
<p>@ViewData["CDAC-Message"]</p>

@foreach (var item in ViewData["EmployeeList"] as List<string>)
{
        <p>@item</p>
}
```

ViewBag -

While fetching data there is no need for type cast.

```
public ActionResult ContactUs()
{
    List<string> Students = new List<string>();
    Students.Add("Sumit");
    Students.Add("Tushar");
    Students.Add("Mohit");

    ViewBag.StudentList = Students;

    return View();
}
```

```
@foreach (var item in ViewBag.StudentList)
{
    <p>@item</p>
}
```

TempData -

TempData is slower as compared to ViewData. It is used to pass values from one controller to another controller.

```
public ActionResult AboutUs()
{
    List<string> Students = new List<string>();
    Students.Add("Pratik");
    Students.Add("Akshay");
    Students.Add("Swarup");
    TempData["StudentL"] = Students;
    return View();
```

For the same controller.

For passing the value to a different controller. Eg - student controller.

```
public ActionResult AboutUs()
{
    List<string> Students = new List<string>();

    return RedirectToAction("Index", "Student");
```

It will only redirect.

Then to add a different class -

Right-click on Models - Add > new itom > class.

```
public class StudentViewModel
{
    2 references
    public string Name { get; set; }
    2 references
    public int Id { get; set; }

    3 references
    public bool IsMarried { get; set; }

    2 references
    public string City { get; set; }
}
```

Creating a different controller -

Right-click on controllers > add > Controller.

```
0 references
public class StudentController : Controller
{
    // GET: Student
    0 references
    public ActionResult Index()
    {
        StudentViewModel studentObject = new StudentViewModel();
        studentObject.Id = 101;
        studentObject.Name = "sumit bhavsar";
        studentObject.City = "beed";
        studentObject.IsMarried = false;

        return View(studentObject);
    }
}
```

```
@model WebApplicationDemo.Models.StudentViewModel

@{
    ViewBag.Title = "Index";
}

<h2>Index - Student Controller</h2>
<p>Name: @Model.Id</p>
<p>Name: @Model.Name</p>
<p>Name: @Model.City</p>
@if (Model.IsMarried) {
    <p>Marital Status: Yes</p>
}
else
{
    <p>Marital Status: No</p>
}
<p>@TempData["Message"]</p>
```

Used the values in the Index of StudentController.

To create a new COn

Adding a list of elements through an object -

7-6-22

1. Create a view Model -

```
public class StudentViewModel
{
    7 references
    public StudentViewModel()
    {
        StudentList = new List<StudentViewModel>();
    }
    6 references
    public string Name { get; set; }
    8 references
    public int Id { get; set; }
    7 references
    public string City { get; set; }
    6 references
    public bool IsMarried { get; set; }
    5 references
    public string FatherName { get; set; }
    4 references
    public string Address { get; set; }
    5 references
    public bool Ismale { get; set; }
    10 references
    public List<StudentViewModel> StudentList { get; set; }
}
```

2. Create an object and assign the values -

```
public List<StudentViewModel> GetStudentsList()
{
    StudentViewModel studentModel = new StudentViewModel();

    StudentViewModel studentObject1 = new StudentViewModel();
    studentObject1.Id = 102;
    studentObject1.Name = "sumit Bhavsar";
    studentObject1.City = "beed";
    studentObject1.IsMarried = false;
    studentObject1.FatherName = "xyz";
    studentObject1.Address = "n.k colony abc";
    studentObject1.Ismale = false;


   StudentViewModel studentObject2 = new StudentViewModel();
    studentObject2.Id = 103;
    studentObject2.Name = "tushar gadge";
    studentObject2.City = "pune";
    studentObject2.IsMarried = false;
    studentObject2.FatherName = "xyz";
    studentObject2.Address = "n.k colony abc";
    studentObject2.Ismale = false;

    StudentViewModel studentObject3 = new StudentViewModel();
    studentObject3.Id = 104;
    studentObject3.Name = "akshay saste";
    studentObject3.City = "xyz";
    studentObject3.IsMarried = false;
    studentObject3.FatherName = "xyz";
    studentObject3.Address = "n.k colony abc";
    studentObject3.Ismale = false;

