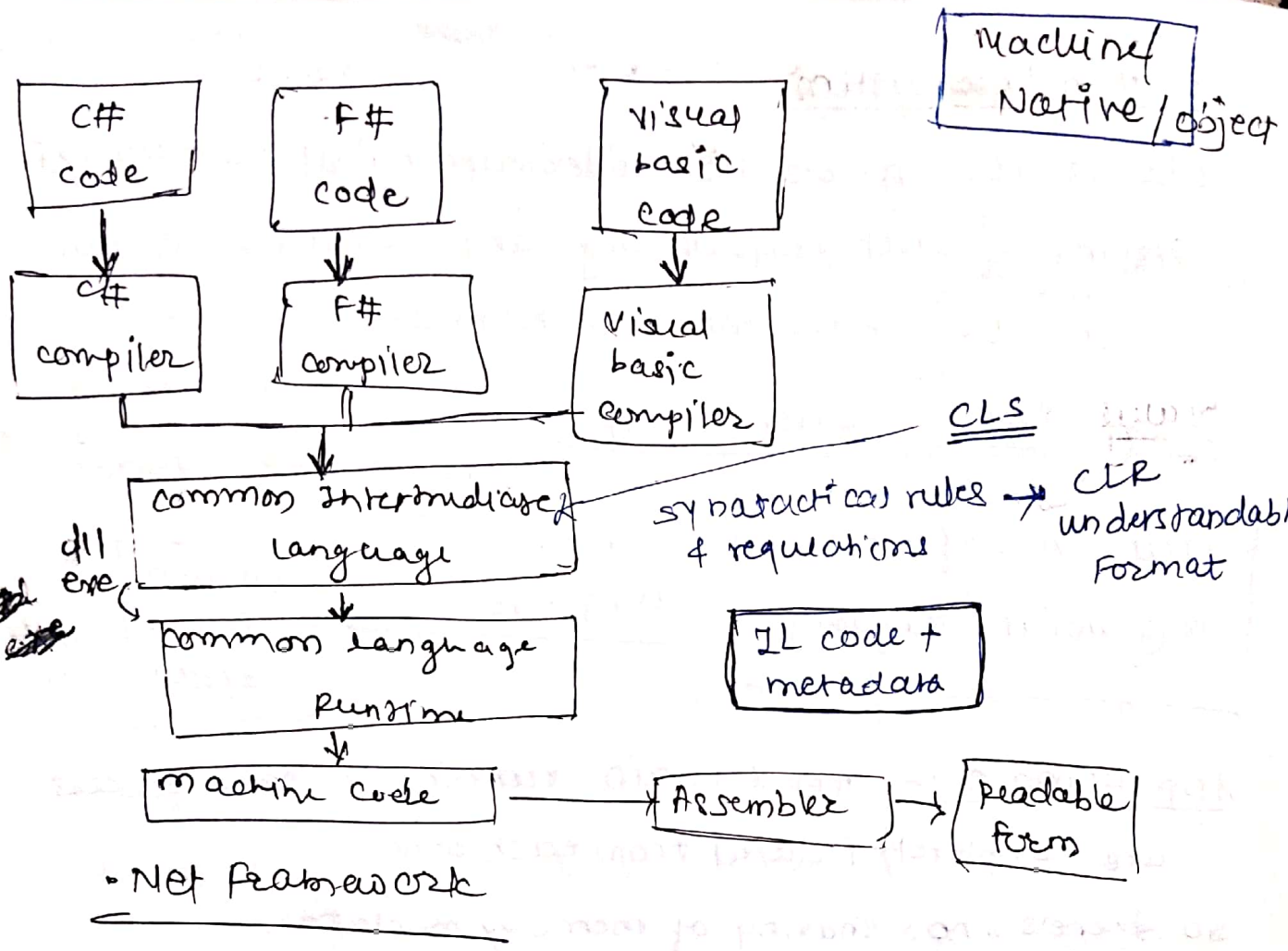
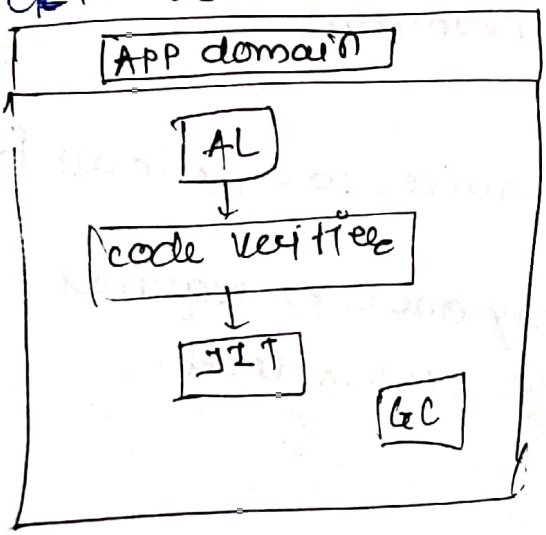


