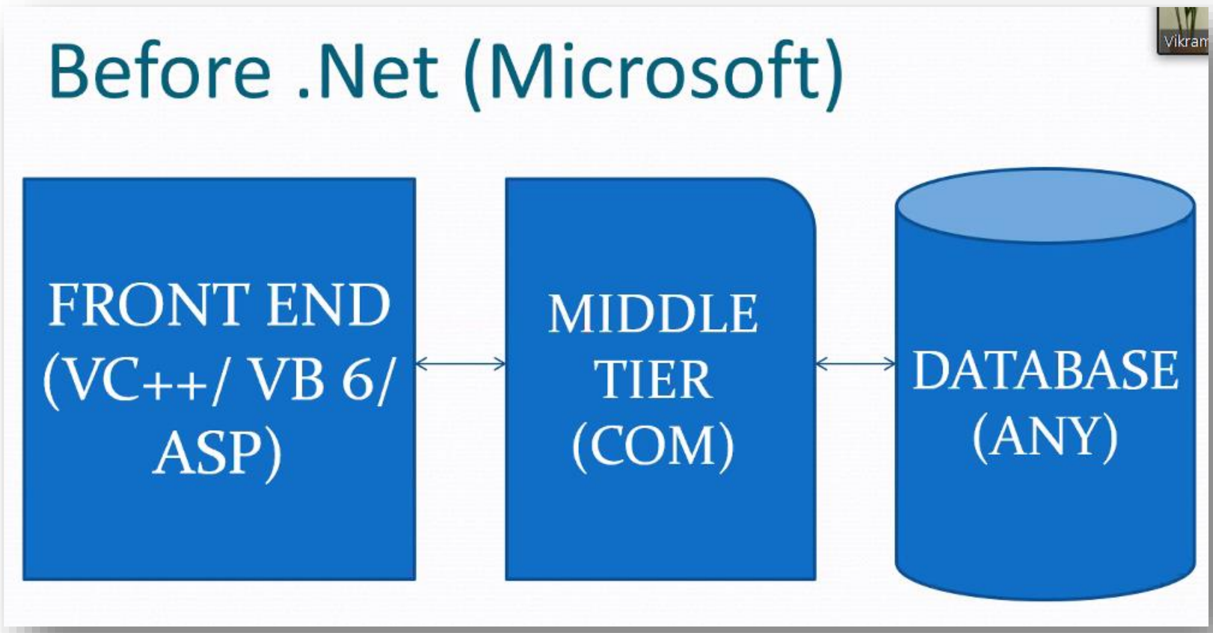
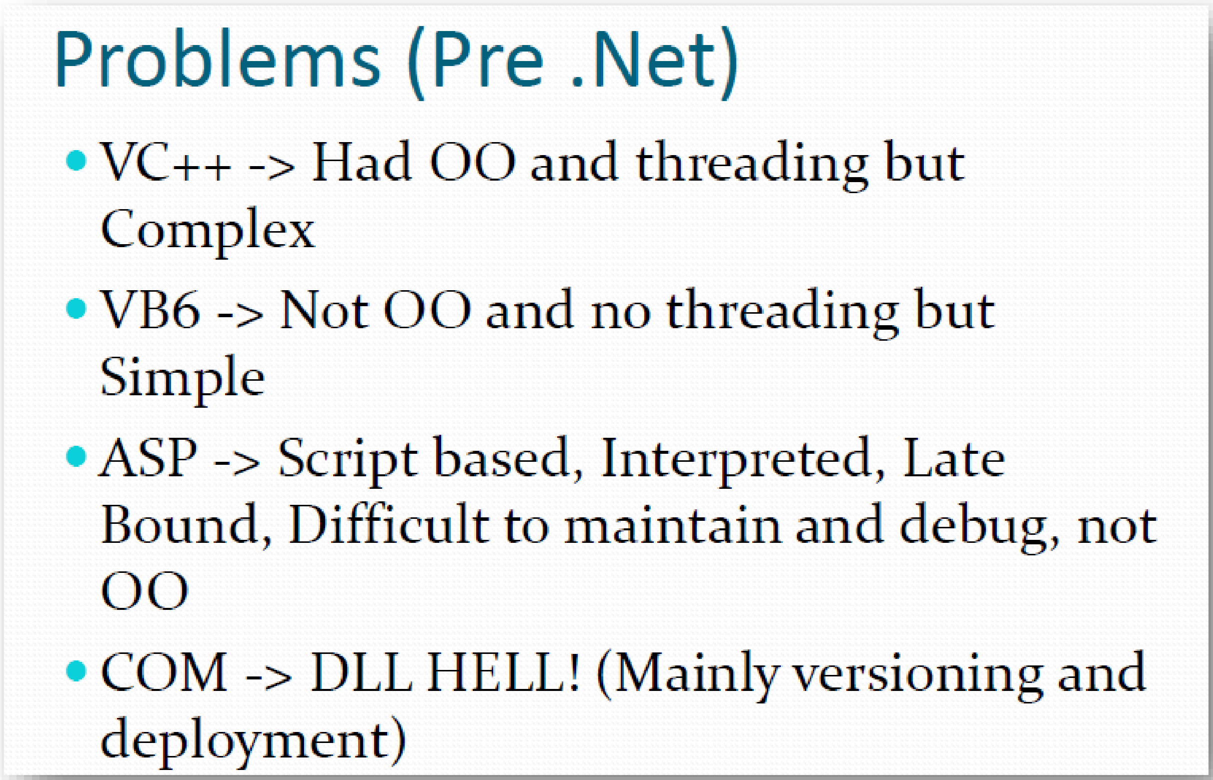
Day1

.NET



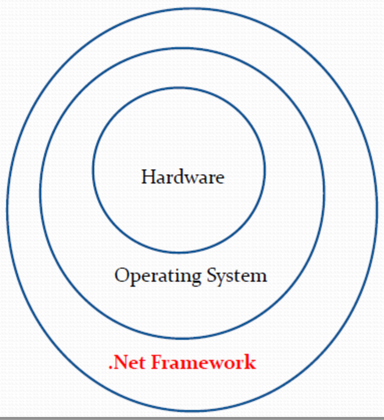
Before .Net (Microsoft)

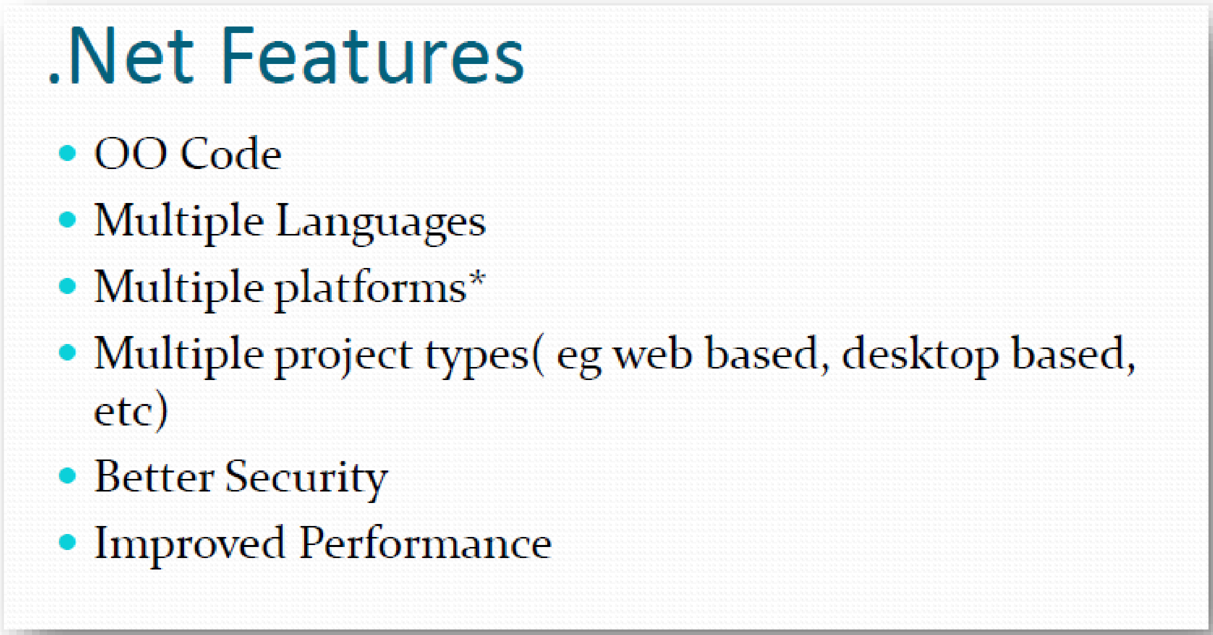


Problems (Pre .Net)

- VC++ -> Had OO and threading but Complex
- VB6 -> Not OO and no threading but Simple
- ASP -> Script based, Interpreted, Late Bound, Difficult to maintain and debug, not OO
- COM -> DLL HELL! (Mainly versioning and deployment)

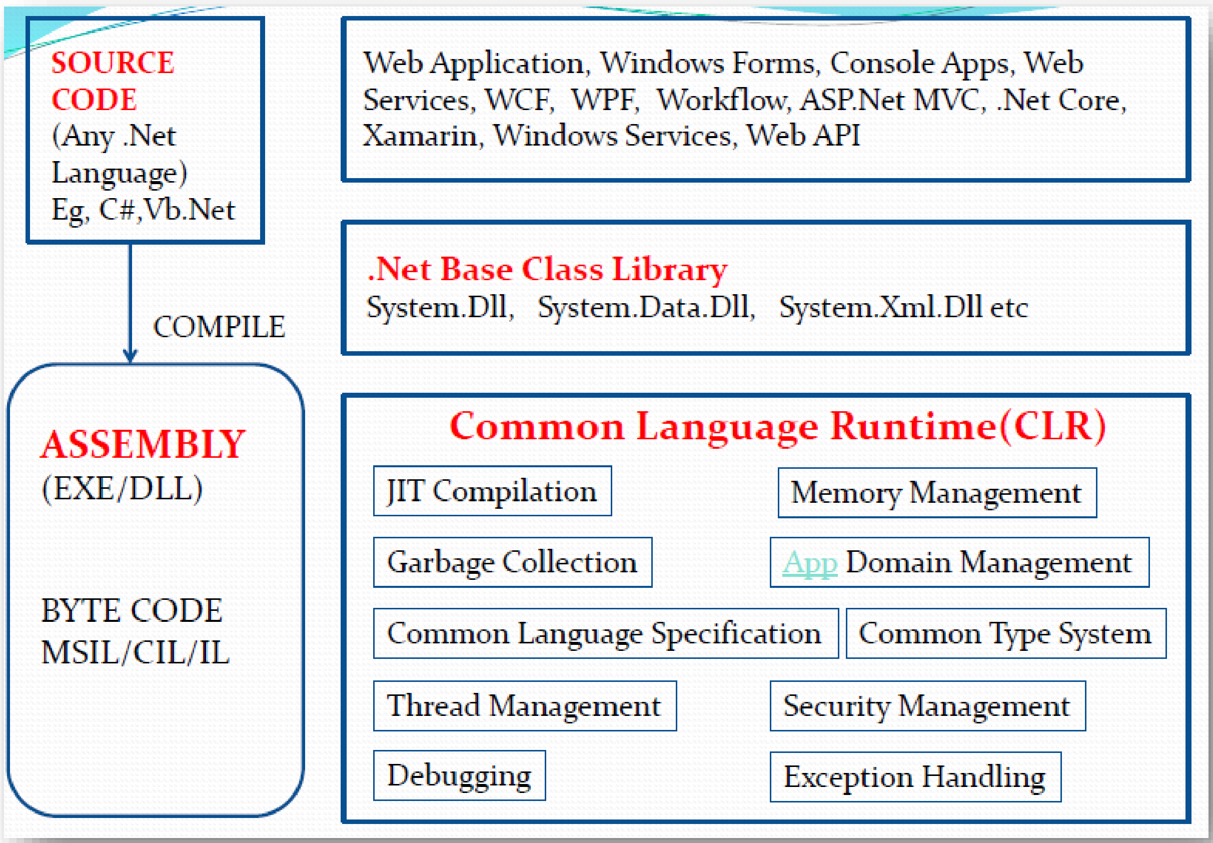
.Net framework





.Net Features

- OO Code
- Multiple Languages
- Multiple platforms*
- Multiple project types(eg web based, desktop based, etc)
- Better Security
- Improved Performance


SOURCE CODE (Any .Net Language) Eg, C#,Vb.Net
COMPILE
ASSEMBLY (EXE/DLL)
BYTE CODE MSIL/CIL/IL
Web Application, Windows Forms, Console Apps, Web Services, WCF, WPF, Workflow, ASP.Net MVC, .Net Core, Xamarin, Windows Services, Web API
.Net Base Class Library
System.Dll, System.Data.Dll, System.Xml.Dll etc
Common Language Runtime(CLR)
JIT Compilation
Memory Management
Garbage Collection
App Domain Management
Common Language Specification
Common Type System
Thread Management
Security Management
Debugging
Exception Handling

Source Code- Any language code after compilation we get **Assesemly** with .dll or .exe extensions,

Having byte code format known as IL(**Intermediate language**) /MSIL(microsoft IL)/CIL(Common IL)

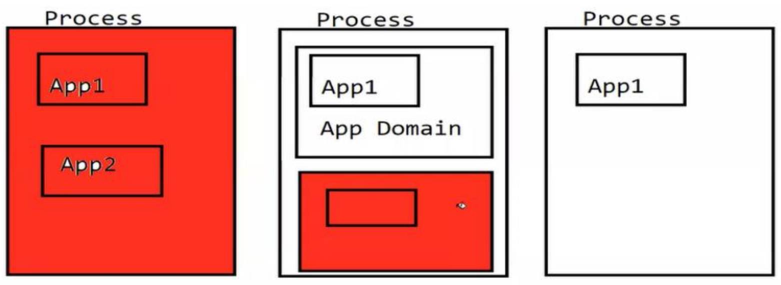
An Assembly - A collection of types and resources that are built to work together and form a logical unit of functionality. Common Language Runtime(CLR) take the form of executable (.exe) or dynamic link library (. dll) files, and are the building blocks of . NET applications.

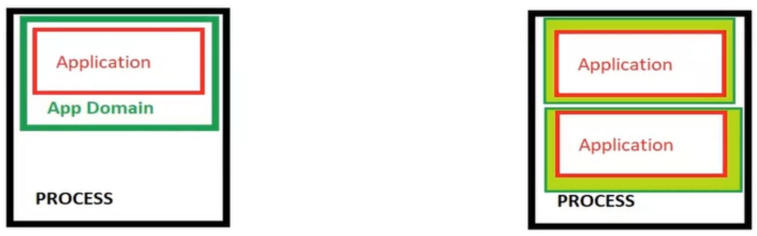
Assembly is equivalent to jar file in Java.

Common Language Runtime (CLR) similar to JVM

- **JIT Complication- IL to native code**
- IL to Native code invloves a overhead but does not impact speed/performance as it is miniscule (code will run with same speed as of that native code due to faster load time only require fucntion is loaded and not entire library) also native code is converted based on processor on which it is going to run. Smart conversion.
- Also known as **Jitter**
- **Memory management**
- Keeping track of objects, references

- **Garbage Collection**
- Contructor- Allocates memory- A function w/o return type and same name as of class
- Destructor-Deallocates memory-call at the time of memory is deallocating(GC)
- Manage by CLR and we haven't controll over GC same as Java(Non deterministic finalization)
  Beacause of above reason it is not recomanded to write code in destructor.
- Wirte a fucntion dispose obj.Dispose() ---> call by externally ; to run the code required in destructor.
- **App Domain management** – can run multiple application in single process using app domain





-**CLS (Common Language Specification)**

-Roles specified which any .Net language should follow(Only common features of languages allowed e.g VB multiple inheritance but .Net doesn't support)

-Common type System

-Data types of differnet languages can have different sizes (e.g int is 4 byte in VC++ and 2 byte in VB)

-This will cause issue in .Net as there are many .net languages

-In order to avoid this problem int of any language is coverted to Int 32 (CTS data type 4 bytes)

-Thread Management

-Security Management

-Debugging

-Exception Handling

So what is the .NetFramework???

⇨ .Net Base Class Library+CLR+Utilities (JIT compiler,csc, etc.)

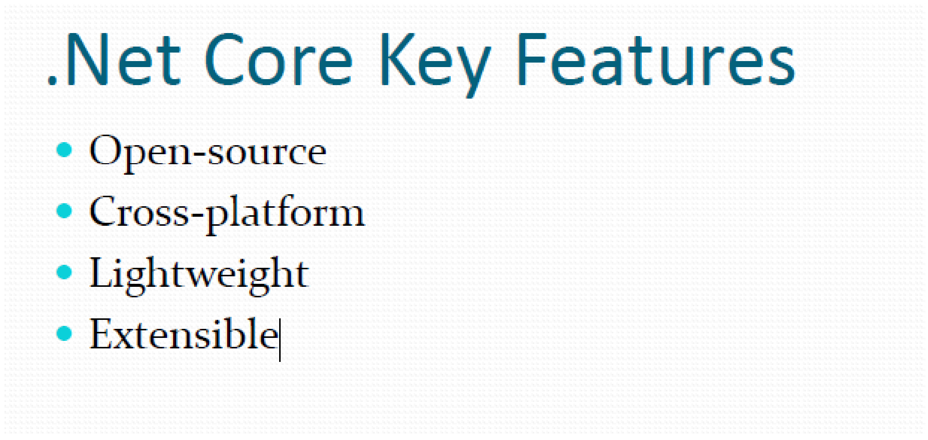
And what would you need to run your code on other platforms???

⇨ **.Net Base Class Library**

⇨ **CLR**

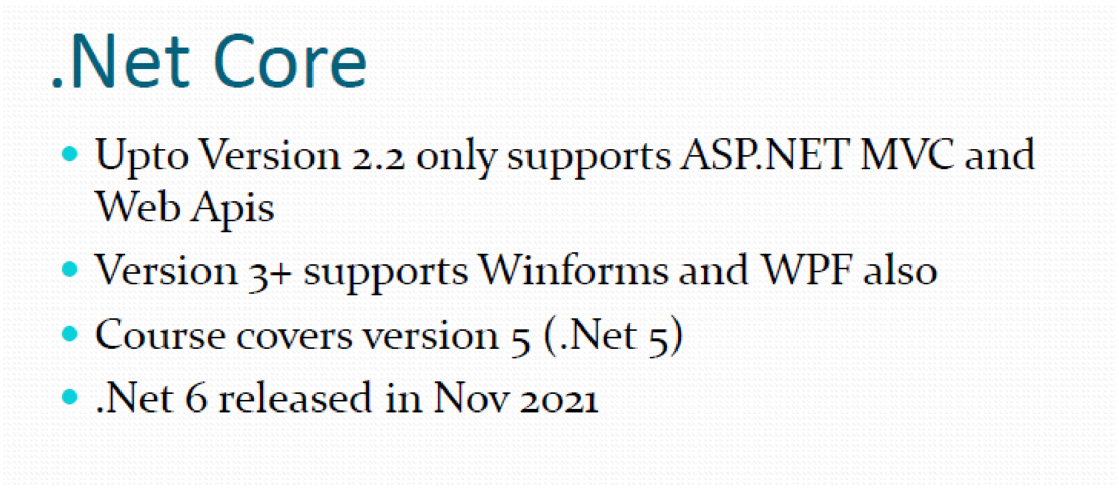
.Net Framework windows only
.Net Core all Platforms
Depends which version!!!