    StudentViewModel studentObject4 = new StudentViewModel();
    studentObject4.Id = 105;
    studentObject4.Name = "pratik zambre";
    studentObject4.City = "xyz";
    studentObject4.IsMarried = false;
    studentObject4.FatherName = "xyz";
    studentObject4.Address = "n.k colony abc";
    studentObject4.Ismale = false;

    //adding student data in list of student object
    studentModel.StudentList.Add(studentObject1);
    studentModel.StudentList.Add(studentObject2);
    studentModel.StudentList.Add(studentObject3);
    studentModel.StudentList.Add(studentObject4);
    return studentModel.StudentList;
```

3. goto view of the Student record and add the items -

```
@model WebApplicationModelUsage.Models.StudentViewModel
```

```
<h2>StudentRecords</h2>

<table border="1">
    <tr>
        <th>Id</th>
        <th>Name</th>
        <th>City</th>
        <th>Marital Status</th>
    </tr>
```

```
<tbody>
    @foreach (var item in Model.StudentList)
    {
        <tr>
            <td>@item.Id</td>
            <td>@item.Name</td>
            <td>@item.City</td>
            @if (item.IsMarried)
            {
                <td>Married</td>
            }
            else
            {
                <td>Unmarried</td>
            }
            @*<td><a href="/Home/Index?studentId=@item.Id">select</a></td>*@
        </tr>

    }
</tbody>
```

Also add the ActionLink for the Student Records -

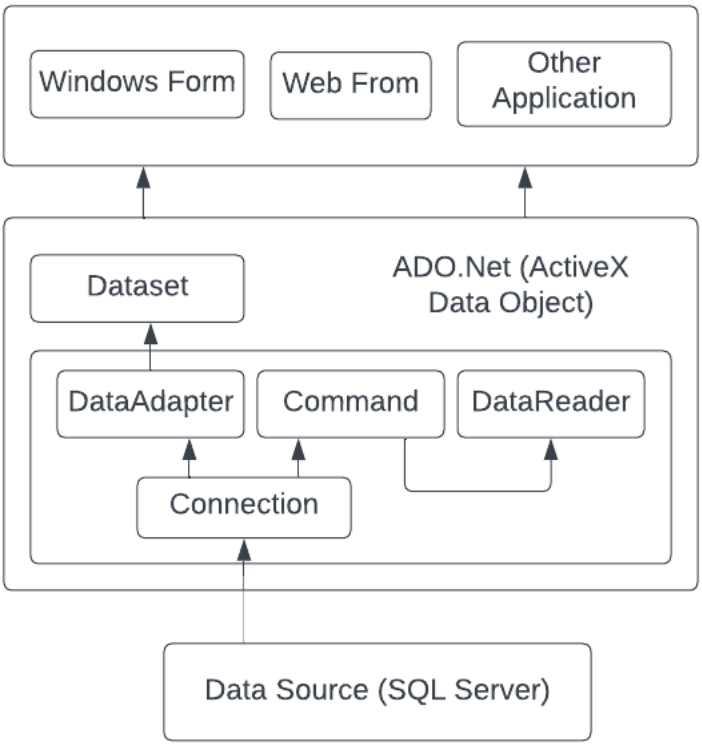
```
@Html.ActionLink("Student Records", "StudentRecords", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

In the Index view -

```
<h2>Home - Index</h2>

<h4>Student Details</h4>

<p>ID - <b>@Model.Id</b></p>
<p>NAme - <b>@Model.Name</b></p>
<p>City - <b>@Model.City</b></p>
@if (Model.Ismale)
{
    <p>Gender - <b>Male</b></p>
}
else
{

    <p>Gender - <b>Female</b></p>
}

<p>Father Name - <b>@Model.FatherName</b>></p>
<p>City - <b>@Model.City</b>></p>
@if (Model.IsMarried)
{
    <p>Marital Status - <b>Married</b></p>
}
else
{
    <p>Marital Status - <b>Unmarried</b></p>
}
```

```
public ActionResult Index(int studentId = 0)
{
    StudentViewModel studentModel = new StudentViewModel();
    studentModel.StudentList = GetStudentsList();
    foreach(var item in studentModel.StudentList)
    {
        if(item.Id==studentId)
        {
            studentModel = item;
        }
    }
    return View(studentModel);
```

.Net to Database -



1. Select project ASP.NET Web Application (.NET Framework)
    a. Empty project - mvc
2. Add controller - add Index view
3. Add a class in Models.
4. Change name in layout.