CLR uses JIT which converts MSIL code → machine code



managed code

MSIL code managed by CLR is known as

unmanaged code

Managed by OS.

Assembler :- Assembly lang. code → machine code

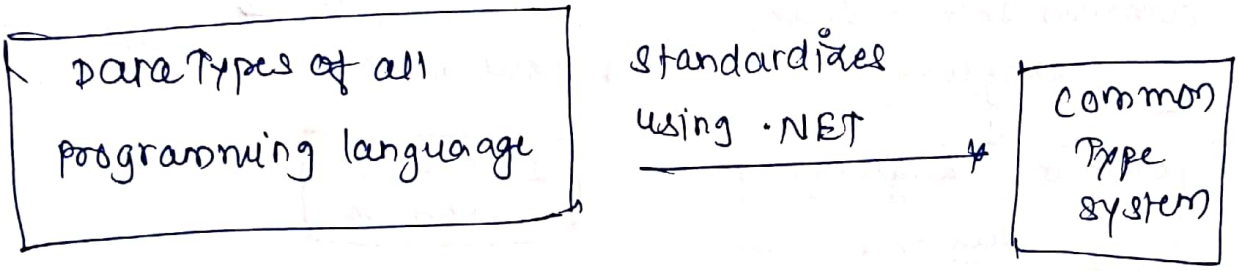
JIT :- high level programming lang. code IL → machine code

Pre-JIT
Econo-JIT
Normal-JIT

Common Type System

CTS is responsible for understanding all the data type systems of .Net programming lang. & converting them in CLR understandable format.

value Type Reference Type



App-Domain :- App domain running in same process are completely isolated from each other.
so there's no sharing of memory or data.

Assembly : A chunk (precompiled) code that can be executed by .NET runtime environment.

.exe .dll

private : exe or dll :- requires to copy in all folders

public/shared : dll :- only one copy required at system level.

.NET core	.NET Framework	Xamarin
- For cross platforms & new apps & services	For legacy/old apps	mobile apps only
- Windows, macOS, Linux	windows only	Android, iOS, macOS.

Windows forms + web forms + WCF

Removed.

Windows Presentation Foundation (WPF)

Legacy apps :- old application program of relating to or being a previous or outdated computer system.

- BCL - Base class library
  GAC - Global Assembly cache

Managed code	Unmanaged code
- managed by runtime environment CLR	- managed/executed directly by operating system.
- provides security to app.	- does not provide security to app.
- Memory buffer overflow does not occur	- Buffer overflow, memory leak etc occured.
-	- Memory allocation, type safety, security are managed by developer.
- App. written in C#, VB.net	- app. written in C, C++

ILDASM - Intermediate Language Disassembler :-

It is an utility which shows the information of assembly in human-readable format by passing an .NET framework DLL.