- From version 5 .net framework is no longer there only .net core
- Last .net framwork 4.8
- Last .net core 3
- .Net Framework preinstall files on machine
- .Net core copies all required files



.Net Core Key Features

- Open-source
- Cross-platform
- Lightweight
- Extensible



.Net Core

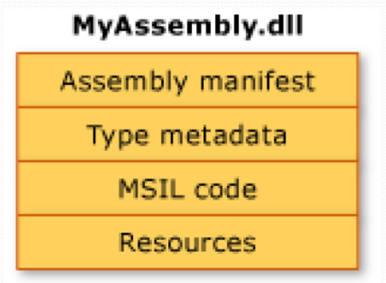
- Upto Version 2.2 only supports ASP.NET MVC and Web Apis
- Version 3+ supports Winforms and WPF also
- Course covers version 5 (.Net 5)
- .Net 6 released in Nov 2021

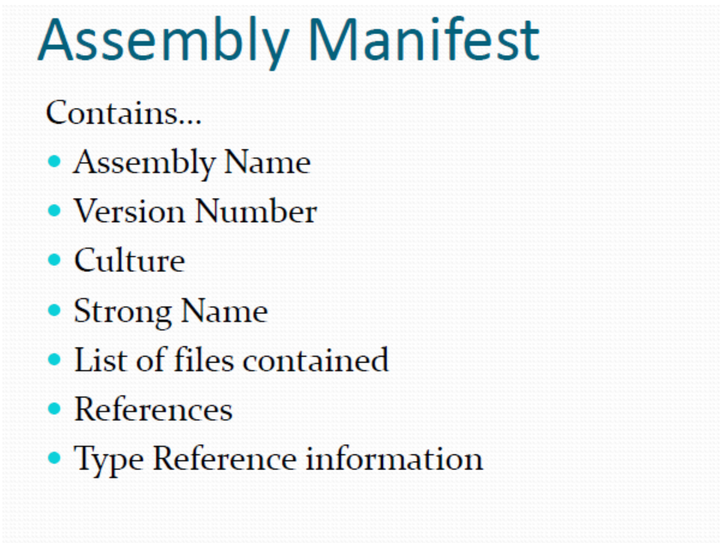
An Assembly - A collection of types and resources that are built to work together and form a logical unit of functionality. Assemblies in the form of executable (.exe) or dynamic link library (. dll) files, and are the building blocks of . NET applications.

Assembly is equivalent to jar file in Java.

Metadata- data about types, classes

Assembly Structure





Assembly Manifest

Contains...

- Assembly Name
- Version Number
- Culture
- Strong Name
- List of files contained
- References
- Type Reference information

Assemblies have the following properties:

- Assemblies are implemented as *.exe* or *.dll* files.m
- For libraries that target .NET Framework, you can share assemblies between applications by putting them in the global assembly cache (GAC). You must strong-name assemblies before you can include them in the GAC. For more information, see Strong-named assemblies.
- Assemblies are only loaded into memory if they are required. If they aren't used, they aren't loaded. This means that assemblies can be an efficient way to manage resources in larger projects.
- You can programmatically obtain information about an assembly by using reflection. For more information, see Reflection (C#) or Reflection (Visual Basic).
- You can load an assembly just to inspect it by using the MetadataLoadContext class on .NET and .NET Framework. MetadataLoadContext replaces the Assembly.ReflectionOnlyLoad methods.

Codes

```
//using is not a import ,it is only used for allias of nameSapce, for import you need
add refernces in refernces.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NameSapceName
{
    class ClassName
    {
        static void Main(string[] args)
        {
            MyNameSpace.Class1 c = new MyNameSpace.Class1();
            System.Console.WriteLine("Hello World");
            c.Display();
            System.Console.ReadLine();
        }
    }

}

- static void Main(string[] args) – cmd line args not compulsory [Main()]
```

```
//optional parameters with default values(always start from end)
        public int Add(int a=0, int b=0, int c=0)
        {
            return a + b + c;

        }

//calling optional parameters method with default values
Add(10,20,30); //a=10,b=20,c=30
Add(10,20) // a=10,b=20,c=0

//Named parameters
 Add(c:20,a:10) // a=10,b=0,c=20
```

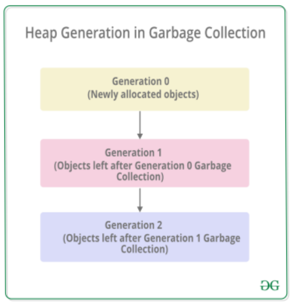
Params

No additional parameters are permitted after the `params` keyword in a method declaration, and only one `params` keyword is permitted in a method declaration

```
public static void UseParams(params int[] list)
```

GC Generrations

The heap memory is organized into 3 generations so that various objects with different lifetimes can be handled appropriately during garbage collection. The memory to each Generation will be given by the Common Language Runtime(CLR) depending on the project size. Internally, Optimization Engine will call the Collection Means Method to select which objects will go into Generation 1 or Generation 2.


Heap Generation in Garbage Collection
Generation 0
(Newly allocated objects)
Generation 1
(Objects left after Generation 0 Garbage Collection)
Generation 2
(Objects left after Generation 1 Garbage Collection)

Generation 0 : All the short-lived objects such as temporary variables are contained in the generation 0 of the heap memory. All the newly allocated objects are also generation 0 objects implicitly unless they are large objects. In general, the frequency of garbage collection is the highest in generation 0.

Generation 1 : If space occupied by some generation 0 objects that are not released in a garbage collection run, then these objects get moved to generation 1. The objects in this generation are a sort of buffer between the short-lived objects in generation 0 and the long-lived objects in generation 2.

Generation 2 : If space occupied by some generation 1 objects that are not released in the next garbage collection run, then these objects get moved to generation 2. The objects in generation 2 are long lived such as static objects as they remain in the heap memory for the whole process duration.

Note: Garbage collection of a generation implies the garbage collection of all its younger generations. This means that all the objects in that particular generation and its younger generations are released. Because of this reason, the garbage collection of generation 2 is called a full garbage collection as all the objects in the heap memory are.released. Also, the memory allocated to the Generation 2 will be greater than Generation 1's memory and similarly the memory of Generation 1 will be greater than Generation 0's memory(Generation 2 > Generation 1 > Generation 0).

Ephemeral Generations and Segments

As mentioned earlier, generations 0 and 1 are short-lived and are known as ephemeral generations. These generations are allocated in a memory segment known as the ephemeral segment. As garbage collection occurs, new segments acquired by the garbage collection become new ephemeral segments, and the segment containing objects surviving past generation 1 becomes the new generation 2 segment.

Garbage Collection Types

There are two types of garbage collection provided by the runtime:

• **Workstation garbage collection:** This is designed for client applications and is the default for stand-alone applications. Workstation GC can be background (covered next) or nonconcurrent.

• **Server garbage collection:** This is designed for server applications that require high throughput and scalability. Server GC can be background or nonconcurrent, just like workstation GC.

Day2

- local function (Inner)(implicitly private)

```
//local functions
      //local func can be defined within another func
       //local func can only be called from the outer func
       //local func can access variables defined in outer func
       //local func is implicitly private
```

class variables are initialized to 0 or null, method level not initialized.

Field Property

```
        private int i;
        public int I  //property
        {
            set
            {
                if (value < 100)
                    i = value;
                else
                    Console.WriteLine("Invalid value for i");
            }
            get
            {
                return i;
            }

        }

      public int P3 { get; set; }//if no validation required

      obj.P3 => obj.getP3();

      obj.P3=10; => obj.setP3(10);
```

Optional Parameter Method

```
public int Add(int a=0, int b=0, int c=0)
        {
            return a + b + c;

        }
```

Add(10,20,30); // Positional Parameters

Add(c:20,a:20,b:30) // Named parameter

Object initializer

```
//object initializer
            Class3 o2 = new Class3(10, 20, 30) { P4 = 40, P5 = 50, P6 = 60, P7 = 70 };
            Class3 o3 = new Class3() { P4 = 40, P5 = 50, P6 = 60, P7 = 70 };
            Class3 o4 = new Class3{ P4 = 40, P5 = 50, P6 = 60, P7 = 70 };

            //Class3 o5 = new Class3(10, 20, 30);


              //Class3 o6 = new Class3 { I = 10,P2= 20, P3= 30 };
              //is same as
              //Class3 o6 = new Class3();
              //o6.I = 10;
              //o6.P2 = 20;
              //o6.P3 = 30;
```

O5 and o6 are not same, o5 calls contr and initialized them as per parameters while o6 calls default parameterless contr and then use setters

```
public class Class3
    {

        private int i;
        public int I  //property
        {
            set
            {
                if (value < 100)
                    i = value;
                else
                    Console.WriteLine("Invalid value for i");
            }
            get
            {
                return i;
            }
        }
        private int p2;
        public int P2
        {
            get { return p2; }
        }
        public int P3 { get; set; }
        public int P4 { get; set; }
        public int P5{ get; set; }
        public int P6{ get; set; }
        public int P7 { get; set; }
        public int P8 { get; set; }
        #endregion


        public Class3(int I = 10, int P2 = 20, int P3 = 30)
        {
            this.I = I;
            this.p2 = P2;
            this.P3 = P3;
            Console.WriteLine("int param cons called");
        }
    }
}
```

Static Members:

Need - Static memebrs can be call using class name and without instatiating class.

e.g: Console - method, Background color-property

Static property- to have single copy with validations.

```
public static int J { set;get; } //property{}
```

```
public static int static_i; //single copy for class
```

Static constructor is implicitly private and parameterless

static Class()

{

    //Initialize static variables here…

    Static_i=10;

}

//create static class

Stastic class can contain only static members,

It cannot be instantiated

Cannot be used as base class

Class loading: 1. When 1st time object is created of the class or

2. When 1st time a static member (field/ method) is accessed

Inheritance in .Net

Why inheritance ? -> code reusability

Single inheritance:

class BaseClass{}

class DerivedClass:BaseClass{}

multiple inheritance (not allowed in .Net)

class BaseClass1{}

class BaseClass2{}

class DerivedClass:BaseClass1,BaseClass2{}

Access Specifier's

private – same class

public – anywhere

protected – same and derived class

internal- same class,same assembly (same project)

protected internal- same and inherited class(derived)+ same assembly

private protected- same class, **derived class in same assembly(derived class must be in same assembly)**

If no main then use library to create project* dll

Add reference from projects to import

Anything in class is private by default

Anything in namespace level internal by default

Anything in method is private

Base class to Dervied class contr calling

```
public DerivedClass(int i, int j):base(i)
```

Day3

```
class Employee
{
        GetNetSalary()
                {…}
        Void show();
}

class Manager : Employee
{
        GetNetSalary(…)   //Overloading
        {…}
        Void new show(); //now this is hiding base class method
}
```

1.Derived class can **overload** base class method if derived class wants diff implementation but in this case there will be 2 methods availble in dervied class.

2.derived class can hide the base class method with same signature(with **new keyword** not mandatory,compiler gives warning)

This is known as method **hiding**(any method can be hidden)

Only dervied class method can be access.

Early binding method will be called based on reference irrespective of object class

3. derived class can **override** the base class method with same signature.

***Only a virtual method can be overridden**

- virutal methods are late bound, **run time binding** (polymorphism)

  [In java all methods are virtual, this make .Net faster than java as all methods are not late bound(one of the reason)]

- Both derived and base class methods can be accessed here by ref.
- **All overidden methods are virtual**

By default method hiding will be done for both virtual and non-virtual methods if not specifide explicitly using override keyword for overriding.

Diff Between virtual and non-virtual

-Late Binding	Early Binding
-override keyword	new keyword

```
[//1.derived class can overload base class method
//derived class method has diff signature
//derivedobj.GetNetSalary(); --base class method
//derivedobj.GetNetSalary(...); --derived clas method

//2.derived class can hide the base class method
//derived class method has same signature
//derivedobj.GetNetSalary(); --derived clas method
//ANY method can be hidden


//3.derived class can override the base class method
//derived class method has same signature
//derivedobj.GetNetSalary(); --derived clas method
//only a VIRTUAL method can be overridden


//Virtual methods are late bound
//run time binding/ run time polymorphism]
```

	Abstract Class	Sealed Class
Can be instantiated	NO	YES
Can be used as a base class	YES	NO

Method hiding and overrding e.g.

```
class Base
    {
        public void show()
        {
            Console.WriteLine("Base class show method");
        }

        virtual public void display()
        {
            Console.WriteLine("Base class display method");
        }

    }



class Derived : Base
    {
        public new void show()
        {
            Console.WriteLine("Derived class show method(method hiding)");
        }

        public override void display()
        {
            Console.WriteLine("Derived class display method(overriding)");
        }
    }
```

If we want to avoid further overriding we make it **sealed** method

```
  public sealed override void display()
```

`only virtual methods can be sealed`

Abstract Class: Cannot be instantiated can be used as base class, need not have any astract method,if class have 1 or more abstarct method it has to or must be abstract.

Interface -All methods are implicitly public and abstarct (cannot specify externally)

```
        Class1 o = new Class1();
        //method 1
            o.Insert();

        //method 2
        IDbFunctions oIDb;
           oIDb = o;
           oIDb.Insert();

        //method 3
        ((IDbFunctions)o).Insert();

        //method 4
         (o as IDbFunctions).Insert();


public interface IDbFunctions
    {
        void Insert();  //public abstract  (implicit)
        void Update(); //public abstract
        void Delete();  //public abstract

    }

public interface IFileFunctions
    {
        void Open();
        void Close();
        void Delete();
    }

public class Class1 : IDbFunctions, IFileFunctions
    {
        void IDbFunctions.Delete()
        {
            Console.WriteLine("idb.Delete");
        }
        public void Insert()
        {
            Console.WriteLine("Insert");
        }
        public void Update()
        {
            Console.WriteLine("Update");
        }
        public void Display()
        {
            Console.WriteLine("Display");
        }

        //void IFileFunctions.Open()
        //{
        //    Console.WriteLine("Open");
        //}

        //void IFileFunctions.Close()
        //{
        //    Console.WriteLine("Close");
        //}

        void IFileFunctions.Delete()
        {
            Console.WriteLine("ifile.Delete");
        }

        public void Open()
        {
            Console.WriteLine("Open");
        }

        public void Close()
        {
            Console.WriteLine("Close");
        }
    }
}
```

Default method functionality available with core only and not with .net framework

All classes and thiere variants(classes,interfaces,delegates,arrays,object class,string class)
are examples of **ref type**

All structs and enums are **vlaue type**

Data types

```
// Int32 sruct

            byte b1;-struct System.Byte repersents 8-bit unsinghed integer
            sbyte b2; //Sbyte- 1 repersents 8-bit singhed integer
            char ch; //Char  -2
            short sh1; //Int16  -2
            ushort sh2; //UInt16  -2
            int i1; //Int32 //4
            uint i2;//UInt32 //4
            long l1;  //Int64  -8
            ulong l2; //UInt64 -8
            float f;  //Single  //4
            double d; //Double  //8
            decimal d2; //Decimal //16
            bool b;  //Boolean

            string s;  //class System.String
            object o; // Object
```

Call by value and ref

```
           int i=10;
            int j;

            Init(out i, out j); // for intitializing
            //Swap(ref i,ref j); //call By ref
            Swap2( i, j);
            Console.WriteLine(i);//20
            Console.WriteLine(j);
            Console.ReadLine();

static void Swap(ref int i, ref int j)

static void Init(out int i, out int j) // descard the i value i.e 10
      {
            // must initialize the out varibles in this function other
      wise complier error.
            i=20;
            j=50;
      }
```

- Out is similar to ref - changes made in func reflect back in calling code
- the initial value is discarded,need not have initialize value while passing but while returning all var should be intialize
- out variables MUST be initialized in the function

Day 3

```
Class1 o = new Class1();
            o.i = 100;
Int j =20;
            //DoSomething1(o,j);
            //DoSomething2(o);
        DoSomething3(ref o);

      // DoSomething3(in o); //we can’t change value



static void DoSomething3(ref Class1 obj)  //obj = o
        {
          //changes made in func (obj pointing to some other block) is reflected
      in calling code o

            obj = new Class1();
            obj.i = 200;
        }


  static void DoSomething1(in Class1 obj, in int i) //obj =o
        {

            //all in variables are readonly – can’t change
            //i = 10; ---->error
            //obj = new Class1(); ----> complier error
            obj.i = 200;
        }
```

To Set Property as read

```
private int empNo;
        public int EmpNo
        {
            get { return empNo; }
            //property accessor
            //can only be given for one out of get/set
            //access can only be reduced, not increased
            Order of access
            //public
            //protected internal
            //protected   OR   internal
            //private protected
            //private
            private set { empNo = value; }
        }
```

```
public class Emp3
    {
        public int EmpNo
        {
            get;
        }
        public Emp3()
        {
            EmpNo = 10;  //do not call set instead treat EmpNo as field
        }
    }
```

Value Types:

1. **Structs**

```
//same as class with below diffrences
//structs are Value Types - stored on stack. Faster than Heap operations
    //No Inheritance allowed in structs
    //Parameterless constructor not allowed in structs
    //fields cannot be initialize outside constr in stuct and must be initialize in
       parameterise constr
    //all fields must be initialize

class Program
    {
        static void Main1()
        {
            int i = new int();
            //int i=0;
            Console.WriteLine(i);
            Console.ReadLine();
        }
        static void Main2()
        {
            MyPoint p = new MyPoint(10, 20, 30);
            Console.WriteLine(p.A);
            Console.ReadLine();
        }
    }


    public struct MyPoint
    {
        public int A
        {
            get; set;
        }
        public int X, Y;
        private int b;
        public int B
        {
            get { return b; } set { b = value; }
        }
        public MyPoint(int X = 0, int Y = 0, int A = 0)
        {
            Console.WriteLine("cons called");
            this.b = 0;
            this.X = X;
            this.Y = Y;
            this.A = A;
        }}
```

2. Enums: Enums are constant type

```
public enum TimeOfDay //: long  //by default int32 //cannot take decimal
values
    {
        Morning = 10,
        Afternoon=20,
        Evening,
        Night
       }
```

//Boxing a value type

```
//boxing is overhead and should be minimized
static void Main2()
        {
            //Display1(0);
            Display2(TimeOfDay.Morning);
            Console.ReadLine();
        }


static void Main()
        {
            object o;
            int i = 100;
            o = i;  //ref type = value type
            //BOXING a value type

            int j;
            j =(int) o;
            //UNBOXING

            Console.ReadLine();
        }
```

//Nullable Types

After datatype used ? for nullable

```
static void Main(string[] args)
        {
            //int a;
            //a = 100;
            //a = null;  //not possible
            //a = ReadValueFromDb(); //can contain null value

            int? i = 10;  // i is nullable

            i = null;

            int j;
        // j=(int) i;  Will throw exception :
    InvalidOperationException

            if (i != null)
                j = (int)i;
            else
                j = 0;

            if (i.HasValue)  // if i is not null, if it has value
                j = i.Value;
            else
                j = 0;

            j = i.GetValueOrDefault(); // if i is null then j=0
            j = i.GetValueOrDefault(10); // if i is null then j=10
            j = i ?? 10; //null coalescing operator  (same as above)

            Console.WriteLine(j);
            Console.ReadLine();
        }
    }
```

Idisposable interface : provides mechanism to for relaesing
                           unmanaged resources.