1. ModelIt > Class > DemoUserModel

```
namespace WebApplicationApnaAdoUse.Models
{
    14 references
    public class DemoUserViewModel
    {
        3 references
        public DemoUserViewModel()
        {
            UsersList = new List<DemoUserViewModel>();
        }
        8 references
        public int Id { get; set; }
        6 references
        public string Name { get; set; }
        6 references
        public string City { get; set; }
        3 references
        public List<DemoUserViewModel> UsersList { get; set; }
    }
}
```

Index and SqlConnection -

```
namespace WebApplicationApnaAdoUse.Controllers
{
    0 references
    public class HomeController : Controller
    {
        SqlConnection con = new SqlConnection("Data Source = LAPTOP-N64HMONM; Database=dbCdacBcatch12022;Integrated Security=true;");
        // GET: Home
        0 references
        public ActionResult Index()
        {
            DemoUserViewModel userModel = new DemoUserViewModel();

            con.Open();
            SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM DemoUser", con);
            DataTable dt = new DataTable();
            da.Fill(dt);

            foreach (DataRow dr in dt.Rows)
            {
                DemoUserViewModel user = new DemoUserViewModel();
                user.Id = Convert.ToInt32(dr["Id"]);
                user.Name = dr["Name"].ToString();
                user.City = dr["City"].ToString();

                userModel.UsersList.Add(user);
            }

            con.Close();
            return View(userModel);
        }
```

Action Edit Demouser Get -

```
public ActionResult EditDemoUser(int id)
{
    DemoUserViewModel userModel = new DemoUserViewModel();
    TempData["Title"] = "Add new User";

    if (id > 0)
    {
        TempData["Title"] = "Edit user";
        con.Open();
        SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM DemoUser WHERE ID = " + id, con);
        DataTable dt = new DataTable();
        da.Fill(dt);

        userModel.Id = Convert.ToInt32(dt.Rows[0][0]);
        userModel.Name = dt.Rows[0][1].ToString();
        userModel.City = dt.Rows[0][2].ToString();
        con.Close();
    }
    return View(userModel);
}
```

Action EditDemouser for edit and update -

```
[HttpPost]
0 references
public ActionResult EditDemoUser(DemoUserViewModel userModel)
{
    if (userModel.Id > 0)
    {
        con.Open();

        string updateQuery = "UPDATE DemoUser SET Name='" + userModel.Name + "', City='" + userModel.City + "' WHERE Id=" + userModel.Id;
        SqlCommand cmd = new SqlCommand(updateQuery, con);
        cmd.ExecuteNonQuery();

        con.Close();
    }
    else
    {
        con.Open();

        string query = "INSERT INTO DemoUser(Name,City) VALUES('" + userModel.Name + "', '" + userModel.City + "')";
        SqlCommand cmd = new SqlCommand(query, con);
        cmd.ExecuteNonQuery();

        con.Close();
    }
    return RedirectToAction("Index", "Home");
}
```

Delete -

```
public ActionResult DeleDemoUser(int id)
{
    con.Open();

    string query = "DELETE FROM DemoUser WHERE ID = "+id;
    SqlCommand cmd = new SqlCommand(query, con);
    cmd.ExecuteNonQuery();

    con.Close();
    return RedirectToAction("Index", "Home");
}
```

Index View Setting -

```
@model WebApplicationApnaAdoUse.Models.DemoUserViewModel
@{
    ViewBag.Title = "Index";
}
```

```
<table>
    <tr>
        <th>Id</th>
        <th>Name</th>
        <th>City</th>
    </tr>
    <tbody>
        @foreach (var item in Model.UsersList)
            {
                <tr>
                    <td>@item.Id</td>
                    <td>@item.Name</td>
                    <td>@item.City</td>

                    <td><a href="/Home/EditDemoUser?id=@item.Id">Edit</a></td>
                     <td><a href="/Home/EditDemoUser?id=@item.Id">Delete</a></td>

                </tr>
            }
    </tbody>
</table>
```