- shows namespaces, types & interfaces of DLL

→ shows info. about .dll or .exe

cmd → ildasm → open dll/exe file in it → {show info. in window}

Session 5 : C# Basics

Everything is an object

```
using system;
  namespace HelloWorld App {
      class Hello {
          static void main(string[] args) {
              Console.WriteLine("Hello World");
              Console.Read();
          }
      }
  }
```

Value Type

Reference Type

Value Type: primitive, Enum, struct, Nullable

primitive
- int
- float
- char
- boolean

Reference Type: class, array, interface, delegate

Boxing :- Value Type ⇒ Object Type ⇒ Object obj = 100;

Unboxing :- Object Type ⇒ Value Type

↓

```
int i=123;
object ob = i;
    int j = (int) ob;
```

C# programming structure

- Namespace declaration.
- A class or structs or interfaces or enums, or delegates.
- Members : constants, fields, methods, properties, indexers, events, operators, constructors, destructors
- Statements & Expressions
- comments

Access specifiers

Abstraction → Allows making relevant information visible.

Encapsulation → Enables a programmer to implement desired level of abstraction.

C# supports following access specifiers

1. public
2. private
3. protected
4. Internal
5. protected internal

① Internal :- Accessed from any class or method defined within the application in which member is defined.

② protected Internal :- Accessed by only child class within the same application.

	Entire program	containing class	current assembly	derived types	derived types within curr. assem
public	✓	✓	✓	✓	✓
private	X	✓	X	X	X
protected	X	✓	X	✓	✓
Internal	X	✓	✓	X	✓
Protected Internal.	X	✓	✓	✓	✓
private protected	X	✓	X	X	✓

	Nested Types		Non-nested types	
namespaces →	public	, none	public	none
class members →	private	, All	internal	public, Int.
struct members →	private,	public internal private	internal	public, Inter
Interface →	public	, All	internal	public, inter
Enums →	public	, All	public	public, inter
delegate →	private	, All	internal	public, internal
constructor →			private	All
method →			private	All

↑ Consider these default

Java { packag- private ⇒ default }

classes, enums, interfaces & annotation types ⇒ package private / default

Fields, methods, nested type decl. & constructors ⇒ ~~default~~ default.

System.object ⇒ Root of .Net type hierarchy.

System.object ⇒ root of .net type hierarchy

⊙ Optional parameters :-

public void add (int x, int y, int z=30) {

}

⟹ add (10, 20, 40);

⟹ add (10, 20); This is allowed because we give default value to z.

⊙ Named parameters

⟹ add(x: 10, y: 20, z: "Sanket")

sending parameter value using variable name / parameter name

⊙ positional parameters :-

parameters which that are passed in order in which they are defined in the method signature.

⊙ params :- used as a parameter which can take variable number of parameters / arguments.

⟹ void display student (params int [] marks) {

⟹ void display (params object [] a) {

allowed any data type.

⊙ properties

get, set

get accessor

set accessor

Read only → only get

write only → only set

static program() {

Console.writeline()

}

single | Hierarchical | multilevel | multiple (Interface)

Constructors

static → called only once → singleton pattern
private → object created only within that class.

Emp emp = new Emp() { Emp ID = 10; Emp Name = "Alex"; }

object Initializer { } → copy constructor

session 4

Static members of class

① static ~~fields/members~~ variables

single copy of variable created

② static methods

→ static field are directly accessed by static methods
→ Non-static field requires object creation, for accessing them

③ static properties → having getter & setter.

static int EmpID;

in main() program. emp ID = 101; → program. EmpID;

④ static constructor

called only once → singleton pattern

⑤ static class :-

→ only contain { static data members, static methods, & static constructor }
→ can't create object of static class.
→ It acts as a sealed class → can't inherit.

Inheritance

Case 1 :- only derived class constructor created ⇒ Default constructor of base class called automatically

Case 2 :- Both class has their constructor. ⇒ To execute constructor of base class `base` keyword used

Method overriding ⇒ Run-time → Dynamic

Method overloading ⇒ compile-time → Static

Multiple Inheritance ✗ ⇒ Interfaces

overriding :-

2 Types of keywords for method overriding.

- virtual keyword :-

Use this keyword in base class.

- override keyword :-

Use with derived class methods

```
⇒ class Base {
        protected virtual string GetMood() { }
  }

⇒ class Derived : Base {
        protected override string GetMood() { }
  }
```

sealed class ⇒ can't be inherited

sealed method ⇒ can't be overridden

~~method~~ If you want declare method as sealed

⇒ declare it as virtual in base class.

Abstract classes

- Can't be instantiated → can't create object
- serves as a base class for others classes.
- must have at least one abstract method.