Makes use of using which makes auto call to Dispose and obj not available after using block.

```
static void Main2()
        {
            using (Class1 o = new Class1())
            {
                o.Display();
            } // at end auto calls Dispose()

        }

public class Class1 : IDisposable
    {
        public Class1()
        {
            //open file here
            //open db here
            Console.WriteLine("class1 constructor");
        }
        public void Display()
        {
            Console.WriteLine("Display called");
        }
        public void Dispose()
        {
            //close file
            //close db
            CheckForDisposed()
            Console.WriteLine("Dispose code called. Write code here
instead of Destructor");
        }
    }

private void CheckForDisposed()
        {
            if (isDisposed) //isDisposed bool var
                throw new ObjectDisposedException("Class1");

        }
```

Arrays:

```
int[] arr = new int[5];


      for (int i = 0; i < arr.Length; i++)
            {
                Console.Write("enter value for arr[{0}]:", i);
                arr[i] = int.Parse(Console.ReadLine());
                //arr[i] = Convert.ToInt32(Console.ReadLine());
                //{0} : 0 is placeholder replces with 1st param after (,)
                //same as %d in C
            }
            for (int i = 0; i < arr.Length; i++)
            {
                Console.WriteLine("value for arr[{0}] is {1}", i, arr[i]);
            }

            Console.WriteLine();
            foreach (int item in arr)
            {
                Console.WriteLine(item);
            }
```

```
static void Main3()
        {
            int[,] arr = new int[3, 2];

            //arr[0,0] arr[0,1]
            //arr[1,0] arr[1,1]
            //arr[2,0] arr[2,1]

            Console.WriteLine(arr.Length);  //6
            Console.WriteLine(arr.Rank);  //number of dimensions - 2

            Console.WriteLine(arr.GetLength(0));  //3
            Console.WriteLine(arr.GetLength(1));  //2

            Console.WriteLine(arr.GetUpperBound(0));  //2
            Console.WriteLine(arr.GetUpperBound(1));  //1

            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    //Console.Write("Enter arr[{0},{1}] : ", i, j);
                    Console.Write($"Enter arr[{i},{j}] : ");
                  //string interpolation

                    arr[i, j] = int.Parse(Console.ReadLine());
                }
            }

            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    Console.WriteLine($"The value of arr[{i},{j}] is
{arr[i, j]} ");  //string interpolation

                }
            }
            foreach (int item in arr)
            {
                Console.WriteLine(item);
            }
            Console.ReadLine();
        }
```

String Interpolation: `($"Enter arr[{i},{j}] : ")`

```
//array of totalmarks for 5 students
//int[] arr = new int[5];

//array of totalmarks for 3 batches 5 students
//int[,] arr = new int[3, 5];

//array of totalmarks for 2 centres, 3 batches 5 students
//int[,,] arr = new int[2, 3, 5];

//array of totalmarks for 4 cities, 2 centres, 3 batches 5 students
//int[,,,] arr = new int[4, 2, 3, 5];

//jagged
            int[][] arr = new int[4][];
            arr[0] = new int[3]; // arr[0][0] arr[0][1] arr[0][2]
            arr[1] = new int[4]; // arr[1][0] arr[1][1] arr[1][2]
arr[1][3]
            arr[2] = new int[2];//  arr[2][0] - arr [2][1]
            arr[3] = new int[3];//  arr[3][0] arr[3][1] arr[3][2]
```

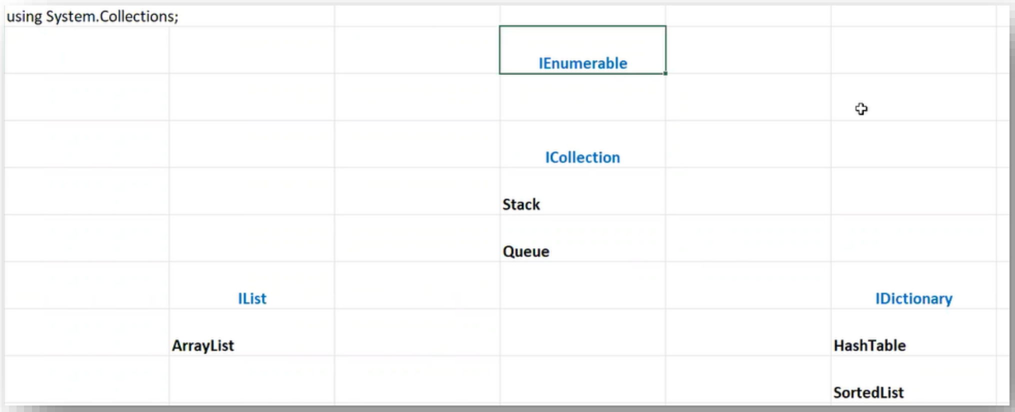
Array of References `Employee[] arrEmps = new Employee[4];`

Generics in .Net

```
class MyStack<T>
    //where T : class  //T must be a reference type
    //where T : struct  //T must be a value type
    //where T : ClassName  //T must be the class specified or its derived
                class
    //where T : InterfaceName  //T must be a class that implements the
                specified interface
    //where T : new() //T must have a no param constructor
    //where T : ClassName,InterfaceName,new()
{
   public T Data { get; set; }

}
```

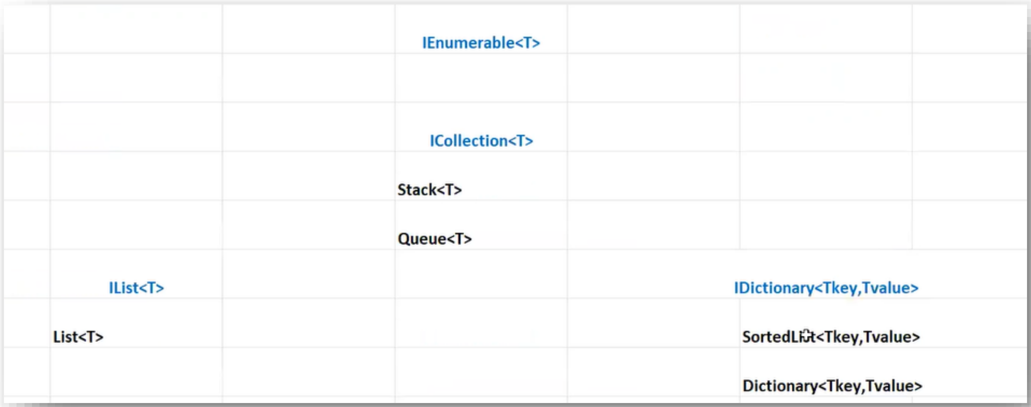
Collections in .Net

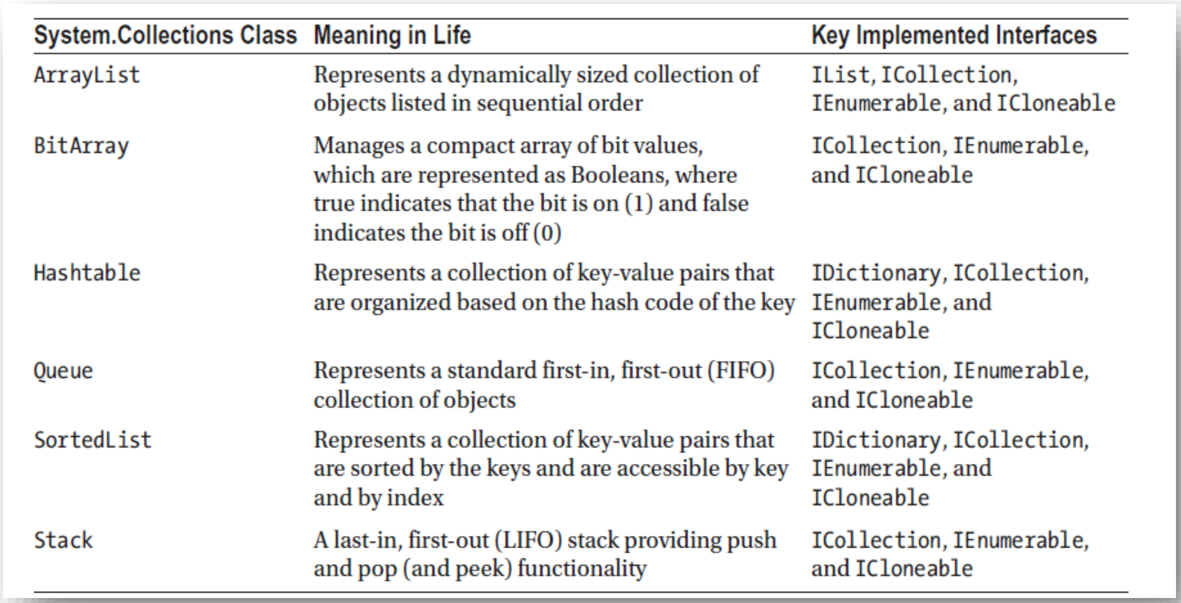
Non generic collection, generic
(System.Collections, System.Collections.Generic)
Non generic does boxing which is time consuming
IEnumaration<T>



```
using System.Collections;
```

Generics





System.Collections Class	Meaning in Life	Key Implemented Interfaces
`ArrayList`	Represents a dynamically sized collection of objects listed in sequential order	`IList`, `ICollection`, `IEnumerable`, and `ICloneable`
`BitArray`	Manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0)	`ICollection`, `IEnumerable`, and `ICloneable`
`Hashtable`	Represents a collection of key-value pairs that are organized based on the hash code of the key	`IDictionary`, `ICollection`, `IEnumerable`, and `ICloneable`
`Queue`	Represents a standard first-in, first-out (FIFO) collection of objects	`ICollection`, `IEnumerable`, and `ICloneable`
`SortedList`	Represents a collection of key-value pairs that are sorted by the keys and are accessible by key and by index	`IDictionary`, `ICollection`, `IEnumerable`, and `ICloneable`
`Stack`	A last-in, first-out (LIFO) stack providing push and pop (and peek) functionality	`ICollection`, `IEnumerable`, and `ICloneable`

ArrayList example:
```
static void Main()
        {
            ArrayList o = new ArrayList();
            o.Add(10);
            o.Add("vikram");
            o.Add(10.34);
            o.Insert(0, "new");
            Console.WriteLine(o.Count);
            o.Remove("vikram");
            o.RemoveAt(0);
            o.RemoveRange(0, 3);
            //o.AddRange(o2);
            //o.InsertRange(0, o2);
            //o.IndexOf
            //o.LastIndexOf
            //o.BinarySearch
            //o.Clear
            //bool ans = o.Contains(10);
            //o.CopyTo(arr)
            ArrayList o2 = o.GetRange(0, 3);
            //o.SetRange(0,o2)
            object[] arr2 = o.ToArray();

            //o.CopyTo(arr2);
            //o.Capacity = large_number
            //add elements in a loop
            //o.TrimToSize();

            foreach (object item in o)
            {
                Console.WriteLine(item);
            }
        }
```

Day5

Collections:

HashTable

```
  //Hashtable o = new Hashtable();
            SortedList o = new SortedList();
            o.Add(1, "Vikram");
            o.Add(2, "Shweta");
            o.Add(3, "Harsh");

            o[1] = "changed"; //get or set value with specified key
            o[4] = "Ananya";

            o.Remove(1);

            bool ispresent = o.Contains(2);
            ispresent = o.ContainsKey(2);
            ispresent = o.ContainsValue("Harsh");
            //o.GetByIndex  --gets the value at index
            //o.GetKey  --
            // o.GetEnumerator -- IDictionaryEnumator
            IList lst = o.GetKeyList();
            //o.GetValueList -- IList
            //o.IndexOfKey
            //o.IndexOfValue
            //o.Keys          --ICollection
            //o.SetByIndex
            //o.Values        --ICollection
            foreach (DictionaryEntry item in o)
            {
                Console.WriteLine(item.Key);
                Console.WriteLine(item.Value);
            }
```

All Non generic collections holds the collection of object

Generics

```
      SortedList<int, string> o = new SortedList<int, string>();

                  o.Add(1, "Vikram");
                  o.Add(2, "Shweta");
                  o.Add(3, "Harsh");

            o[1] = "changed";
            o[4] = "Ananya";

            o.Remove(1);
            foreach (KeyValuePair<int,string> item in o)
            {
                Console.WriteLine(item.Key);
                Console.WriteLine(item.Value);
            }
```

Delegates:

```
namespace Delegates
{
    //step 1 : create a delegate class having the same signature as the
method to call
    public delegate void Del1();
    //Object
    //Delegate
    //MulticastDelegate
    //Del1
    public delegate int DelAdd(int a, int b);
    //public delegate int DelAdd2(int a = 0, int b = 0, int c = 0);
    class Program
    {

        static void Main1()
        {
            //step 2: create an object of the delegate class, pass it
function name as a paramter
            Del1 objDel = new Del1(Display);

            //step 3 : call the function via the delegate object
            objDel();
            Console.ReadLine();
        }


static void Main2()
        {
            Del1 objDel = Display;
            objDel();
            Console.ReadLine();
        }
        static void Main3()
        {
            Del1 objDel = Display;
            objDel();
            objDel = Show;
            objDel();
            Console.ReadLine();
        }
        static void Main4()
        {
            Del1 objDel = Display;
            objDel();

            Console.WriteLine();
            objDel += Show;
            objDel();
            Console.WriteLine();
            objDel += Display;
            objDel();

        Console.WriteLine();
        objDel -= Display;//here + for add delegate function and - for remove delegate
            objDel();

            Console.ReadLine();
        }
        static void Main5()
        {
            Del1 o =(Del1) Delegate.Combine(new Del1(Display), new Del1(Show), new
Del1(Display));
            o();

            Console.WriteLine();
            //o = (Del1)Delegate.Remove(o, new Del1(Display));
            //o = (Del1)Delegate.RemoveAll(o, new Del1(Display));
            o();
            Console.ReadLine();
        }
        static void Main6()
        {
            DelAdd objDel = Add;
            Console.WriteLine(objDel(10,20));
            Console.ReadLine();
        }
        static void Main7()
        {
            Del1 objDel = Class1.Display;
            objDel();

            Class1 o = new Class1();
            objDel = o.Show;
            Console.ReadLine();
        }
        //to do - try calling a multicast delegate for a function that has return
values.
        static void Display()
        {
            Console.WriteLine("Display");
        }
        static void Show()
        {
            Console.WriteLine("Show");
        }
        static int Add(int a, int b)
        {
            return a + b;
        }
        //static int Add2(int a=0, int b=0, int c=0)
        //{
        //    return a + b;
        //}
    }
    public class Class1
    {
        public static void Display()
        {
            Console.WriteLine("Display");
        }
        public void Show()
        {
            Console.WriteLine("Show");
        }
    }
}

//Call Function Passed as a Parameter
namespace Delegates2
{
    public delegate int DelAdd(int a, int b);
    class Program
    {
        static void Main()
        {
            Console.WriteLine(CallFunctionPassedAsAParameter(Add, 20, 10));
            Console.WriteLine(CallFunctionPassedAsAParameter(Subtract, 20, 10));
            Console.WriteLine(CallFunctionPassedAsAParameter(Multiply, 20, 10));
            Console.WriteLine(CallFunctionPassedAsAParameter(Divide, 20, 10));
            Console.ReadLine();
        }
        static int CallFunctionPassedAsAParameter(DelAdd objFunc, int a, int b)
        {
            return objFunc(a,b);
        }

        static int Add(int a, int b)
        {
            return a + b;
        }
        static int Subtract(int a, int b)
        {
            return a - b;
        }
        static int Multiply(int a, int b)
        {
            return a * b;
        }
        static int Divide(int a, int b)
        {
            return a / b;
        }

    }

}
```

GetInvocationList() -to get values of all functions in multicast delegeate

There are delegates provided by .Net which can be used directly

1. **Action – delegate for void returning method with no parameters;**
   Action o1 = Display;
   public void Display(){…}
   Action<> takes multiple params(16) returning void

2. **Func<>** – delegate for method having return type and accepting multiple parameters,

   Func<int, bool>- int–parameter type, bool method return type always at last<>

```
Func<int, bool> o4 = IsEven;

public bool IsEven(int a)
    {
        if (a % 2 == 0)
            return true;
        else
            return false;
    }
```

3. **Predicate**–delegate for method having boolean return type and accepting one one, <paramsType>

   Predicate<int> o5 = IsEven;

Anonymous Method:

```
Action a=delegate(){

};
```

With Lambda

```
Func<int,int> f1= a=>a*2;

f1(10);  // 20
```

Exception handling in .Net

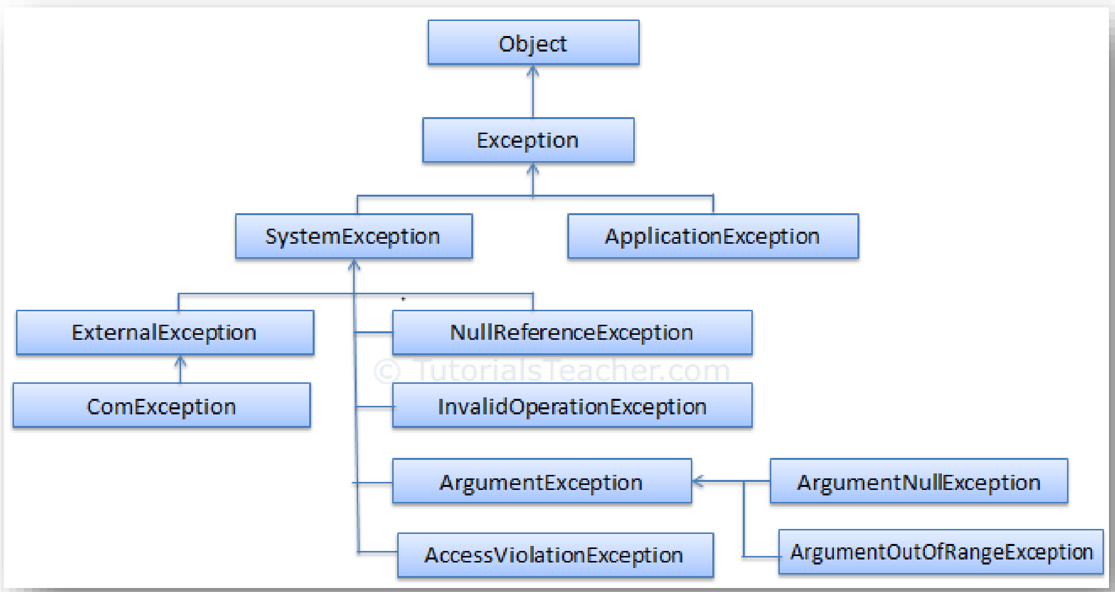


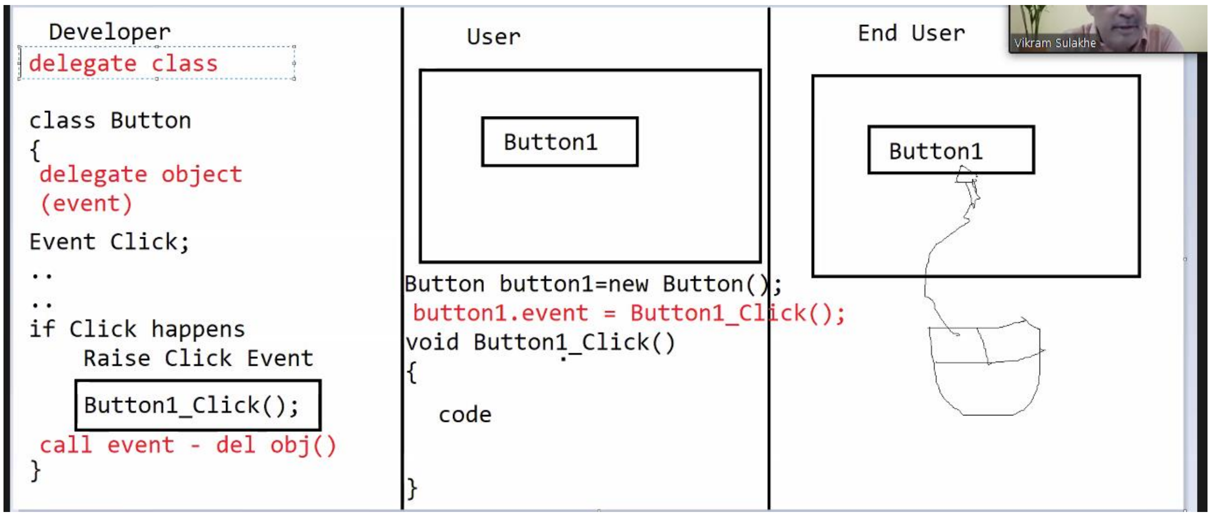
Table 7-1. Core Members of the System.Exception Type