View of EditDemoUser -

```
@model WebApplicationApnaAdoUse.Models.DemoUserViewModel

@{
    ViewBag.Title = "EditDemoUser";
}

<h2>Home - @TempData["Title"]</h2>

@using (Html.BeginForm("EditDemoUser", "Home", FormMethod.Post))
{
    @Html.HiddenFor(m => m.Id)
    <br />

    @Html.TextBoxFor(m => m.Name)
    <br />
    @Html.TextBoxFor(m => m.City)
    <br />
    <input type="submit" value="Save"/>
}
```

Changes in Layout.cshtml file -

```
    </button>
    @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
    @Html.ActionLink("Edit User", "EditDemoUser", "Home", new { area = "" }, new { @class = "navbar-brand" })
</div>
```

- C#.net
    - Console app and windows app
- ADO.net and entity framework
- .NET MVC A
- API.

Entity Framework -

Microsoft has provided an ORM (Object Relational Model) called “EF” to automate database-related activities for our applications. AN ADO.Net EF is an object-relational Mapping Framework that enables developers to work with relational data as domain specific object.

Using EF, developers issue queries using LINQ (Language Integrated Query)

```
Int [ ] values = new int [ ] {4,5,6,7,8}
Var output = from i in values
                Where i < 6
                Select i;
```

Project -

1. webApplication .Net Framework
2. Create project > select MVC
3. Add Controller
4. Change name of controller and ActionResult from _layout.cshtml
5. Add a folder names Data > Add new itom > Data > ADO.net Entity Data Model.
    a. Select ADO.NET Entity DataModel from data... > next -
    b. Choose Your data Connection window will open > New Connection
    c. Choose Data source > Microsoft SQL Server > continue
    d. Again connection properties window will open > Enter into Server Name >
        i. LAPTOP-N64HMONM
        ii. Select database i.e - dbCdacBcatch12022
        iii. Test The connection
    e. Entity Data Model Wizard will open > Next
    f. Select Entuty Framework 6.x or latest > Next
    g. Select Tables > Finish
6. Save the window “EmployeeData.edmx”
7. In Data folder > Context.cs > It will have all the info about the table
8. DemoUser.cs it will have properties.
9. Open Context.cs file > copy the class ex ➔ dbCdacBcatch12022Entities1
    a. Goto controller and create an object.
10. Add Model class > DemoUserViewModel > Add the properties from demoUser.cs.
11. Change Route > App_start folder > RouteConfig > change controller and action ex employee and EmployeeList.

1. In Employee Controller -

```
public class EmployeeController : Controller
{
    dbCdacBcatch12022Entities3 db = new dbCdacBcatch12022Entities3();
    // GET: Employee
    0 references
    public ActionResult Index()
    {
        return View();
    }
    0 references
    public ActionResult EmployeeList()
    {
        EmployeeViewModel model = new EmployeeViewModel();

        var records = db.DemoUsers.ToList();

        foreach(var item in records)
        {
            EmployeeViewModel employee = new EmployeeViewModel();
            employee.Id = item.Id;
            employee.City = item.City;
            employee.Name = item.Name;
            employee.StateId = item.StateId;

            if (item.StateId !=null && item.StateId > 0)
            {
                employee.StateName = db.States.Where(m => m.Id == item.StateId).FirstOrDefault().StateName;
            }
            model.EmployeeList.Add(employee);
        }
        return View(model);
    }
```

2. Get Edit and Add Employee Method -

```
public ActionResult AddEditEmployee(int employeeId)
{
    EmployeeViewModel model = new EmployeeViewModel();
    if(employeeId > 0)
    {
        var employeeData = db.DemoUsers.FirstOrDefault(m => m.Id == employeeId);


        model.Id = employeeData.Id;
        model.City = employeeData.City;
        model.Name = employeeData.Name;
        model.StateId = employeeData.StateId;


        if (employeeData.StateId != null && employeeData.StateId > 0)
        {
            model.StateName = db.States.Where(m => m.Id == employeeData.StateId).FirstOrDefault().StateName;
        }
    }
    return View(model);
}
```