System.Exception Property	Meaning in Life
`Data`	This read-only property retrieves a collection of key-value pairs (represented by an object implementing `IDictionary`) that provide additional, programmer-defined information about the exception. By default, this collection is empty.
`HelpLink`	This property gets or sets a URL to a help file or website describing the error in full detail.
`InnerException`	This read-only property can be used to obtain information about the previous exceptions that caused the current exception to occur. The previous exceptions are recorded by passing them into the constructor of the most current exception.
`Message`	This read-only property returns the textual description of a given error. The error message itself is set as a constructor parameter.
`Source`	This property gets or sets the name of the assembly, or the object, that threw the current exception.
`StackTrace`	This read-only property contains a string that identifies the sequence of calls that triggered the exception. As you might guess, this property is useful during debugging or if you want to dump the error to an external error log.
`TargetSite`	This read-only property returns a `MethodBase` object, which describes numerous details about the method that threw the exception (invoking `ToString()` will identify the method by name).

```
static void Main1() //simple try block with catch
        {
            Class1 obj = new Class1();
            try
            {
                obj = null;
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = 100 / x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }
            catch
            {
                Console.WriteLine("Exception occurred");
            }
            Console.ReadLine();
        }
        static void Main2()//try with multiple catch blocks
        {
            Class1 obj = new Class1();
            try
            {
                obj = null;
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = 100 / x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("DBException occurred");
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("NRException occurred");
            }
            catch (FormatException ex)
            {
                Console.WriteLine("FormatException occurred");
            }
            Console.ReadLine();
        }
        static void Main3()// catching base class exceptions
        {
            Class1 obj = new Class1();
            try
            {
                obj = null;
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = 100 / x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }

            //catch (SystemException ex)
            catch (FormatException ex)
            {
                Console.WriteLine("FormatException occurred");
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("NRException occurred");
            }
            catch (DivideByZeroException ex)
            //catch (ArithmeticException ex)
            //catch (SystemException ex) //base class exception has to caught after
    derived class exceptions
            {
                Console.WriteLine("DBException occurred");
            }
            catch (Exception ex) //catches all unhandled exceptions
            {
                Console.WriteLine(ex.TargetSite);
                Console.WriteLine(ex.Source);
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }
            Console.ReadLine();
        }
        static void Main4()// finally block
        {
            Class1 obj = new Class1();
            try
            {
                //obj = null;
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = 100 / x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }

            catch (FormatException ex)
            {
                Console.WriteLine("FormatException occurred");
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("NRException occurred");
            }
            //catch (DivideByZeroException ex)
            //catch (ArithmeticException ex)
            catch (SystemException ex) //base class exception has to caught after
derived class exceptions
            {
                Console.WriteLine("DBException occurred");
            }
            catch (Exception ex) //catches all unhandled exceptions
            {
                Console.WriteLine(ex.Message);
            }
            //finally runs when Exception does not occur,
            //or Exception occurs and is handled or
            //Exception occurs and is NOT handled
            finally
            {
                Console.WriteLine("finally");
            }
            Console.ReadLine();
        }

        static void Main5()// nested try block
        {
            Class1 obj = new Class1();
            try
            {
                //obj = null;
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = 100 / x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }
            catch (FormatException ex)
            {
                try
                {
                    Console.WriteLine("FormatException occurred. Enter only numbers");
                    int x = Convert.ToInt32(Console.ReadLine());
                    obj.P1 = 100 / x;
                    Console.WriteLine(obj.P1);
                }
                catch
                {
                    Console.WriteLine("nested try catch example");
                }
                finally
                {
                    Console.WriteLine("nested try finally example");
                }
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("NRException occurred");
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine("DBException occurred");
            }
            catch (Exception ex) //catches all unhandled exceptions
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Console.WriteLine("outer finally");

            }
            Console.ReadLine();
        }
    }

}

namespace ExceptionHandling2
{
    class Program
    {
        static void Main()
        {
            Class1 obj = new Class1();
            try
            {
                int x = Convert.ToInt32(Console.ReadLine());
                obj.P1 = x;
                Console.WriteLine(obj.P1);
                Console.WriteLine("No Exceptions");
            }
            catch (FormatException ex)
            {
                Console.WriteLine("FormatException occurred");
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("NRException occurred");
            }
            catch (InvalidP1Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            catch (SystemException ex)  //all exceptions thrown by .net base classes
            {
                Console.WriteLine(ex.Message);
            }

            catch (ApplicationException ex) //all user defined exceptions
            {
                Console.WriteLine(ex.Message);
            }
            catch (Exception ex) //all other exceptions
            {
                Console.WriteLine(ex.Message);
            }


            Console.ReadLine();
        }
    }

    public class Class1
    {
        private int p1;
        public int P1
        {
            get
            {
                return p1;
            }
            set
            {
                if (value < 100)
                    p1 = value;
                else
                {
                    //Console.WriteLine("invalid P1");   //DONT EVER DO THIS
                    //Exception ex;
                    //ex = new Exception();
                    //throw ex;

                    //Exception ex;
                    //ex = new Exception("invalid P1");
                    //throw ex;

                    //throw new Exception("Invalid P1");

                    throw new InvalidP1Exception("Invalid P1");
                }
            }
        }
    }

    public class InvalidP1Exception : ApplicationException
    {
        public InvalidP1Exception(string message) : base(message)
        {
        }}}
```

Day6

Events: event handling using delegates. all events return void .
 In c# event handling is by default dynamic, at run time we can add or remove methods from events.



```
Developer
delegate class

class Button
{
 delegate object
 (event)
Event Click;
..
..
if Click happens
    Raise Click Event

    Button1_Click();
 call event - del obj()
}
```

```
User

Button1

Button button1=new Button();
 button1.event = Button1_Click();
void Button1_Click()
{
   code

}
```

```
End User

Button1
```

Delegate class use for event handling

1.create delegate class matching the event handling function signature
2.return type always void as events return void
public delegate void Del1();

Declare the event //event is delegate object

public event InvalidP1EventHandler InvalidP1;//event for marking only and not mandatory

3.raise the event
4.add methods to delegate

```
namespace EventHandling
{
    class Program
    {
        //static void Main()
        //{
        //    Class1 o = new Class1();
        //    o.InvalidP1 += o_InvalidP1;
        //    o.P1 = 1000;
        //    Console.ReadLine();
        //}
        //static void o_InvalidP1()
        //{
        //    Console.WriteLine("event handled here");
        //}
        static void Main1()
        {
            Class1 o = new Class1();
            o.InvalidP1 += O_InvalidP1;
            o.InvalidP1 += Handler2;
            o.P1 = 200;

            Console.WriteLine();
            o.InvalidP1 -= Handler2;
            o.P1 = 200;

            Console.WriteLine();
            o.InvalidP1 -= O_InvalidP1;
            o.P1 = 200;
            Console.ReadLine();
        }

        private static void O_InvalidP1()
        {
            Console.WriteLine("event handled here");
        }
        private static void Handler2()
        {
            Console.WriteLine("event also handled here");
        }
    }
    //step1 : create a delegate class matching the event handling function signature
    public delegate void InvalidP1EventHandler();

    public class Class1
    {
        //step2 : declare the event
        //event is a delegate object
        public event InvalidP1EventHandler InvalidP1;


        private int p1;
        public int P1
        {
            get
            {
                return p1;
            }
            set
            {
                if (value < 100)
                    p1 = value;
                else
                {
                    //step 3: raise the event
                    if(InvalidP1!=null)
                        InvalidP1();
                }
            }
        }
    }
```

```
}


//event with parameters

namespace EventHandling2
{
    class Program
    {

        static void Main()
        {
            Class1 o = new Class1();
            o.InvalidP1 += O_InvalidP1;
            o.P1 = 200;
            Console.ReadLine();
        }

        private static void O_InvalidP1(int InvalidValue)
        {
            Console.WriteLine($"invalidp1 event raised....{InvalidValue}");
        }
    }
    //step1 : create a delegate class matching the event handling function signature
    public delegate void InvalidP1EventHandler(int InvalidValue);

    public class Class1
    {
        //step2 : declare the event
        //event is a delegate object
        public event InvalidP1EventHandler InvalidP1;


        private int p1;
        public int P1
        {
            get
            {
                return p1;
            }
            set
            {
                if (value < 100)
                    p1 = value;
                else
                {
                    //step 3: raise the event
                    if (InvalidP1 != null)
                        InvalidP1(value);
                }
            }
        }
    }
}
```

Multithreading using delegates:

Assynch

```
namespace AsyncCodeWithDelegates1
{
    class Program
    {
        static void Main1()
        {
            Action oDel = Display;
            Console.WriteLine("Before");
            //oDel();
            oDel.BeginInvoke(null, null);  //Display called on a separate
thread(parallel excution)
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static void Display()
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display");
        }
    }
}

namespace AsyncCodeWithDelegates2
{
    class Program
    {
        static void Main2()
        {
            Action<string> oDel = Display;
            Console.WriteLine("Before");
            oDel.BeginInvoke("abc", null, null);// here we pass the parameters(what
ever you want)
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static void Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
        }
    }
}

namespace AsyncCodeWithDelegates3
{
    class Program
    {
        static void Main3()
        {
            Func<string,string> oDel = Display;
            Console.WriteLine("Before");
            oDel.BeginInvoke("abc", CallbackFunction, null); //here we call the
delegates as well as callback function .
            //oDel.BeginInvoke("abc", new AsyncCallback(CallbackFunction), null);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }
        static void CallbackFunction(IAsyncResult ar)
        {
            Console.WriteLine("callback func called");
        }
    }
}

//move oDel to the class level
namespace AsyncCodeWithDelegates4
{
    class Program
    {
        static Func<string, string> oDel = Display;
        static void Main1()
        {
            Console.WriteLine("Before");
            IAsyncResult ar = oDel.BeginInvoke("abc", CallbackFunction, null);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }
        static void CallbackFunction(IAsyncResult ar)
        {
            Console.WriteLine("callback func called");
            string retval = oDel.EndInvoke(ar);  //retrive the return value of
delegate function.
            Console.WriteLine($"return value is {retval}");
        }
    }
}


//make callback func as a local func/anon method
namespace AsyncCodeWithDelegates5
{
    class Program
    {
        static void Main1()
        {
            Func<string, string> oDel = Display;
            Console.WriteLine("Before");
            IAsyncResult ar1 = oDel.BeginInvoke("abc", delegate(IAsyncResult ar)
            {
                Console.WriteLine("callback func called");
                string retval = oDel.EndInvoke(ar);
                Console.WriteLine($"return value is {retval}");
            }, null);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }

    }
}

//pass oDel as the last parameter in BeginInvoke. Access it as ar.AsyncState in the
callback method
namespace AsyncCodeWithDelegates6
{
    class Program
    {
        static void Main1()
        {
            Func<string, string> oDel = Display;
            Console.WriteLine("Before");
            //oDel.BeginInvoke("abc", CallbackFunction, "extra data");
            oDel.BeginInvoke("abc", CallbackFunction, oDel);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }
        static void CallbackFunction(IAsyncResult ar)
        {
            Console.WriteLine("callback func called");
            //string lastparam = ar.AsyncState.ToString();
            //Console.WriteLine(lastparam);

            Func<string, string> oDel = (Func<string, string>)ar.AsyncState;
            string retval = oDel.EndInvoke(ar);
            Console.WriteLine($"return value is {retval}");

        }
    }
}


//Access it as  result.AsyncDelegate in the callback method. result is of type
AsyncResult
namespace AsyncCodeWithDelegates7
{
    class Program
    {
        static void Main1()
        {
            Func<string, string> oDel = Display;
            Console.WriteLine("Before");
            oDel.BeginInvoke("abc", CallbackFunction, null);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }
        static void CallbackFunction(IAsyncResult ar)
        {
            AsyncResult result =(AsyncResult) ar;

            Console.WriteLine("callback func called");
            Func<string, string> oDel = (Func<string, string>)result.AsyncDelegate;
            string retval = oDel.EndInvoke(ar);
            Console.WriteLine($"return value is {retval}");

        }
    }
}


//use ar.AsyncWaitHandle.WaitOne() --- wait for the thread to complete -- like
Thread.Join
namespace AsyncCodeWithDelegates8
{
    class Program
    {
        static void Main()
        {
            Func<string, string> oDel = Display;
            Console.WriteLine("Before");
            IAsyncResult ar = oDel.BeginInvoke("abc", null, null);
            Console.WriteLine("After");

            //while (!ar.IsCompleted) ;// it is check frequently to complete
            ar.AsyncWaitHandle.WaitOne();// here wait to complete thread excuation

            string retval = oDel.EndInvoke(ar);
            Console.WriteLine($"return value is {retval}");
            Console.ReadLine();
        }
        static string Display(string s)
        {
            Thread.Sleep(5000);
            Console.WriteLine("Display" + s);
            return s.ToUpper();
        }

    }
}
```

LINQ Query:

Language features
Implicit/explicit variables

```
Static void Main(){
   int i=100; //explicit
   var j=200;  //implicit variable (data type can be anything based on value assigned)
   //var j="abc"; error
}
```

Implicit variables:

1.**implicit variable** (data type can be anything based on value assigned)
2.once value is assigned var datatype cannot be change
3.Mostly used in Link queries to avoid changes of left side when query changes,or to avoid complexity of query [multiple data types return by query]
4.**only used as local variables.**
5.cant be used for class level vars,fn params and return types

Anonumous types:

type can be a class,struct,interface,delegate,enum

Var o=new {a=10,b=20}; //anonymous type and obj of anonymous class

```
  var obj = new { a = 10, b = "aaa", c = true };
//obj.GetType() : <>f__AnonymousType0`3[System.Int32,System.String,System.Boolean]

//here class type is a
```

Partial class:

```
// partial classes used for multi devloper working on same class.

//partial classes
namespace LanguageFeatures5
{
    //PARTIAL CLASSES

    //partial classes must be in the same assembly
    //partial classes must be in the same namespace
    //partial classes must have the same name

    public partial class Class1
    {
        public int i;
    }
    public partial class Class1
    {
        public int j;
    }
    public class Program
    {
        public static void Main1()
        {
            Class1 o = new Class1();

        }
    }
}
namespace LanguageFeatures5
{
    public partial class Class1
    {
        public int k;
    }
}
```

Partial methods: acts like placeholder

1.Partial methods can only be defined within a partial class.
2.Partial methods must return void.
3.Partial methods can be static or instance level.
4.Partial methods cannnot have out params
5.Partial methods are always implicitly private.

```
//partial methods
namespace PartialMethods
{
    public class MainClass
    {
        public static void Main1()
        {
            Class1 o = new Class1();

            Console.WriteLine(o.Check());

            Console.ReadLine();
        }
    }

    public partial class Class1
    {
        private bool isValid = true;
        partial void Validate();
        public bool Check()
        {
            //.....
            Validate();   // calling the method if no code all traces will be removed during
compilation
            return isValid;
        }
    }

    public partial class Class1
    {
        partial void Validate()
        {
            //perform some validation code here
            isValid = false;
        }
    }
```

}

Extension Methods: can be define in static class

```
1. create a static method in the class

2. first parameter should be the type for which you are writing the
extention method

3.precede the 1st parameter with the 'this' keyword

4.if you define an extension method for a base class, it is also available
to all derived classes.

5.if you define an extension method for an interface, it is also available
to all classes that implement that interface.
```

Public static void display(this int i)   ------> this display method add in int datatype(we can access it by int i.e. --- > int.display())

```
{
 Console.WriteLine(i);
}

Main()
{
  Employee e;
  String s;
}

Employee.Xyz(){….}
String.Prq(){…}
```

```
Int i=10;
Int sum=i.Add(5);
```

Public static int Add(this int I,int j)

{ Return i+j;}

This is same as

Int sum=Class1.Add(I,j);

Static class Class1{

Int sum=i.Add(5);

```
public static int Add(this int i, int j)
        {
            return i + j;
        }
```

```
namespace ExtensionMethods

{

  public class MainClass

  {

    public static void Main1()

    {

      int i = 100;

      i = i.Add(5);

      i.Display();

      i.ExtMethodForBaseClass();

      string s = "abcd";

      s.Show();

      s.ExtMethodForBaseClass();

      Console.ReadLine();

    }

    public static void Main2()

    {

      int i = 100;

      //i = i.Add(5);

      //i.Display();

      Class1.Add(i, 5);

      Class1.Display(i);


      string s = "abcd";

      //s.Show();
```

```
        Class1.Show(s);
        Console.ReadLine();
    }
    static void Main()
    {
        ClsMaths o = new ClsMaths();
        Console.WriteLine(o.Multiply(10, 5));
        Console.WriteLine(o.Subtract(10, 5));

        Console.ReadLine();
    }
}
public static class Class1
{
    //create a static method in the class
    //first parameter should be the type for which you are writing the ext method
    //precede the 1st parameter with the 'this' keyword
    public static void Display(this int i)
    {
        Console.WriteLine(i);
    }
    public static int Add(this int i, int j)
    {
        return i + j;
    }
    public static void Show(this string s)
    {
        Console.WriteLine(s);
    }
    //if you define an extension method for a base class,
    // it is also available to all derived classes
    public static void ExtMethodForBaseClass(this object s)
    {
        Console.WriteLine(s);
    }
    //if you define an extension method for an interface,
    // it is also available to all classes that implement that interface
    public static int Subtract(this IMathOperations i, int a, int b)
    {
        return a - b;
    }
}

public interface IMathOperations
{
    int Add(int a, int b);
    int Multiply(int a, int b);
}

public class ClsMaths : IMathOperations
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public int Multiply(int a, int b)
    {
        return a * b;
```

Day7

Language Integrated Query:

MS .Net code will need to access data from various sources during its execution.
This data can be found in numerous(many) locations, including XML files, relational databases, in-memory collections, and primitive arrays.
Earlier programmer needed to use different and unrealted API's to access this data.
The Language Integrated Query (LINQ) technology set provides a symmetrical and strongly typed manner to access varity of data sources.

LINQ to Object,LINQ to Dataset,XML,Entities,SQL for other soruces implement IQureable i/f.

```
//var emps = from emp in lstEmp select emp;
->//emps type is IEnumerable<Employee>

//var emps = from emp in lstEmp select emp.Name;
--> //emps type is IEnumerable<string>

var emps = from emp in lstEmp select new { emp.Name, emp.EmpNo
};//IEnumerable<`a>

static List<Employee> lstEmp = new List<Employee>();
static List<Department> lstDept = new List<Department>();

AddRecs();

            var emps1 = from e in lstEmp select e;

            //var returnvalue = from single_object in collection select
something
            //IEnumerable<Employee> emps = from emp in lstEmp select emp;
            var emps = from emp in lstEmp select emp;
            foreach (var item in emps)  //item-> Employee
            {
                //Console.WriteLine(item.Name);
                //Console.WriteLine(item.EmpNo);
                Console.WriteLine(item);
            }
            Console.ReadLine();
        }


            1. var emps = from emp in lstEmp
                        where emp.Basic > 10000 && emp.Basic < 12000
                         select emp;

            2.var emps = from emp in lstEmp
                          where emp.Name.StartsWith("V")
                           select emp;
```

```
3. var emps = from emp in lstEmp
                   // where emp.Basic > 10000
                    orderby emp.Name  //defaut ascending
                    select emp;

4.var emps = from emp in lstEmp
                   orderby emp.Name descending
                     select emp;

5.var emps = from emp in lstEmp
                   orderby emp.DeptNo, emp.Name descending
                    select emp;
             // in this query only sorted by first condition i.e by DeptNo.
```

Join :

```
6. var emps = from emp in lstEmp

                      join dept in lstDept

                            on emp.DeptNo equals dept.DeptNo

                      select emp;


7. var emps1 = from emp in lstEmp

                        join dept in lstDept

                              on emp.DeptNo equals dept.DeptNo

                        select dept;


8. var emps2 = from emp in lstEmp

                        join dept in lstDept

                              on emp.DeptNo equals dept.DeptNo

                        select new { emp, dept };


9. var emps3 = from emp in lstEmp

                        join dept in lstDept

                              on emp.DeptNo equals dept.DeptNo

                        select new { emp.Name, dept.DeptName }

foreach (var item in emps)

            {

               item.dept.DeptName;

            }
```

Group by:

```
10. var emps = from emp in lstEmp
                       group emp by emp.DeptNo;


            foreach (var emp in emps) ----->  here take the one depart.
            {
                Console.WriteLine(emp.Key); //deptno

                foreach (var e in emp)
                     ----> e is an Employee, emp is a grouping of Employee
                {
                    Console.WriteLine("  " + e);
                }
             }
          }


11. var emps = from emp in lstEmp

            group emp by emp.DeptNo into group1 // here given allias for list

           select new { group1, Count = group1.Count(),
                      Max = group1.Max(x=> x.Basic), Min = group1.Min(x => x.Basic) };


            foreach (var emp in emps)

            {

                Console.WriteLine(emp.group1.Key); //DeptNo

                Console.WriteLine(emp.Count); //count

                Console.WriteLine(emp.Min); //min

                Console.WriteLine(emp.Max); //max

                //emp.group1.Key;  //DeptNo
       }


static void Main2()
        {
            AddRecs();
            //var emps = from emp in lstEmp
            //           where emp.Basic > 10000
            //           select emp;

            //foreach (var emp in emps)
            //{
            //    Console.WriteLine(emp.Name);
            //}

            var emps1 = lstEmp.Where(emp => emp.Basic > 10000);
            var emps2 = lstEmp.Where(emp => emp.Basic > 10000).Select(emp => emp);
            var emps3 = lstEmp.Select(emp => emp).Where(emp => emp.Basic > 10000);
            var emps2a = lstEmp.Where(emp => emp.Basic > 10000).Select(emp =>
emp.Name);

            //error below
            //var emps3 = lstEmp.Select(emp => emp.Name).Where(emp => emp.Basic >
10000);

            foreach (var emp in emps2)
            {
                Console.WriteLine(emp);
            }

            Console.ReadLine();

        }
        static void Main3()
        {
            AddRecs();


            //var emps = from emp in lstEmp
            //            orderby emp.Name
            //            select emp;
            ////var emps = from emp in lstEmp
            ////                                  orderby emp.Name descending
            ////                                  select emp;

            ////var emps = from emp in lstEmp
            ////            orderby emp.DeptNo, emp.Name descending
            ////            select emp;
            ///
            var emps = lstEmp.OrderBy(emp => emp.Name);
            var emps2 = lstEmp.OrderByDescending(emp => emp.Name);
            var emps3 = lstEmp.OrderBy(emp => emp.DeptNo).ThenByDescending(emp =>
emp.Name);
            foreach (var emp in emps3)
            {
                Console.WriteLine(emp);
            }



            Console.ReadLine();

        }

        static void Main4()
        {
            AddRecs();

            //var emps = from emp in lstEmp
            //            join dept in lstDept
            //                  on emp.DeptNo equals dept.DeptNo
            //            select emp;
            //var emps1 = from emp in lstEmp
            //             join dept in lstDept
            //                   on emp.DeptNo equals dept.DeptNo
            //             select dept;
            //var emps2 = from emp in lstEmp
            //             join dept in lstDept
            //                   on emp.DeptNo equals dept.DeptNo
            //             select new { emp, dept };
            //var emps3 = from emp in lstEmp
            //            join dept in lstDept
            //                  on emp.DeptNo equals dept.DeptNo
            //            select new { emp.Name, dept.DeptName };

            //foreach (var emp in emps3)
            //{
            //    Console.WriteLine(emp);
            //}



            var emps = lstEmp.Join(lstDept, emp => emp.DeptNo, dept => dept.DeptNo,
(emp, dept) => emp);

            //var emps2a = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => emp);
            //var emps2b = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => dept);
            //var emps2c = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => emp.DeptNo);
            //var emps2d = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => dept.DeptNo);
            //var emps2e = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => new { emp, dept });
            //var emps2f = lstEmp.Join(lstDept, emp => emp.DeptNo, dept =>
dept.DeptNo, (emp, dept) => new { emp.Name, dept.DeptName });

            Console.ReadLine();
        }

        static void Main5()
        {
            AddRecs();
            //deferred execution
            var emps = from emp in lstEmp select emp;

            Console.WriteLine();
            lstEmp.RemoveAt(0);

            foreach (var emp in emps)
            {
                Console.WriteLine(emp.Name + "," + emp.EmpNo);
            }
            Console.ReadLine();


            Console.WriteLine();
            lstEmp.Add(new Employee { EmpNo = 9, Name = "New", Basic = 11000, DeptNo =
40, Gender = "F" });
            foreach (var emp in emps)
            {
                Console.WriteLine(emp.Name + "," + emp.EmpNo);
            }
            Console.ReadLine();
        }
        static void Main6()
        {
            AddRecs();
            var emps = from emp in lstEmp select emp;
            //immediate execution
            emps = emps.ToList();  //.ToArray .ToDictionary
            //Employee [] arrEmps = emps.ToArray();
            //Dictionary<int, Employee> d = emps.ToDictionary(e => e.EmpNo);


            Console.WriteLine();
            lstEmp.RemoveAt(0);
            foreach (var emp in emps)
            {
                Console.WriteLine(emp.Name + "," + emp.EmpNo);
            }
            Console.ReadLine();
            Console.WriteLine();
            lstEmp.Add(new Employee { EmpNo = 9, Name = "New", Basic = 11000, DeptNo =
40, Gender = "F" });
            foreach (var emp in emps)
            {
                Console.WriteLine(emp.Name + "," + emp.EmpNo);
            }
            Console.ReadLine();
        }
        static void Main7()
        {
            AddRecs();
            //Employee emp = lstEmp.Single(e => e.EmpNo == 1); //one record = okay
            //Employee emp = lstEmp.Single(e => e.EmpNo == 10);  //no records = error
            //Employee emp = lstEmp.Single(e => e.Basic > 5000); //multiple records -
error


            Employee emp = lstEmp.SingleOrDefault(e => e.EmpNo == 1); //one record =
okay
            //Employee emp = lstEmp.SingleOrDefault(e => e.EmpNo == 10); //no
records=null
            //Employee emp = lstEmp.SingleOrDefault(e =>  e.Basic > 5000);//multiple
records - error

            //if (emp != null)
            //    Console.WriteLine(emp.Name + "," + emp.EmpNo);
            //else
            //    Console.WriteLine("not found");
            Console.ReadLine();
        }



        static void Main()
        {

            Thread t = new Thread();
            //AddRecs();
            AddRecs2();
            //plinq example

            Stopwatch stopwatch = new Stopwatch();

            stopwatch.Start();
            //var emps = lstEmp.Select(emp => new { Name = LongRunningFunc(emp.Name),
emp.EmpNo });
            var emps = lstEmp.AsParallel().WithDegreeOfParallelism(2).Select(emp =>
new { Name = LongRunningFunc(emp.Name), emp.EmpNo });
```

```
            //Console.WriteLine("Elapsed Time is {0} ms", 
stopwatch.ElapsedMilliseconds);
            foreach (var emp in emps)
            {
                Console.WriteLine(emp.Name + "," + emp.EmpNo);
            }
            stopwatch.Stop();
            Console.WriteLine("Elapsed Time is {0} ms", 
stopwatch.ElapsedMilliseconds);

            Console.ReadLine();
        }

        public static void ThreadTesting()
        {
            Console.WriteLine("Thread");
        }
        public static void AddRecs2()
        {
            //Stopwatch stopwatch = new Stopwatch();

            //stopwatch.Start();
            for (int i = 0; i < 200; i++)
            {
                lstEmp.Add(new Employee { EmpNo = i, Name = "Vikram"+i, Basic = 10000, 
DeptNo = 10, Gender = "M" });
            }
            //stopwatch.Stop();
            //Console.WriteLine("Elapsed Time is {0} ms", 
stopwatch.ElapsedMilliseconds);
        }
        public static string LongRunningFunc(string s)
        {
            System.Threading.Thread.Sleep(10);
            return s.ToUpper();
        }
    }
}

//https://linqsamples.com/

//https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-
to-plinq
```

Day8

Threading:

Thread t = new Thread(new ThreadStart(Display));

ThreadStart is delegate for function with void return type and args

//below syntax calls the above code internally

Thread t = new Thread(Display);

t.Start(); //function is called here on a separate thread

Threads runs in foreground by default and main waits still all threads are terminated.(similar to t.join() at the end of main)

We can make thread background : t1.IsBackground = true;

- Foreground Thread runs at front application wait to complete the excuation of foreground thread.

- background Thread runs at back end of application and appln NOT wait to complete the excuation of background threads.

If we want main to wait before executing till some thread completes

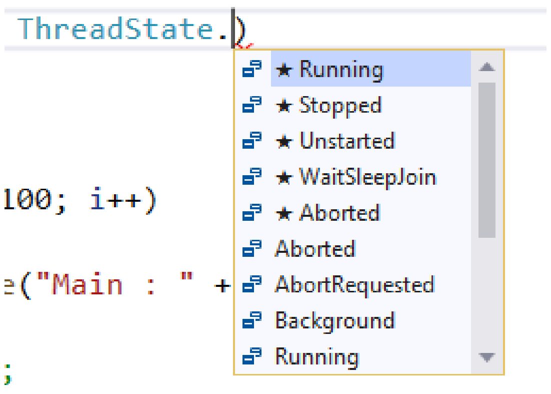
We can use t1.join();

We can assign diff priorities to threads to request more time for particular thread however this is always a request and there is no guarantee that the thread will actaully get more time.

t1.Priority = ThreadPriority.Highest;

Abort: to terminate thread, (not immediate termination)

Thread states:



```
ThreadState.)
100; i++)
e("Main : " +
;
```

```
void Display(){…}
```

```
  public static void Main(){
        Thread t1 = new Thread(Func1);
        Thread t2 = new Thread(Func2);

        //t1.IsBackground = true;
        //t1.Priority = ThreadPriority.Highest;
        //if(t1.ThreadState == ThreadState.)

        t1.Start();
        t2.Start();
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine("Main : " + i);
            //if(i==50)
            //    t1.Abort();

        }
        //t1.Suspend();  //obsolete - has no effect on code
        //t1.Resume();  //obsolete- has no effect on code

        //instead of suspend, use this
        //WaitHandle wh = new AutoResetEvent(false);
        //wh.WaitOne();


        //t1.Resume(); //deprecated
        // instead of resume, use this
        //((AutoResetEvent)wh).Set();


        //t1.Join(); //waiting call for t1 to join back again
        //Console.WriteLine("this code should run only after func1");
        Console.ReadLine();
        ///main thread waiting for foreground thread to complete
    }

    static void Func1()
    {
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine("First : " + i);
        }

    }
    static void Func2()
    {
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine("Second : " + i);
        }

    }
}
```

```
}
```

Passing parameters to threads:

Single parameter:

```
      Thread t1 = new Thread(new ParameterizedThreadStart(Func1));
            //Thread t2 = new Thread(new ParameterizedThreadStart(Func2));
            Thread t2 = new Thread(Func2);

            t1.Start("o1"); // here passing the parameter
            t2.Start("o2");
public void Funct1(object o){…}
```

Multiple Parameters:

1 way can be pass Collection or Object

Anonymous method in same class as it can access local variables;

Thread Pool: accessing available resources and return it after completing the task

```
//ThreadPool.GetMinThreads (by default available threads ) //1000

class MainClass
    {
        static void PoolFunc1(object o)
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("First Thread " + i.ToString() +
o.ToString());
            }
        }
        static void PoolFunc2(object o)
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("Second Thread " + i.ToString());
            }
        }
        static void Main()
        {
            //ThreadPool.QueueUserWorkItem(new WaitCallback(PoolFunc1),
"aaa");
```

```
            ThreadPool.QueueUserWorkItem(PoolFunc1, "aaa");
            ThreadPool.QueueUserWorkItem(new WaitCallback(PoolFunc2));

            QueueUserWorkItem:  Queues a method for execution. The method
        executes when a thread pool thread becomes available.

            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("Main Thread " + i.ToString());
            }
            int workerthreads, iothreads;

            ThreadPool.GetAvailableThreads(out workerthreads, out
iothreads);
            //ThreadPool.SetMinThreads
            //ThreadPool.SetMaxThreads
            //ThreadPool.GetMinThreads  //1000
            Console.WriteLine(workerthreads);
            Console.WriteLine(iothreads);

            Console.ReadLine();
        }
    }
}
```

Synchronisation:

`Lock(obj)  // Monitor.Enter(lockObject);`

`Interlocked.Add(ref i, 10); //have multiple methods to avoid writing lock every time`

Interlocked functions do not lock. They are atomic, meaning that they can complete without the possibility of a context switch during increment. So there is no chance of deadlock or wait.

```
static object lockObject = new object();
        static int i = 0;
        static void Main1()
        {
            Thread t1 = new Thread(new ThreadStart(FuncLock));
            Thread t2 = new Thread(new ThreadStart(FuncMonitor));
            Thread t3 = new Thread(new ThreadStart(FuncInterlocked));
            t1.Start();
            t2.Start();
            t3.Start();

         }

static void FuncLock()
       {
           lock (lockObject) //Monitor.Enter(lockObject)

             …..
        }
static void FuncMonitor()
        {
            Monitor.Enter(lockObject);

              .…

            Monitor.Exit(lockObject);


        }
static void FuncInterlocked()
        {

            Interlocked.Add(ref i, 10);   //i+=10

              …….
        }
```

Task::

Repersents asynch operations. Task are use to handle threading. Tasks internally use threading

Following the different way to create Task:

1. **Task t1** = **new Task(Func1)**; -> Intializes new task with specified action
   t1.Start(); ----> starts Task

2. **Task t1** = **Task.Run(Func1)**;//Exception NullArgsExcp

   Run- Queues the specified work to run on the thread pool and returns a System.Threading.Tasks.Task object that represents that work.
   By using Run method we create and start the task at one line.

3. **Task t2** = **Task.Factory.StartNew(Func2)**; ->StartNew creates and starts a task

Use Task.Run when no args are passed and Task.Factory.StartNew with 1 arg

```
//calling a method with void return type using taskobj.Start
namespace Example1
{
    class Program
    {
        static void Main1()
        {

            Task t1 = new Task(Func1);

            //Action objAction1 = Func1;
            //Task t1 = new Task(objAction1);


            //Task t3 = new Task(new Action(Func1));

            Action objAction2 = Func2;
            Task t2 = new Task(objAction2);

            t1.Start();
            t2.Start();

            Console.ReadLine();
        }
        static void Func1()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine("first Func called from {0},{1}",
Thread.CurrentThread.ManagedThreadId, i);
            }
        }
        static void Func2()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine("second Func called from {0},{1}",
Thread.CurrentThread.ManagedThreadId, i);
            }
        }
    }
}



//calling a method with void return type using Task.Run and Task.Factory.StartNew
namespace Example2
{
    class Program
    {
        static void Main2()
        {
            //Action objAction1 = Func1;
            //Task t1 = Task.Run(objAction1);
            Task t1 = Task.Run(Func1);


            //Action objAction2 = Func2;
            //Task t2 = Task.Factory.StartNew(objAction2);
            Task t2 = Task.Factory.StartNew(Func2);

       --Both Run and StartNew takes Action as param

            Console.ReadLine();
        }
        static void Func1()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine("first Func called from {0},{1}",
Thread.CurrentThread.ManagedThreadId, i);
            }
        }
        static void Func2()
        {
            for (int i = 0; i < 100; i++)
            {
                Console.WriteLine("second Func called from {0},{1}",
Thread.CurrentThread.ManagedThreadId, i);
```

```
            }
        }
    }
}

//calling a method with void return type and parameters
namespace Example3
{
    class Program
    {
        static void Main3()
        {

            //Action<object> objAction1 = Func1;
            //Task t1a = new Task(objAction1, "aaa");
            Task t1 = new Task(Func1, "aaa");
            //Task.Run - cannot be used with parameters.
             //use anonymous methods instead to access variables declared in
calling code

            string s = "aaa";
            Task.Run(delegate () { Console.WriteLine(s); });
            Task.Run( ()=> { Console.WriteLine(s); });

            //Action<object> objAction2 = Func2;
            //Task t2 = Task.Factory.StartNew(objAction2, "bbb");
            Task t2 = Task.Factory.StartNew(Func2, "bbb");

            t1.Start();
            Console.ReadLine();
        }
        static void Func1(object obj)
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("first Func called {0}{1}", i, obj.ToString());
            }
        }
        static void Func2(object obj)
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("second Func called {0},{1}", i, obj.ToString());
            }
        }
    }
}
```

Blocking and Non Blocking calls:

If we want to wait till t1.Result get result and we wrote synchronous code then the thread gets bloack and no other task can be performed(UI example).

If we want to call this with seprate thread to avoid bloacking such call is called non-bloacking call.

We use **await()** for this requirement;

The function must return a awaitable Task for await fucntion call,

Any function returning Task(awaitable) can be called with await

Await can be only called in async method(Main()) -> and in this case Main() cannot return void but has to return Task

The function that can be called with await keyword must have async

```
static async Task Main()
        {
            Console.WriteLine("Before");
            //string message = DoWorkAsync();  //blocking call
            string message = await DoWorkAsync();  //waiting call
            string newmsg = await NewFunc();
            Console.WriteLine(message);
            Console.WriteLine("After");
            Console.ReadLine();
        }
        static async Task<string> DoWorkAsync()  //awaitable,must for
await
        {
            return await Task.Run(() =>
            {
                Thread.Sleep(5000);
                return "Done with work!";
            });
        }
```

```
//this also works, just simple Task returning string.
      static Task<string> NewFunc()
       {
           Task<string> t = Task.Run<string>(func1);
           return t;
       }

       static string func1()
       {
           return "this is cool";
       }
```

```
//Another option,generally not used
        static async Task Main2()
        {
            Console.WriteLine("Before");

            Task<Task<string>> t1 = new Task<Task<string>>(DoWorkAsync);
            t1.Start();
            Console.WriteLine("After");

            string message = await t1.Result;
            Console.WriteLine(message);
            Console.ReadLine();
        }
```

Operator Overloading:

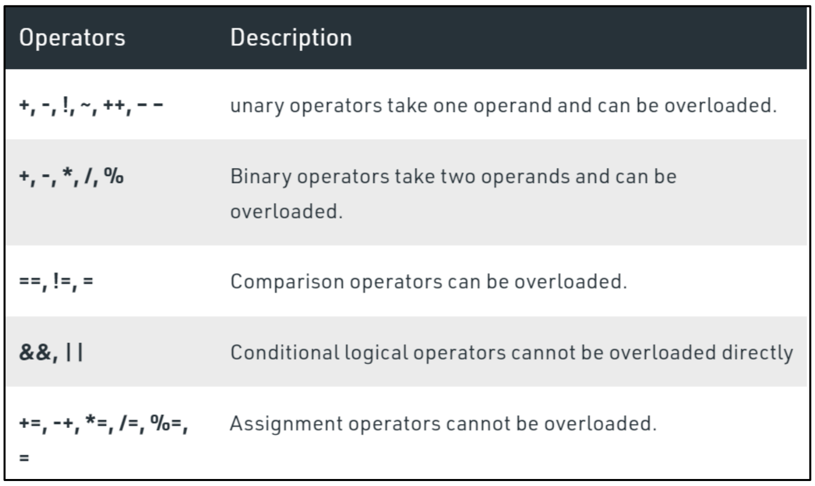
-The concept of overloading a function can also be applied to *operators*.
-Operator overloading gives the ability to use the same operator to do various operations.
-It provides additional capabilities to *C#* operators when they are applied to user-defined data types

Syntax:

```
Access_specifier static className  operator Operator_symbol
(parameters)
{
    // Code
}
```

```
class Program
    {
        static void Main(string[] args)
        {
            Class1 o = new Class1 { i = 10 };
            Class1 o2 = new Class1 { i = 20 };
            o = o + 5;  ----> this calls the overloading method internally

             o += 5;  ---->//o = Class1.operator+(o,5)

            o = o + o2;----->//o = Class1.operator+(o,o2);

            Console.WriteLine(o.i);
            Console.ReadLine();
        }
    }
    public class Class1
    {
        public int i;
        //public int j ;
        public static Class1 operator +(Class1 o, int i)
        {
            Class1 retval = new Class1();
            retval.i = o.i + i;
            return retval;
        }
        public static Class1 operator +(Class1 o, Class1 o2)
        {
            Class1 retval = new Class1();
            retval.i = o.i + o2.i;
            return retval;
        }
        }
}
```

```
https://docs.microsoft.com/en-us/dotnet/csharp/language-
reference/operators/operator-overloading


Operators that can be and connot be overloaded:
```



Operators	Description
+, -, !, ~, ++, - -	unary operators take one operand and can be overloaded.
+, -, *, /, %	Binary operators take two operands and can be overloaded.
==, !=, =	Comparison operators can be overloaded.
&&, ||	Conditional logical operators cannot be overloaded directly
+=, -+, *=, /=, %=, =	Assignment operators cannot be overloaded.

Indexer:

Simple word: indexer used for store the miltiple values in field member for single object of the class(generally required object per value) and also start the indexing from wherever required.

An indexer allows an instance of a class or struct to be indexed as an array. If the user will define an indexer for a class, then the class will behave like a virtual array. Array access operator i.e ([ ]) is used to access the instance of the class which uses an indexer. A user can retrieve or set the indexed value without pointing an instance or a type member. Indexers are almost similar to the Properties. The main difference between Indexers and Properties is that the accessors of the Indexers will take parameters.

Important Points About Indexers:

1. There are two types of Indexers i.e. **One Dimensional Indexer** & *MultiDimensional Indexer*. The above discussed is One Dimensional Indexer.
2. Indexers can be overloaded.
3. These are different from *Properties*.
4. This enables the object to be indexed in a similar way to arrays.
5. A set accessor will always assign the value while the get accessor will return the value.
6. "*this*" keyword is always used to declare an indexer.
7. To define the value being assigned by the set indexer, " *value*" keyword is used.

8. Indexers are also known as the *Smart Arrays* or *Parameterized Property* in C#.
9. Indexer can't be a static member as it is an instance member of the class.