3. Post Edit and Add Employee Method -

```
[HttpPost]
0 references
public ActionResult AddEditEmployee(EmployeeViewModel model)
{
    if (model.Id > 0)
    {
        //Edit / Update Code
        var employeeData = db.DemoUsers.FirstOrDefault(m => m.Id == model.Id);
        employeeData.Name = model.Name;
        employeeData.City = model.City;
        db.SaveChanges();//Saves all the above changes.

    }
    else
    {
        //Insert Code
        DemoUser user = new DemoUser();
        user.City = model.City;
        user.Name = model.Name;
        user.StateId = model.StateId;

        db.DemoUsers.Add(user);
        db.SaveChanges();
    }
    return RedirectToAction("EmployeeList", "Employee");
}
```

4. Delete Employee Method -

```
0 references
public ActionResult DeleteEmployee(int employeeId)
{
    var employeeData = db.DemoUsers.FirstOrDefault(m => m.Id == employeeId);
    db.DemoUsers.Remove(employeeData);
    db.SaveChanges();

    return RedirectToAction("EmployeeList", "Employee");
}
```

Employee View Model Class -

```
namespace WebApplicationEFUse.Models
{
    14 references
    public class EmployeeViewModel
    {
        3 references
        public EmployeeViewModel()
        {
            EmployeeList = new List<EmployeeViewModel>();
        }
        8 references
        public int Id { get; set; }
        6 references
        public string Name { get; set; }
        7 references
        public string City { get; set; }
        3 references
        public Nullable<int> StateId { get; set; }
        3 references
        public string StateName { get; set; }
        3 references
        public List<EmployeeViewModel> EmployeeList { get; set; }
    }
}
```

View of Employee Liste Method -

```
@model WebApplicationEFUse.Models.EmployeeViewModel
@{
    ViewBag.Title = "EmployeeList";
}

<h2>EmployeeList</h2>
<a href="/Employee/AddEditEmployee?employeeId=0">Add New Employee</a><br /><br />
```

```
<table border="1">
    <tr>
        <th>Id</th>
        <th>Name</th>
        <th>City</th>
        <th>State</th>
        <th>Edit</th>
        <th>Delete</th>
    </tr>

    @foreach (var item in Model.EmployeeList)
    {
        <tr>
            <td>@item.Id</td>
            <td>@item.Name</td>
            <td>@item.City</td>
            <td>@item.StateName</td>

            <td><a href="/Employee/AddEditEmployee?employeeId=@item.Id">Edit</a></td>
            <td><a href="/Employee/DeleteEmployee?employeeId=@item.Id">Delete</a></td>

        </tr>
    }
</table>
```

View of Add Edit Employee Method -

```
@model WebApplicationEFUse.Models.EmployeeViewModel
@{
    ViewBag.Title = "AddEditEmployee";
}

<h2>Employee - Add / EditEmployee</h2>

@using (Html.BeginForm("AddEditEmployee", "Employee", FormMethod.Post))
{
    @Html.HiddenFor(m => m.Id)

    <div>Name</div>
    @Html.TextBoxFor(m => m.Name)
    <br />
    <br />

    <div>City</div>
    @Html.TextBoxFor(m => m.City)
    <br />
    <br />

    <div>State</div>
    @Html.TextBoxFor(m => m.City)
    <br />
    <br />

    <input type="submit" value="Save Kara" />
}
```

API (Application Programming Interface) -

A piece of code which does a specific task and show certain properties which we call the state of a piece of code. This kind of piece of code when put together in one single file is called an API.