Syntax:

```
[access_modifier] [return_type] this [argument_list]
{
  get
  {
     // return the value for given index
  }
  set
  {
    // set the value for given index
  }

}
```

```
// C# program to illustrate the Indexer
using System;

// class declaration
class IndexerCreation
{

        // class members
        private string[] val = new string[3];

        // Indexer declaration
        // public - access modifier
        // string - the return type of the Indexer
        // this - is the keyword having a parameters list
        public string this[int index]
        {
                // get Accessor
                // retrieving the values
                // stored in val[] array
                // of strings
                get
                {

                        return val[index];
                }

                // set Accessor
                // setting the value at
                // passed index of val
                set
                {

                        // value keyword is used
                        // to define the value
                        // being assigned by the
                        // set indexer.
                        val[index] = value;
                }
        }
}

// Driver Class
class main {

        // Main Method
        public static void Main() {
                // creating an object of parent class which
                // acts as primary address for using Indexer
                IndexerCreation ic = new IndexerCreation();

                // Inserting values in ic[]
                // Here we are using the object
                // of class as an array
                ic[0] = "C";
                ic[1] = "CPP";
                ic[2] = "CSHARP";

                Console.Write("Printing values stored in objects used as arrays\n");

                // printing values
                Console.WriteLine("First value = {0}", ic[0]);
                Console.WriteLine("Second value = {0}", ic[1]);
                Console.WriteLine("Third value = {0}", ic[2]);

        }
}

//indexer example
class Program
    {
        static void Main1()
        {
                Class1 o = new Class1();
                o[0] = 10;
                Console.WriteLine(o[0]);
                o[1] = 20;
                Console.WriteLine(o[1]);
                o[2] = 30;
                Console.WriteLine(o[2]);

                Console.ReadLine();
        }
    }
    class Class1
    {
        private int[] arr = new int[3];
        public int this[int i]
        {
            get
            {
                return arr[i];
            }
            set
            {
                arr[i] = value;
            }
        }
        //public int Length
        //{
        //    get
        //    {
        //        return arr.Length;
        //    }
        //}
    }
}


namespace Indexer2
{
    class Program
    {
        static void Main()
        {
            Class1 o = new Class1(5,2000);

            //o[0]=100;
            o[2000] = 100;
            o[2001] = 100;
            o[2002] = 100;
            o[2003] = 100;
            o[2004] = 100;
            Console.WriteLine(o[2003]);
            Console.ReadLine();
        }
    }
    public class Class1
    {
        private int[] arr;
        int start;
        public Class1(int Size, int start)
        {
            this.start = start;
            arr = new int[Size];
        }
        public int this[int index]
        {
            get
            {
                return arr[index - start];
            }
            set
            {
                arr[index - start] = value;
            }
        }

    }
}
```

Day9

Files:

Files are examples of streams. **A stream is basically a sequence of data(bytes).** Whatever data we use in our programming flows through a stream.

System.IO

```
DriveInfo drive = new DriveInfo("C");
Info about drive can be access using DriveInfo
```

- drive.IsReady, AvailableFreeSpace,name,total size,drive format

```
Directory has all many static methods
//Directory.CreateDirectory("C:\\aaa");

Directory.–Delete(path,recursive)
//DirectoryInfo dir = new DirectoryInfo("C:\\aaaa");

File.Create("c:\\aaa\\a.txt");
FileInfo
```

Stream class(abstract) and its derived classes:

```
public abstract class Stream : MarshalByRefObject, IAsyncDisposable,
Idisposable
```

Derived:

Microsoft.JScript.COMCharStream

System.Data.OracleClient.OracleBFile

System.Data.OracleClient.OracleLob

System.Data.SqlTypes.SqlFileStream

System.IO.BufferedStream

System.IO.Compression.BrotliStream

System.IO.Compression.DeflateStream

System.IO.Compression.GZipStream

System.IO.Compression.ZLibStream

System.IO.FileStream

System.IO.MemoryStream

System.IO.Pipes.PipeStream

System.IO.UnmanagedMemoryStream

System.Net.Quic.QuicStream

System.Net.Security.AuthenticatedStream

System.Net.Sockets.NetworkStream

System.Printing.PrintQueueStream

System.Security.Cryptography.CryptoStream

Path: C :\\aaaa\\a.txt" :\ as escape character

To avoid we can use (@"C:\aaaa\a.txt")

```
FileMode: Create – overrides if file exist
          CreateNew – if not present creates,and thorws expc if exist

          Truncate- clears the contents of existing file and then opens
```

Unformatted IO

```
        string s = "Hello World";
         byte[] arr = Encoding.Default.GetBytes(s);
         FileStream stream = File.Open("C:\\aaaa\\a.txt",
FileMode.Create);

         stream.Write(arr, 0, arr.Length);

         //stream.Length
         ////stream.Read()

         stream.Close();
         //-------------------------------
```

Formatted IO

```
   For writing Text in File StreamWriter implements TextWriter

         StreamWriter writer = File.CreateText("C:\\aaaa\\a.txt");

         writer.WriteLine("Hello World");
         writer.WriteLine("Line 2");
            writer.Close();


namespace WindowsFormsApplication1
{
 public partial class Form1 : Form
 {
     public Form1()
     {
         InitializeComponent();
     }

     private void button1_Click(object sender, EventArgs e)
     {

         //Directory.CreateDirectory("C:\\aaa");
         Directory.CreateDirectory(@"C:\aaaa");


         //DirectoryInfo dir = new DirectoryInfo("C:\\aaaa");

         //dir.

     }

     private void button2_Click(object sender, EventArgs e)
     {

         File.Create("C:\\aaaa\\a.txt");

         //FileInfo file = new FileInfo("C:\\aaaa\\a.txt");
         //file.
     }

     private void button3_Click(object sender, EventArgs e)
     {

         DriveInfo drive = new DriveInfo("C");
         Console.WriteLine(drive.);

     }

     private void button4_Click(object sender, EventArgs e)
     {
         string s = "Hello World";
         byte[] arr = Encoding.Default.GetBytes(s);
         FileStream stream = File.Open("C:\\aaaa\\a.txt",
FileMode.OpenOrCreate);

         stream.Write(arr, 0, arr.Length);

         //stream.Length
         //stream.Read()

         stream.Close();
         //-------------------------------
         StreamWriter writer = File.CreateText("C:\\aaaa\\a.txt");

         writer.WriteLine("Hello World");
         writer.WriteLine("Line 2");
         writer.Close();
     }

     private void button5_Click(object sender, EventArgs e)
     {
         string s;
         StreamReader reader = File.OpenText("C:\\aaaa\\a.txt");
         while ((s = reader.ReadLine())!= null)
         {
             MessageBox.Show(s);
         }
         reader.Close();
     }

     private void button6_Click(object sender, EventArgs e)
     {
         string s = "Hello";
         int i = 100;
         bool b = true;

         FileInfo f = new FileInfo("C:\\aaaa\\a.dat");

         BinaryWriter binary_writer = new BinaryWriter(f.OpenWrite());
         binary_writer.Write(s);
         binary_writer.Write(i);
         binary_writer.Write(b);

         binary_writer.Close();
     }

     private void button7_Click(object sender, EventArgs e)
     {
         string s ;
         int i ;
         bool b ;

         FileInfo f = new FileInfo("C:\\aaaa\\a.dat");

         BinaryReader binary_reader = new BinaryReader(f.OpenRead());

         s = binary_reader.ReadString();
         i = binary_reader.ReadInt32();
         b = binary_reader.ReadBoolean();

         MessageBox.Show(s);
         MessageBox.Show(i.ToString());
         MessageBox.Show(b.ToString());
         binary_reader.Close();
     }

     private void Form1_Load(object sender, EventArgs e)
     {
        }
    }
```

Serialization: Convert any data into stream of bytes is serilization

CW(“Hello”); -> this convert string into stream of bytes

Object serialization:

[Serializable] -Notation to indiacte that class can be sterilized

```
   [Serializable]
    public class Class1 // :Iserializable –for custom serilization
```

[Serializable]- class SerializableAttribute(sealed) which extends Attribute(abstarct) class

```
//
    // Summary:
    //     Indicates that a class can be serialized. This class cannot be
inherited.
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct |
AttributeTargets.Enum | AttributeTargets.Delegate, Inherited = false)]
    [ComVisible(true)]
    public sealed class SerializableAttribute : Attribute
    {
        //
        // Summary:
        //     Initializes a new instance of the
System.SerializableAttribute class.
            public SerializableAttribute();
```

Example:

```
    [Serializable]
    public class Class1 // :Iserializable-for custom serilization
    {
        private int private_data;
        public int i;
        [XmlElement]
        public string P1
        {
            get;
            set;
        }
        private int mP2;
        [XmlAttribute]
        public int P2
        {
            get { return mP2; }
            set { mP2 = value; }
        }


        //public void GetObjectData(SerializationInfo info,
StreamingContext context)
        //{
        //    info.AddValue("i", i);

        //}
       }





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

        private void button1_Click(object sender, EventArgs e)
        {
            Class1 o = new Class1();
            o.i = 100;
            o.P1 = "aaa";
            o.P2 = 200;
            BinaryFormatter bf = new BinaryFormatter();
            Stream s = new FileStream("C:\\o.dat", FileMode.Create);
            bf.Serialize(s, o);
            s.Close();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            BinaryFormatter bf = new BinaryFormatter();
            Stream s = new FileStream("C:\\o.dat", FileMode.Open);
            Class1 o= (Class1)bf.Deserialize(s);
            s.Close();
            MessageBox.Show(o.i.ToString());
            MessageBox.Show(o.P1);
            MessageBox.Show(o.P2.ToString());
        }

        private void button4_Click(object sender, EventArgs e)
        {
            Class1 o = new Class1();
            o.i = 100;
            o.P1 = "aaa";
            o.P2 = 200;
            SoapFormatter sf= new SoapFormatter();
            Stream s = new FileStream("C:\\o.soap", FileMode.Create);
            sf.Serialize(s, o);
            s.Close();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            SoapFormatter sf = new SoapFormatter();
            Stream s = new FileStream("C:\\o.soap", FileMode.Open);

            Class1 o = (Class1)sf.Deserialize(s);
            s.Close();
            MessageBox.Show(o.i.ToString());
            MessageBox.Show(o.P1);
            MessageBox.Show(o.P2.ToString());
        }

        private void button6_Click(object sender, EventArgs e)
        {
            Class1 o = new Class1();
            o.i = 100;
            o.P1 = "aaa";
            o.P2 = 200;
            XmlSerializer xs = new XmlSerializer(typeof(Class1) );
            Stream s = new FileStream("C:\\o.xml", FileMode.Create);
            xs.Serialize(s, o);
            s.Close();
        }
        private void button5_Click(object sender, EventArgs e)
        {
            XmlSerializer xs = new XmlSerializer(typeof(Class1));
            Stream s = new FileStream("C:\\o.xml", FileMode.Open);
            Class1 o = (Class1)xs.Deserialize(s);
            s.Close();
            MessageBox.Show(o.i.ToString());
            MessageBox.Show(o.P1);
            MessageBox.Show(o.P2.ToString());

        }
        private void button7_Click(object sender, EventArgs e)
        {
            Class1 o = new Class1();
            o.i = 100;
            o.P1 = "aaa";
            o.P2 = 200;
            DataContractJsonSerializer js = new
DataContractJsonSerializer(typeof(Class1));
            Stream s = new FileStream("C:\\o.json", FileMode.Create);
            js.WriteObject(s, o);
            s.Close();
        }
        private void button8_Click(object sender, EventArgs e)
        {
            DataContractJsonSerializer js = new
DataContractJsonSerializer(typeof(Class1));
            Stream s = new FileStream("C:\\o.json", FileMode.Open);
            Class1 o = (Class1)js.ReadObject(s);
            s.Close();
            MessageBox.Show(o.i.ToString());
            MessageBox.Show(o.P1);
            MessageBox.Show(o.P2.ToString());
        }
```

XmlSerializer serilize public members only, other serilizers serilize only variables/fileds and not properties.

Reflection: To read contents of assembly, we can create obj of that assy at run time and also we can create assy at runtime using reflection.

Inspection

System.Reflection: namespace

```
static void Main()

        {

            Assembly asm =
Assembly.LoadFrom(@"D:\Trainings\ActsAug22\Day2\InheritanceExamples\bin\De
bug\InheritanceExamples.exe");

            //Assembly.GetAssembly(typeof(string));

            //Console.WriteLine(asm.FullName);

            Console.WriteLine(asm.GetName().Name);

            Type[] arrTypes = asm.GetTypes();

            foreach (Type t in arrTypes)

            {

                Console.WriteLine("    "+ t.Name);

                MethodInfo[] arrMethods = t.GetMethods();

                //for...

            }

            Console.ReadLine();

        }
```

SQL server

SqlConnection--→SqlCommand--------------→ SqlDataReader

```
SqlConnection cn = new SqlConnection();

Cn.Open();
SqlCommand cmdInsert = new SqlCommand();
cmdInsert.Connection = cn;
cmdInsert.CommandType = System.Data.CommandType.Text;
cmdInsert.CommandText = "insert into Employees values(4,'Aryan',12345,20)";
cmdInsert.ExecuteNonQuery()
```

```
static void Main(string[] args)
        {
            Employee emp = new Employee { EmpNo=5,Name="Nirav",Basic=1200,DeptNo=30};
            // InsertEmployeeUsingStoredProc(emp);
            UpdateEmployee(emp);

            Console.ReadLine();
        }


        static void Connect()
        {
            SqlConnection cn = new SqlConnection();
            try
            {

                //Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mult
iSubnetFailover=False
                cn.ConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;Integrated Security=True";

//get ConnectionString from sql server properties by (right clicking)
                //cn.ConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;User id=user1;Password=pass**";
                cn.Open();


                SqlCommand cmdInsert = new SqlCommand();
                cmdInsert.Connection = cn;
                cmdInsert.CommandType = System.Data.CommandType.Text;
                cmdInsert.CommandText = "insert into Employees
values(4,'Aryan',12345,20)";
                cmdInsert.ExecuteNonQuery();//returns no.of rows affected
                Console.WriteLine("success");

            }
            catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                cn.Close();
            }
        }

        static void InsertEmployeeUsingStoredProc(Employee emp)
        {
            SqlConnection cn = new SqlConnection();
            try
            {

                //Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mult
iSubnetFailover=False
                cn.ConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;Integrated Security=True";
                //cn.ConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;User id=user1;Password=pass**";
                cn.Open();


                SqlCommand cmdInsert = new SqlCommand();
                cmdInsert.Connection = cn;
                cmdInsert.CommandType = System.Data.CommandType.StoredProcedure;
                cmdInsert.CommandText = "InsertEmployee";
                cmdInsert.Parameters.AddWithValue("@EmpNo",emp.EmpNo);
                cmdInsert.Parameters.AddWithValue("@Name", emp.Name);
                cmdInsert.Parameters.AddWithValue("@Basic", emp.Basic);
                cmdInsert.Parameters.AddWithValue("@DeptNo", emp.DeptNo);
                cmdInsert.ExecuteNonQuery();
                Console.WriteLine("success");

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                cn.Close();
            }
        }
        public static void UpdateEmployee(Employee emp)
        {

            SqlConnection cn = new SqlConnection();
            try
            {
                cn.ConnectionString = @"Data Source=(localdb)\ProjectsV13;Initial
Catalog=ActsJuly22;Integrated Security=True";
                cn.Open();
                SqlCommand cmdUpdate = new SqlCommand();
                cmdUpdate.Connection = cn;
                cmdUpdate.CommandType = System.Data.CommandType.StoredProcedure;
                cmdUpdate.CommandText = "UpdateEmployee";
                cmdUpdate.Parameters.AddWithValue("@EmpNo",emp.EmpNo);
                cmdUpdate.Parameters.AddWithValue("@Name", emp.Name);
                cmdUpdate.Parameters.AddWithValue("@Basic", emp.Basic);
                cmdUpdate.Parameters.AddWithValue("@DeptNo", emp.DeptNo);
                cmdUpdate.ExecuteNonQuery();
                Console.WriteLine("Employee record updated ");

            }
            catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                cn.Close();

            }
        }

    }


    public class Employee
    {
        public int EmpNo { get; set; }
        public string Name { get; set; }
        public decimal Basic { get; set; }
        public int DeptNo { get; set; }

    }
}
```

Day10

Single value Query: `ExecuteScalar() -returns 1st col of 1st row`

Multiple values: SqlDataReader- firehouse cursor- quickest way to read data, **readonly,forward only,connected**

SqlDataReader dr=cmd.ExecuteReader();

```
dr.Read()----> true/false

while(dr.Read())

    Console.WriteLine(dr["EmoNo"]); --EmpNo col name
```

// dr[0] ;// 0 : column no.

// dr.GetInt32(0);   - returns data type int

MARS: Used to read data from multiple tables using nested connections,

make MultipleActiveResultSets=true" in connection string

code:

```
SqlCommand cmdDepts = new SqlCommand();
            cmdDepts.Connection = cn;
            cmdDepts.CommandType = CommandType.Text;
            cmdDepts.CommandText = "Select * from Departments";

            SqlCommand cmdEmps = new SqlCommand();
            cmdEmps.Connection = cn;
            cmdEmps.CommandType = CommandType.Text;

            SqlDataReader drDepts = cmdDepts.ExecuteReader();
            //Error if Mars not true - InvalidOperation Exception
            while (drDepts.Read())
            {
                Console.WriteLine((drDepts["DeptName"]));

                cmdEmps.CommandText = "Select * from Employees where DeptNo = " +
drDepts["DeptNo"];
                SqlDataReader drEmps = cmdEmps.ExecuteReader();
                while (drEmps.Read())
                {
                    Console.WriteLine(("    " + drEmps["Name"]));
                }
                drEmps.Close();
            }
            drDepts.Close();
            cn.Close();
```

When a function is returning a data reader :

```
static void CallFuncReturningSqlDataReader()
        {
            SqlDataReader dr = GetDataReader();
            while (dr.Read())
            {
                Console.WriteLine(dr[1]);
            }
            dr.Close();
            Console.WriteLine();
            //Console.WriteLine(cn.State);
            Console.ReadLine();
        }
        //static SqlConnection cn = new SqlConnection();

        static SqlDataReader GetDataReader()
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=ActsJuly22;Integrated Security=True";
```

```
            cn.Open();
            SqlCommand cmdInsert = new SqlCommand();
            cmdInsert.Connection = cn;
            cmdInsert.CommandType = System.Data.CommandType.Text;
            cmdInsert.CommandText = "select * from Employees ";
            //SqlDataReader dr = cmdInsert.ExecuteReader();
            SqlDataReader dr =
cmdInsert.ExecuteReader(CommandBehavior.CloseConnection);--auto close cn when dr is
close
            //cn.Close();
            return dr;
        }
    }
```

Transaction: Ado level trnsn

```
static void Transactions()
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=ActsJuly22;Integrated Security=True";
            cn.Open();
            SqlTransaction t = cn.BeginTransaction();

            SqlCommand cmdInsert = new SqlCommand();
            cmdInsert.Connection = cn;
            cmdInsert.Transaction = t;


            cmdInsert.CommandType = System.Data.CommandType.Text;
            cmdInsert.CommandText = "insert into Employees values(10, 'Shweta', 12345,
30)";


            SqlCommand cmdInsert2 = new SqlCommand();
            cmdInsert2.Connection = cn;
            cmdInsert2.Transaction = t;

            cmdInsert2.CommandType = System.Data.CommandType.Text;
            cmdInsert2.CommandText = "insert into Employees values(1, 'Shweta', 12345,
30)";
            try
            {
                cmdInsert.ExecuteNonQuery();
                cmdInsert2.ExecuteNonQuery();
                t.Commit();
                Console.WriteLine("no errors- commit");
            }
            catch (Exception ex)
            {
                t.Rollback();
                Console.WriteLine("Rollback");
                Console.WriteLine(ex.Message);
            }
            cn.Close();

        }
```

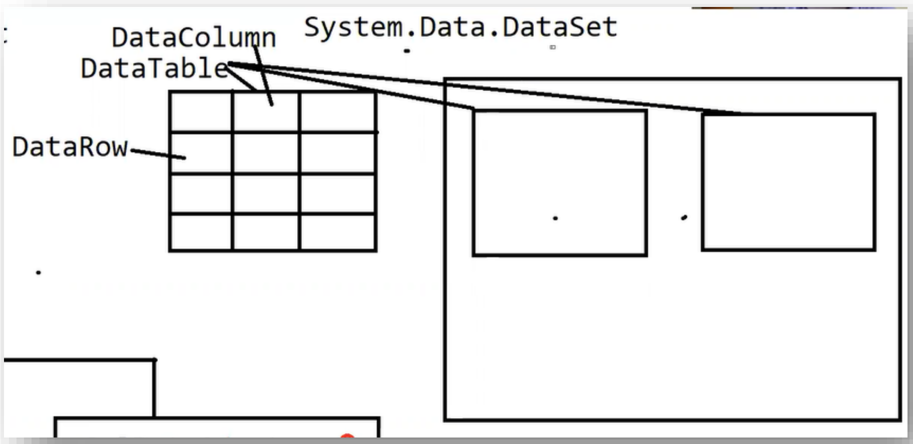
DataSet/DataTable: Disconnected, Updatable,xml set of records – here updatable makes changes in memory only and not in db,

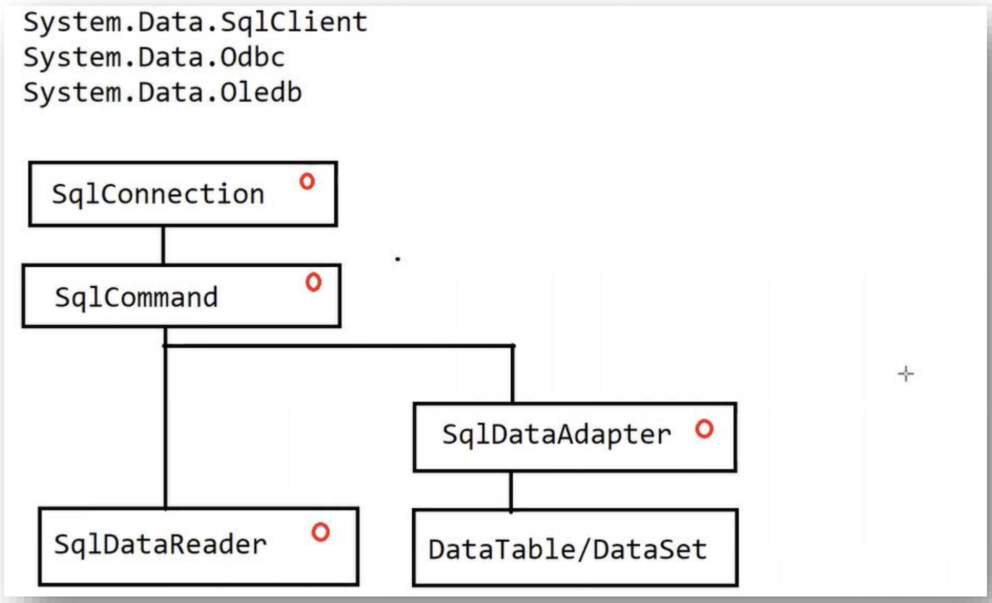
System.Data.DataSet

Collection of DataTables is DataSet

SqlAdaptor: Only required to create datatable/dataset and do following functions

1. Fill -→ from cmnd populate datatable/set
2. Read and update the records in db as per datatable/dataset



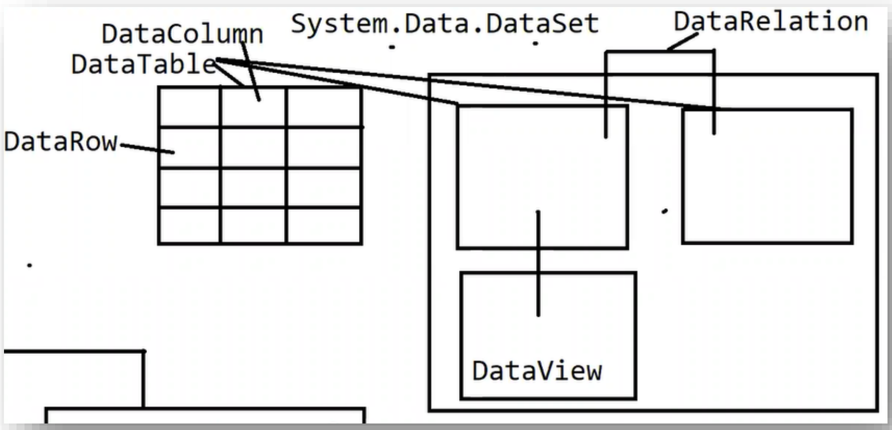


O indicate change as per database

DataRelation:

DataVeiw: allows filtering and sorting of DataTable, it is based on single DataTable, changes made in dataview will reflect in DataTable.

1 DataTable can have multiple DataViews



.

.

.

.

Private and Shared assembly:

When we add reference of any assembly ‘x’ in our assembly we get private copy of that assembly in our assembly folder.

Suppose this x assembly is refered by many other assemblies and any changes are made in original x then we need to manually update these changes in all assemblies.
To avoid this we can insatll that assembly in GAC (Global assembly cache) [This is same as earlier practice of COM]

Putting assembly in GAC

1. Only assembly having a strong name can be put in GAC(Global Assembly cache):

   Generate an assembly with strong name: name of assy,version,location,key file, culture,key pair:

   To generate strong named assy, assy must be signed with a key pair

Signing is done in project properties (Build an)

Generate a key pair

```
-cmd line-> sn -k  filename or in project property
```

2. Install the assy in GAC

```
-cmd line-> gacutil /i Asm1.dll ------- (i-install)
  to uninstall  gacutil /u Asm1 ------(u-uninstall)
```

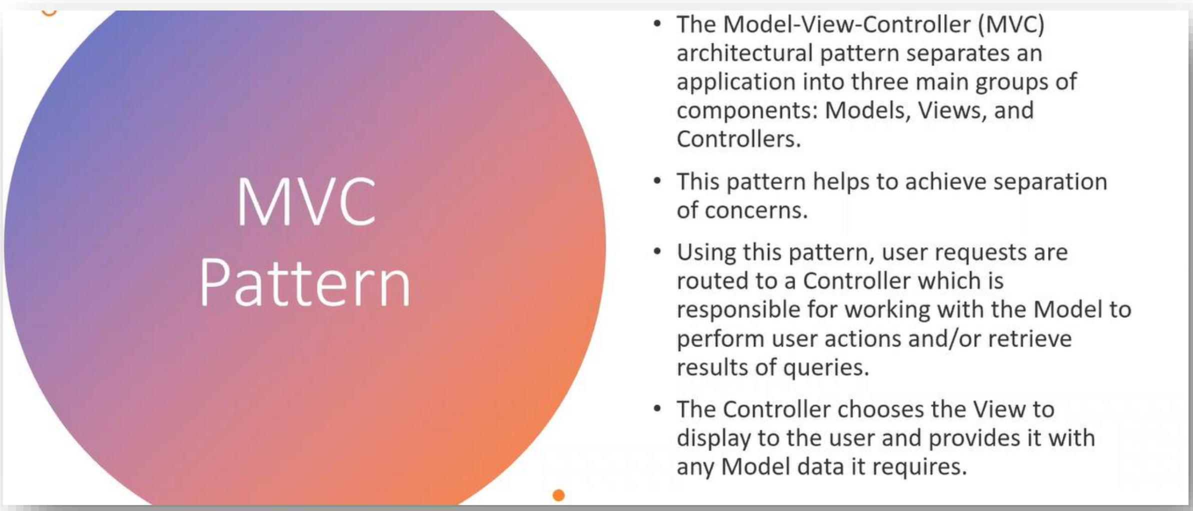
Day11

Asp.Net Web forms- Event model----- easier, less control

Asp .Net MVC- MVC pattern---- comparatively more code,more control,better performance

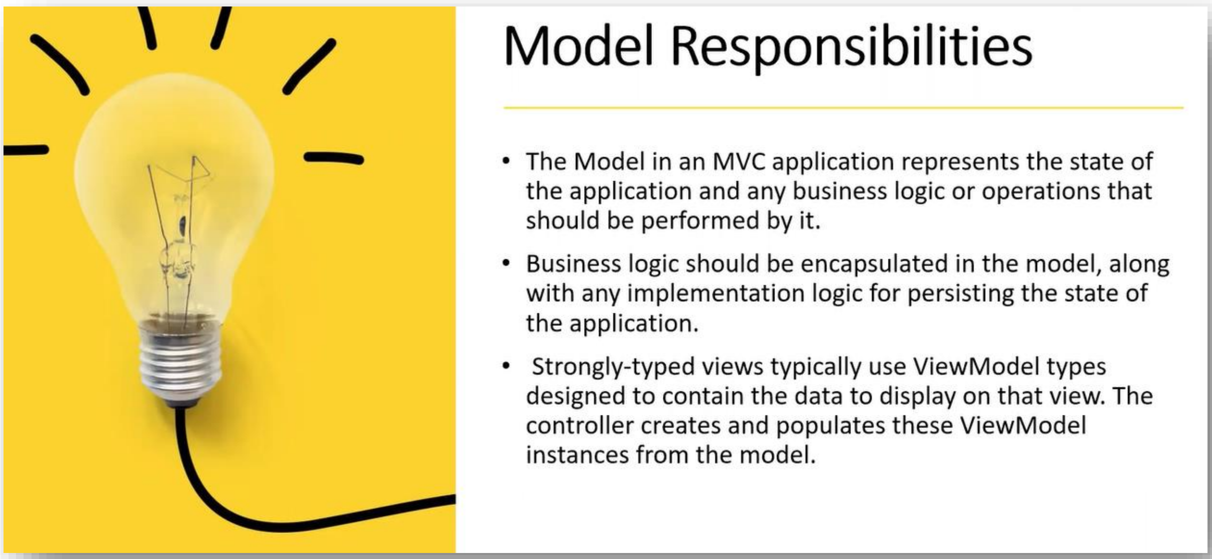
More extensible,more popular

MVC: Model—View—Controller ------ sepration of concenrs,



MVC Pattern

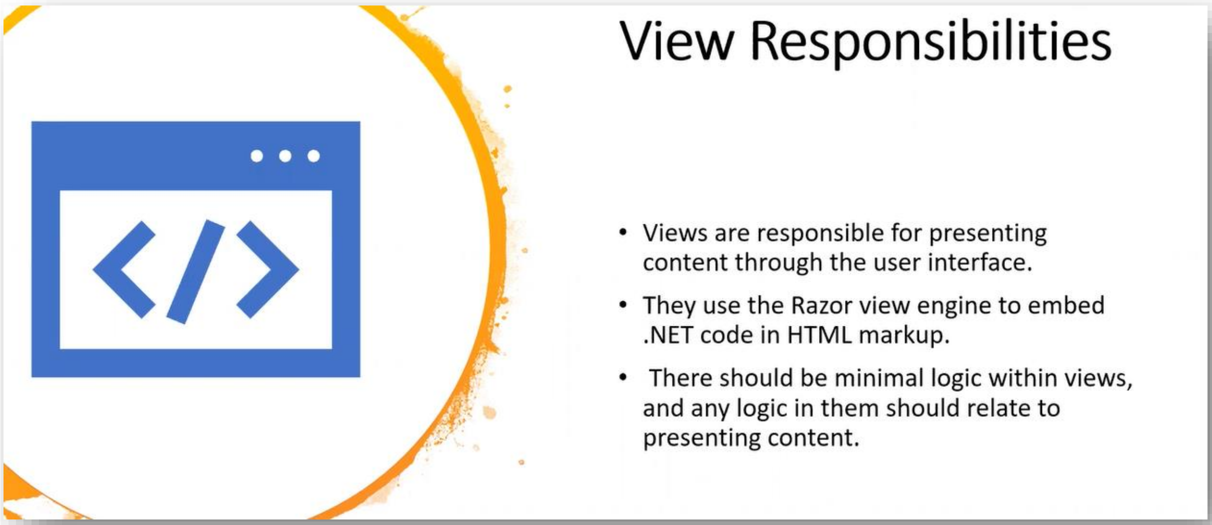
- The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers.
- This pattern helps to achieve separation of concerns.
- Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries.
- The Controller chooses the View to display to the user and provides it with any Model data it requires.



Model Responsibilities

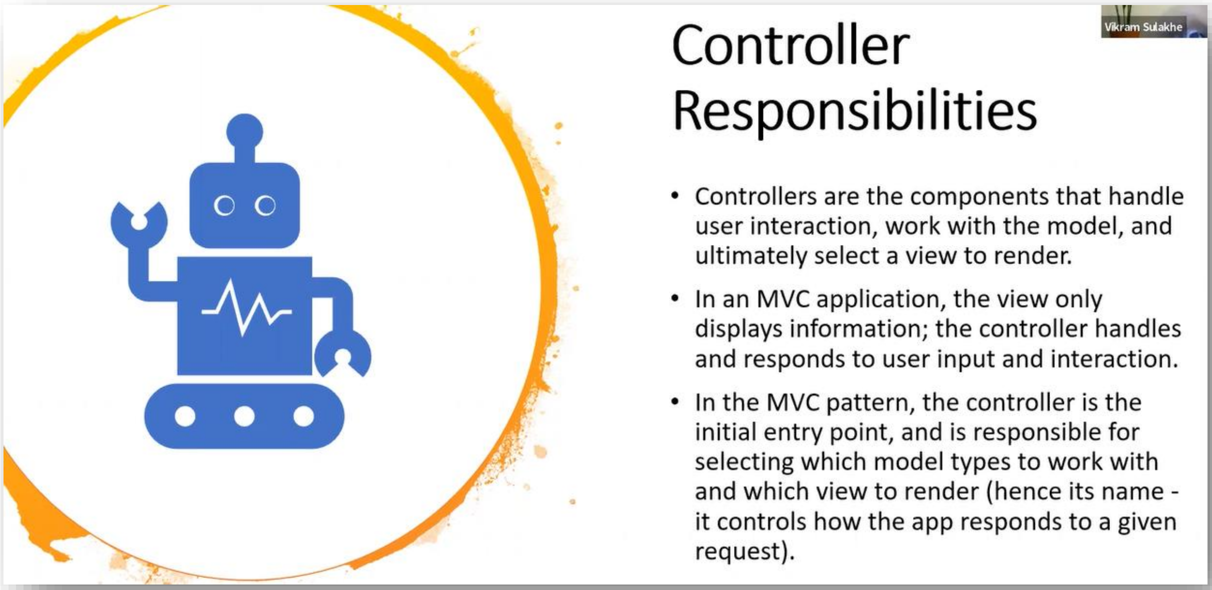
- The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it.
- Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.
- Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

State repersents data in application



View Responsibilities

- Views are responsible for presenting content through the user interface.
- They use the Razor view engine to embed .NET code in HTML markup.
- There should be minimal logic within views, and any logic in them should relate to presenting content.


Vikram Sulakhe

Controller Responsibilities

- Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render.
- In an MVC application, the view only displays information; the controller handles and responds to user input and interaction.
- In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render (hence its name - it controls how the app responds to a given request).

Action result and its derived classes

Understanding Action Results

A controller action returns something called an ***action result***. An action result is what a controller action returns in response to a browser request.

The ASP.NET MVC framework supports several types of action results including:

1. ViewResult - Represents HTML and markup.
2. EmptyResult - Represents no result.

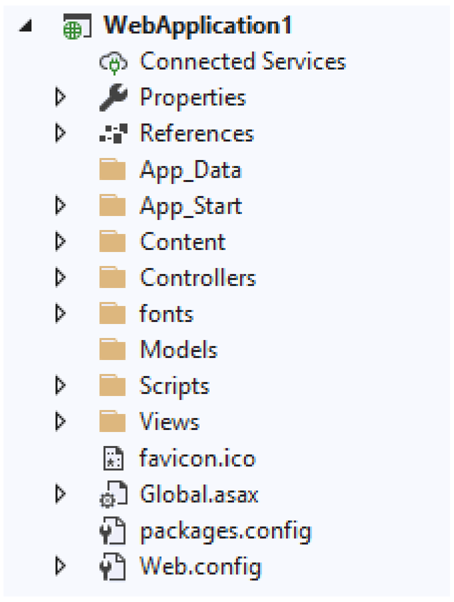
3. RedirectResult - Represents a redirection to a new URL.
4. JsonResult - Represents a JavaScript Object Notation result that can be used in an AJAX application.
5. JavaScriptResult - Represents a JavaScript script.
6. ContentResult - Represents a text result.
7. FileContentResult - Represents a downloadable file (with the binary content).
8. FilePathResult - Represents a downloadable file (with a path).
9. FileStreamResult - Represents a downloadable file (with a file stream).

All of these action results inherit from the base ActionResult class.

Inheritance Hierarchy

System.Object
  System.Web.Mvc.ActionResult
    System.Web.Mvc.ContentResult
    System.Web.Mvc.EmptyResult
    System.Web.Mvc.FileResult
    System.Web.Mvc.HttpStatusCodeResult
    System.Web.Mvc.JavaScriptResult
    System.Web.Mvc.JsonResult
    System.Web.Mvc.RedirectResult
    System.Web.Mvc.RedirectToRouteResult
    System.Web.Mvc.ViewResultBase

Folder structure and Important files of .Net MVC web application:



AppData--- > Local data base if any

AppStart---- > contains Bundle,Router,Filter Config files

Contents--- > css bootstrap files.

Model--- >

Views--- >Controllerfolder-- >cshtml files as per method name of controller

Views /Shared/_ViewStart.cshtml --gives default layout

```
Layout = "~/Views/Shared/_Layout.cshtml";

~ : root
```

Controlllers--- >

Web.Config---- >

Packages.config---- >

Global.asax--- >

```
public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }




public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id
= UrlParameter.Optional }

      // routes.MapRoute(
            //    name: "Default",
            //    url: "{controller}/{action}/{id}/{a}/{b}",
            //    defaults: new { controller = "Default", action =
"Index2", id = UrlParameter.Optional, a= UrlParameter.Optional, b=
UrlParameter.Optional }
            //);
            );
```

```
public class HomeController : Controller
    {
        public ActionResult Index()
        {
            //return HttpNotFound();

            return View();
            //return View("MyView1");
        }
public ActionResult Index2(int id=10, int a =20, int b=30)
        {
            ViewBag.id = id;
            ViewBag.a = a;
            ViewBag.b = b;

            return View();
        }
```

--// All controllers extends from Controller

All method returns ActionResult as it is base class for all results.