API types are -

- HttpGet
- HttpPost
- HttpPut
- HttpDelete

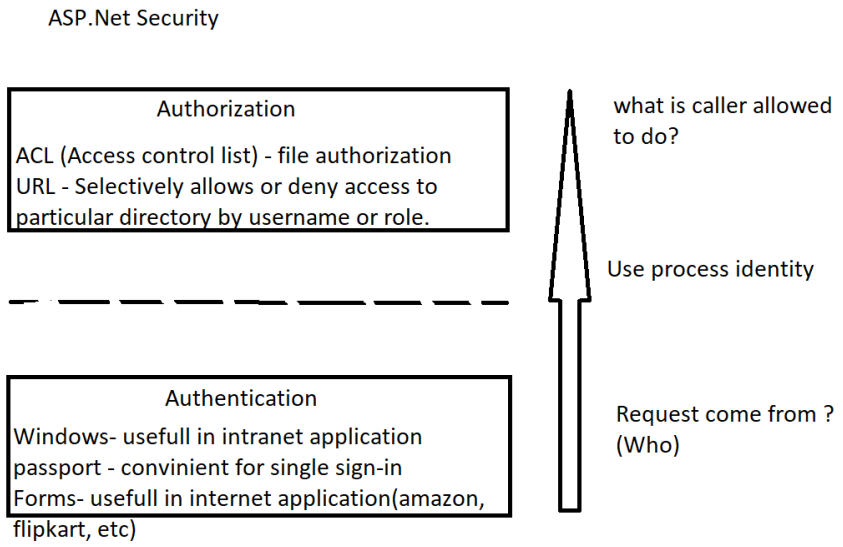
- Web API is a http-based service
- It returns data in JSON and XML format.
- It is open-source, it can be used by any client that understands JSON and XML.
- It is lightweight architecture and good for devices that have limited bandwidth like mobile devices.

https://api.openweathermap.org/data/2.5/weather?q=mumbai&units=metric&cnt=1&APPID=8113fcc5a7494b0518bd91ef3acc074f

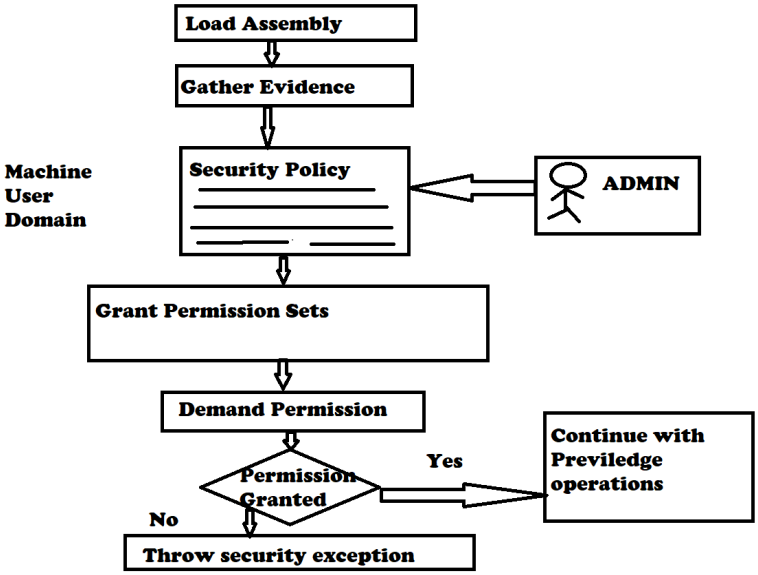
Muti-Threading -

c# supports parallel execution of job or program code through threading. If we talk about multithreading, it contains 2 0r more program codes which run concurrently without affecting to each other and each program code is called Thread.

- We can create a thread with the help of the thread class.
- We can call the method with the help of the ThreadStart delegate.



DOT NET SECURITY -



Security

- Role-Based Security
    - It allows you to partition your website according to the role of the user. That means once the user is logged-in, the determination as to whether or not access to a resource is granted. Roles like - admin, guest, premium subscriber, etc..

- **Code Access Security**
    - **Evidence -**
        - It determines what permission to grant to code area.
        - The known information about .Net assembly.
        - Where the code is loaded from: site, URL & application directory.
        - Who wrote the code.

Policy

- It determines the permission granted to assemblies.
- It is configurable by the System admin & Users.

Permission -

- Rights for code
- It is granted by code access security policy.
- Evidence + Policy = Permission

Resource file -