View() by default searches for method_name.cshtml in Views/Controller_name/… if not found then searches in views/shared folder

View("MyView1"); here view file name should be MyView1.cshtml

)

```
Layout = "~/Views/Shared/_Layout.cshtml";
```

_Layout.cshtml: All formating is given in this layout and it contains @RenderBody() which is must and can be mentioned only once, else error

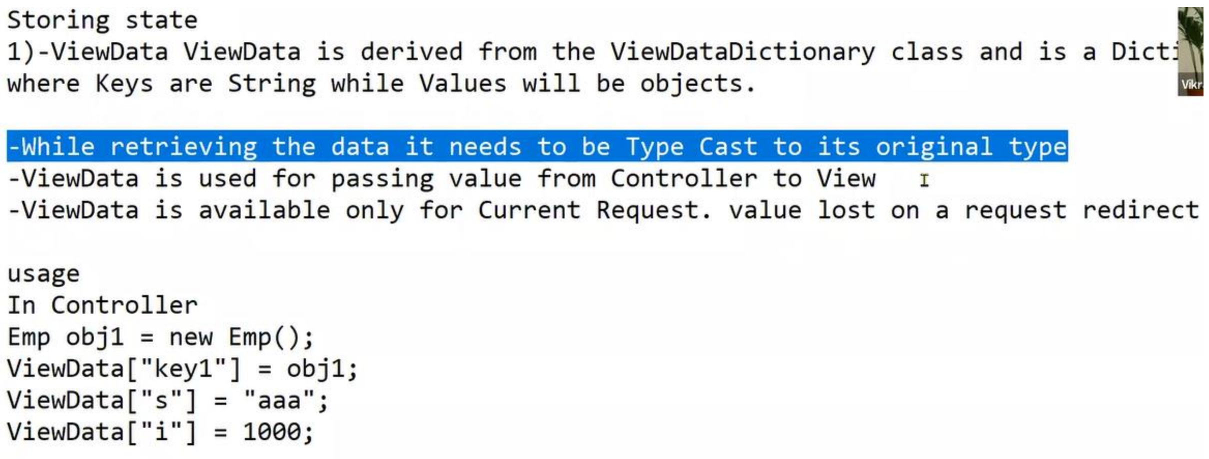
View: @{ } -- Razor code --- > .Net code inside html

@: result -- > html code inside razor code

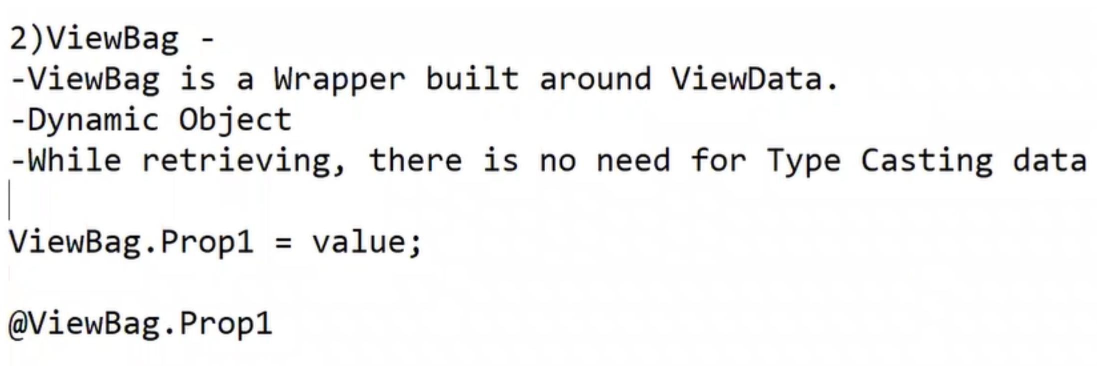
Index2.cshtml
```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index2</title>
</head>
<body>
    <div>
        without layout
        Id = @ViewBag.id<br />
        a = @ViewBag.a<br />
        b= @ViewBag.b<br />
        @{

            string s = "abc";
            int i = 100;
            i = ViewBag.a;
        }
        @{
            if (i % 2 == 0)
            {
                s = "even";
            }
            else
            {
                s = "odd";
            }
        }
        i = @i
        <br />
        s = @s
        <br />

        @{
            if (i % 2 == 0)
            {
                @:result = even<br />
            }
            else
            {
                @:result= odd<br />
            }
        }

        <table border="1">
            @for (int j = 1; j < 5; j++)
            {
                <tr>
```

```
                    <td>5</td>
                    <td>*</td>
                    <td>@j</td>
                    <td>=</td>
                    <td>@(j*5)</td>
                </tr>

            }
        </table>

    </div>
</body>
</html>
```

Day12



```
Storing state
1)-ViewData ViewData is derived from the ViewDataDictionary class and is a Dicti
where Keys are String while Values will be objects.

-While retrieving the data it needs to be Type Cast to its original type
-ViewData is used for passing value from Controller to View
-ViewData is available only for Current Request. value lost on a request redirect

usage
In Controller
Emp obj1 = new Emp();
ViewData["key1"] = obj1;
ViewData["s"] = "aaa";
ViewData["i"] = 1000;
```



```
2)ViewBag -
-ViewBag is a Wrapper built around ViewData.
-Dynamic Object
-While retrieving, there is no need for Type Casting data

ViewBag.Prop1 = value;

@ViewBag.Prop1
```

3)TempData
TempData is derived from the TempDataDictionary class and is a Dictionary object where Keys are String while Values will be objects.
While retrieving the data it needs to be Type Cast to its original type
TempData is available for Current Request. It will not be destroyed on redirection.

Between requests
4) QueryString (passed values is stored in the url)

5) Session Variables
Available for Session

```
Session["key"] = value;
to store value ...
Session["i"] = 100;
To read value....
int i = (int)Session["i"];
```

6) Application variables(common data for all users)
Available across sessions

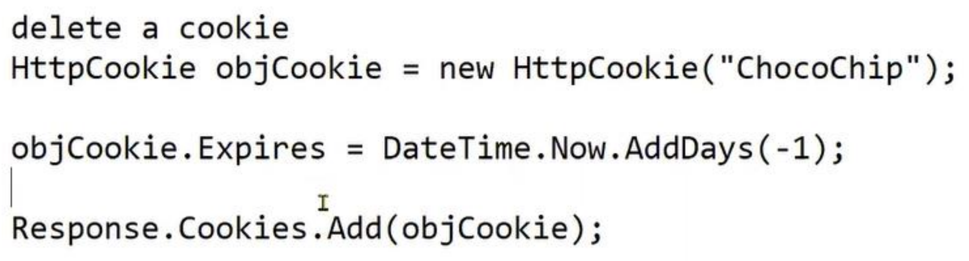
```
System.Web.HttpContext.Current.Application["Name"] = "Value";
```

7) Cookies

```
HttpCookie objCookie = new HttpCookie("ChocoChip");

objCookie.Expires = DateTime.Now.AddDays(1);
objCookie.Value = "a";

//objCookie.Values["key1"] = "a";
//objCookie.Values["key2"] = "b";

Response.Cookies.Add(objCookie);
```

read a cookie

```
HttpCookie objCookie = Request.Cookies["ChocoChip"];
//null if not present
string s;
s = objCookie.Value;
//s = objCookie.Values["key1"];
```

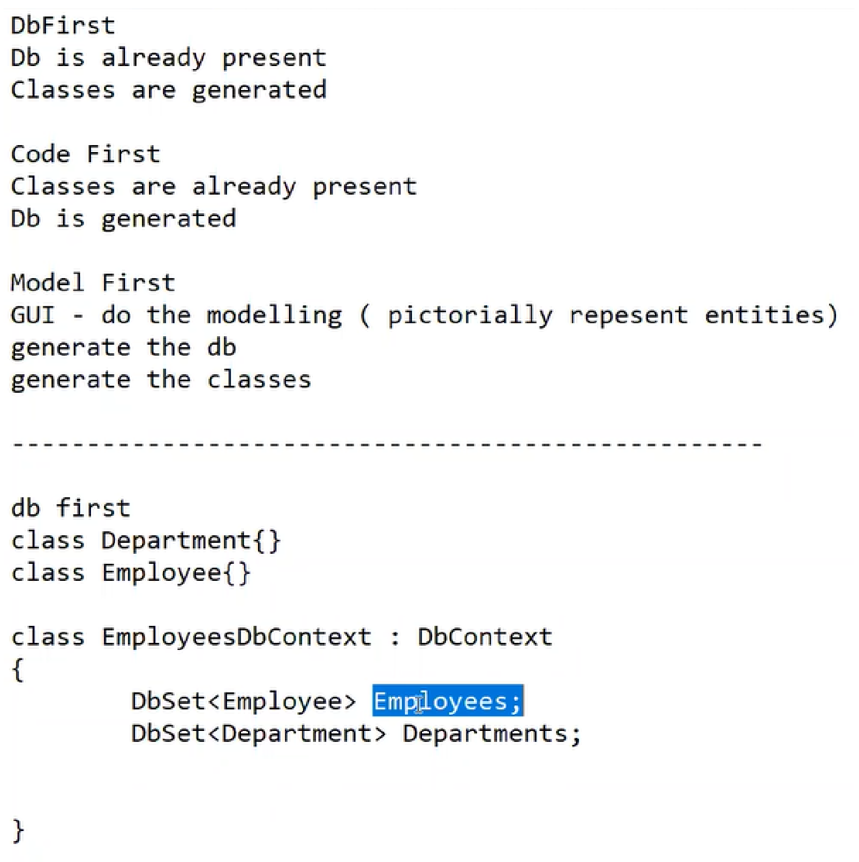


```
delete a cookie
HttpCookie objCookie = new HttpCookie("ChocoChip");

objCookie.Expires = DateTime.Now.AddDays(-1);

Response.Cookies.Add(objCookie);
```

Day-13

2nd half

Entity Framework:



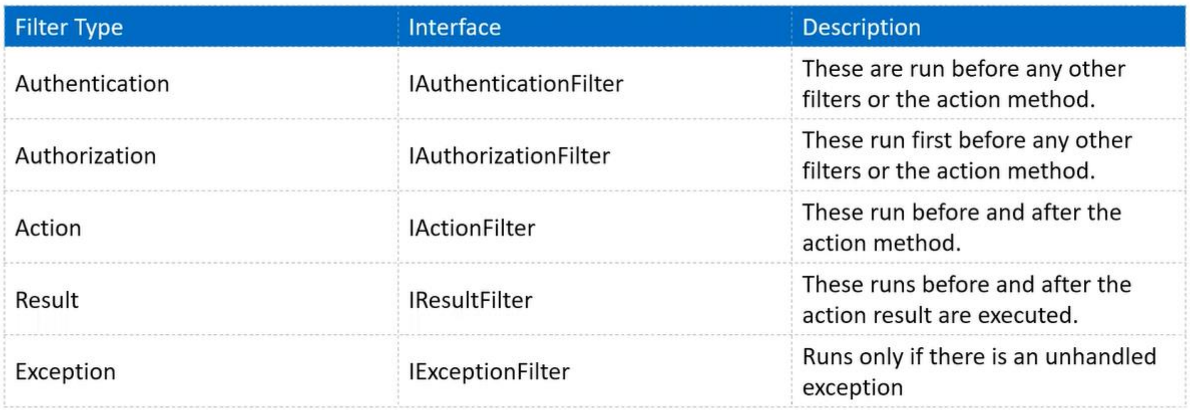
```
DbFirst
Db is already present
Classes are generated

Code First
Classes are already present
Db is generated

Model First
GUI - do the modelling ( pictorially repesent entities)
generate the db
generate the classes

--------------------------------------------------

db first
class Department{}
class Employee{}

class EmployeesDbContext : DbContext
{
        DbSet<Employee> Employees;
        DbSet<Department> Departments;


}
```

Day14

Filters:

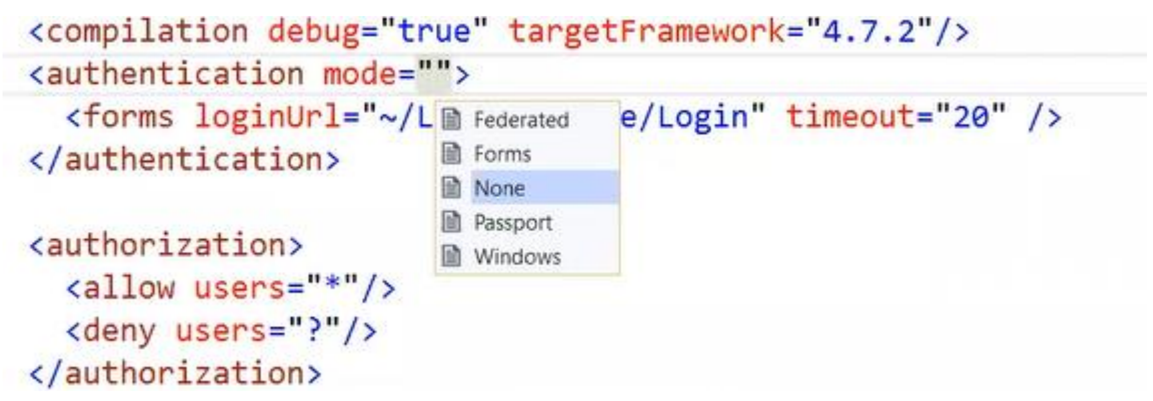
Executing -Before

Executed -After

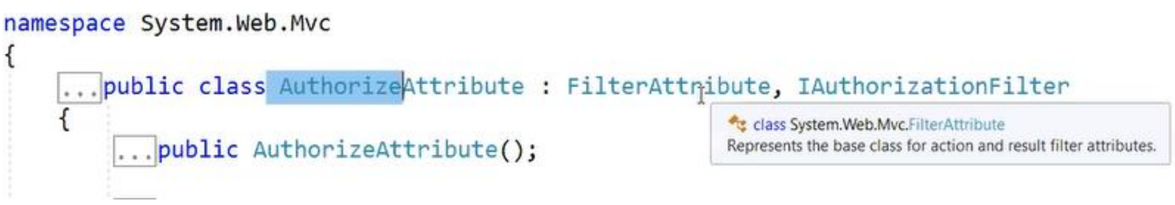


Filter Type	Interface	Description
Authentication	IAuthenticationFilter	These are run before any other filters or the action method.
Authorization	IAuthorizationFilter	These run first before any other filters or the action method.
Action	IActionFilter	These run before and after the action method.
Result	IResultFilter	These runs before and after the action result are executed.
Exception	IExceptionFilter	Runs only if there is an unhandled exception

Authentication:

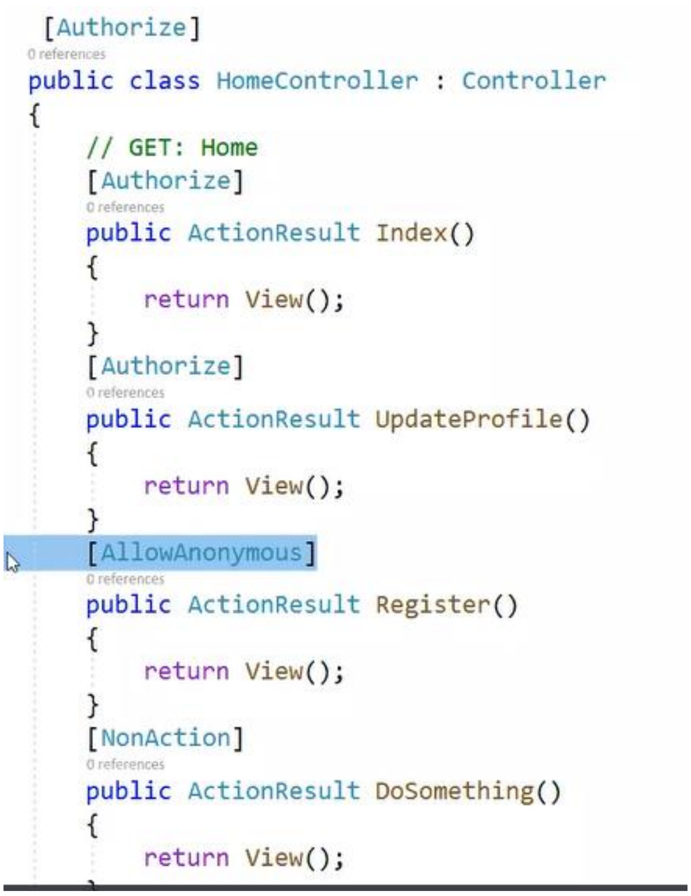


```
<compilation debug="true" targetFramework="4.7.2"/>
<authentication mode="">
  <forms loginUrl="~/L                e/Login" timeout="20" />
</authentication>

<authorization>
  <allow users="*"/>
  <deny users="?"/>
</authorization>
```

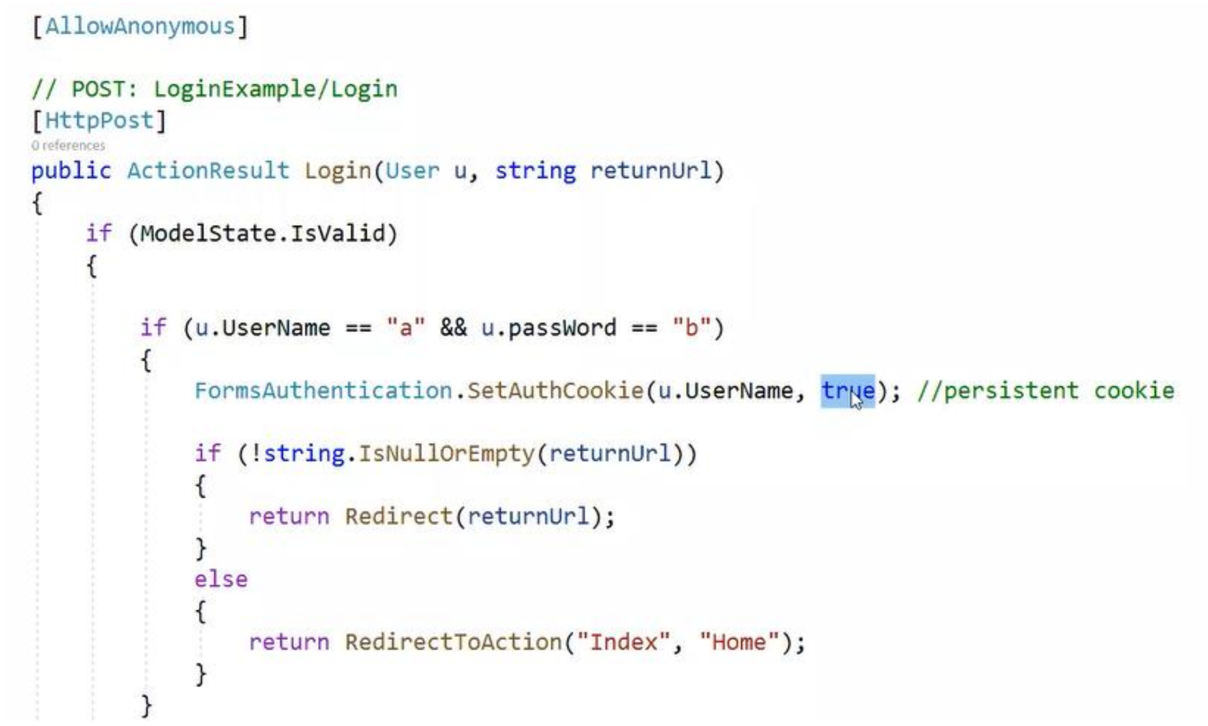


```
namespace System.Web.Mvc
{
    ...public class AuthorizeAttribute : FilterAttribute, IAuthorizationFilter
    {
        ...public AuthorizeAttribute();
```

class System.Web.Mvc.FilterAttribute
Represents the base class for action and result filter attributes.



```
 [Authorize]
0 references
public class HomeController : Controller
{
    // GET: Home
    [Authorize]
    0 references
    public ActionResult Index()
    {
        return View();
    }
    [Authorize]
    0 references
    public ActionResult UpdateProfile()
    {
        return View();
    }
    [AllowAnonymous]
    0 references
    public ActionResult Register()
    {
        return View();
    }
    [NonAction]
    0 references
    public ActionResult DoSomething()
    {
        return View();
```



```
[AllowAnonymous]

// POST: LoginExample/Login
[HttpPost]
0 references
public ActionResult Login(User u, string returnUrl)
{
    if (ModelState.IsValid)
    {

        if (u.UserName == "a" && u.passWord == "b")
        {
            FormsAuthentication.SetAuthCookie(u.UserName, true); //persistent cookie

            if (!string.IsNullOrEmpty(returnUrl))
            {
                return Redirect(returnUrl);
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }
        }
```

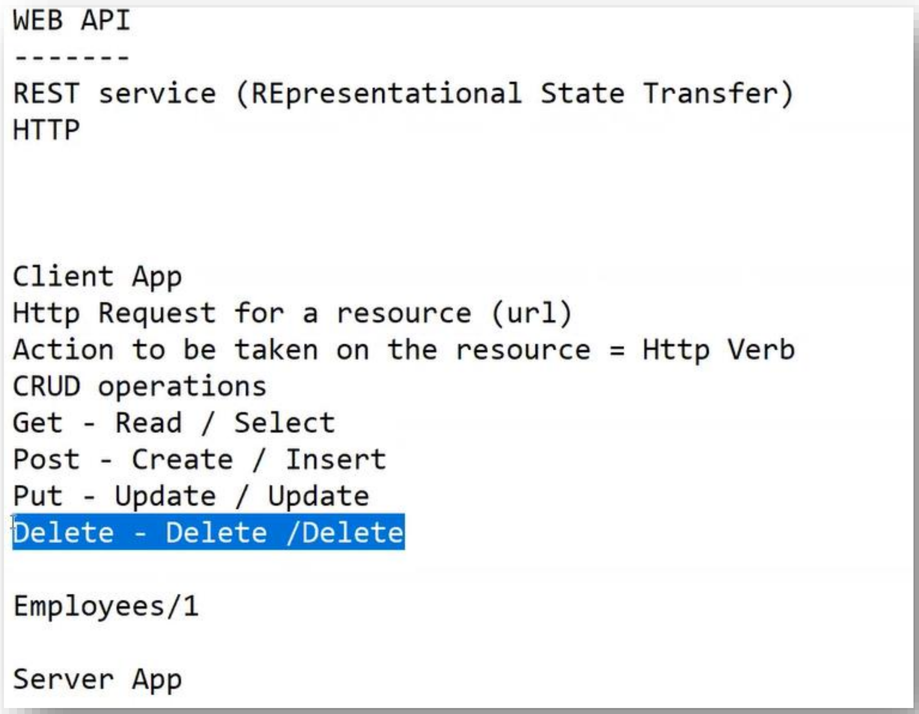
WCF

Day 15

Duplex

Channel Factory

WEB Api



```
WEB API
-------
REST service (REpresentational State Transfer)
HTTP



Client App
Http Request for a resource (url)
Action to be taken on the resource = Http Verb
CRUD operations
Get - Read / Select
Post - Create / Insert
Put - Update / Update
Delete - Delete /Delete

Employees/1

Server App
```