Abstract method

- declared inside only abstract class
- must be implemented in all other non-abstract classes using override keyword.

Session 5

Interface method → public & abstract by default

Interface

- same like abstract class → All methods are abstract
- default method in interface → we can write method body
- Implementation must provided by class or struct

```
internal class Shape : ICalculate {
    public double calculateArea(float x) {
        return 3.14 * x * x;
    }
}
```

(1) Implicitly implementing

Explicitly implementing interface (2)

```
internal class shape : ICalculateShape {
    void ICalculateShape.calculatePerimeter(- -) {
    }
```

<InterfaceName>.<MemberName>

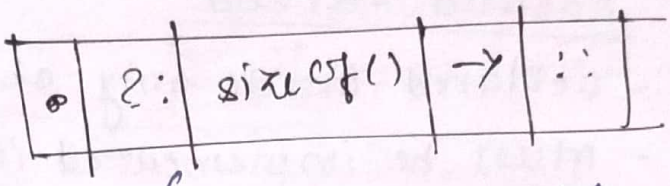
Don't use public access modifier.

Ia : Ib : Cc

Class Cc must provide impl for both interfaces

Default interface methods / virtual extension methods

- Allowed to use access modifiers with default methods
- If class implement → Interface
  that class doesn't know anything about default method



① Operator overloading :-

```
public static Calc operator+(Calc a1, Calc a2){
    Calc a3 = new Calc();
    a3.num1 = a1.num1 + a2.num1;
    a3.num2 = a1.num2 + a2.num2;
    return a3;
}
```


++ , + , > , can be overloaded

conditional logical operator & assignment operator can't be overloaded

② IDisposable Interface

GC clean up managed resources

unmanaged resources ⇒ {file handles, network connections, database connections}

Freeing up unmanaged resources

① Declaring destructor as a member of class
② Implementing System.IDisposable interface

Dispose ()

To prevent dispose method running ⇒ GC.SuppressFinalize() twice

IComparable Interface ⇒ To sort elements

CompareTo() method.

```
int ~~Sort~~ CompareTo(Program obj) {
CompareTo return this.sal.CompareTo(obj.sal);
}
```

Session 6: Reference & value Types

value Types :- ① enum ② struct

① enum :- When you need a predefined list of values which do represent some kind of numeric or textual data, you should use an enum.

```
enum Days { Sun, Mon, Tue, Wed, Sat }
```

② struct :- It helps you to make a single variable hold related data of various data types.

```
struct Book {
    int ID;
    string Title;
    float price;
}
```

- struct cannot inherit
- struct have constructors but not destructors.
- struct can have method, field, properties
- struct can implement one or more interface.

⊙ Reference Types :-

Nullable Types

- Allows you to assign null value to a variable
- Works with value types not with reference types.

```
int ? a = null
```

OR

```
Nullable<int> a = null
```

Accessing value ⇒ `a.GetValueOrDefault()`

default ⇒ zero

?? ⇒ Null coelescing operator

```
int y = x ?? 7   {If x is null assign 7 to y}
```

HasValue → true → variable contains value

out & ref

ref : passes arguments to calling method by reference.

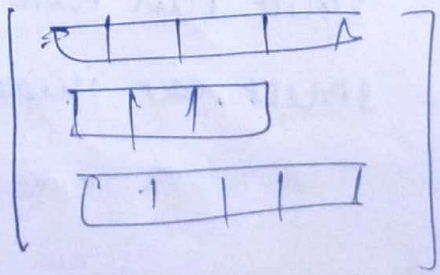
```
void refFun(ref int i) {
        i = 30;
}
main() { int b = 20;   ← must
    refFun(&b);
```

sysout(b) ⇒ 30

out :-

```
void o(out int i)   must
    i = 5   ←
}
```

jagged array



⊙ Arrays (multidimensional Array) :-

int [,] masks = new int [4,2]

⊙ Jagged Array ⇒ An array whose elements are arrays

int [][] arr = new int [2][]; ← Jagged array of 2 arrays.

{ Array of Arrays }

int[] m = {4, 5, 6}
int[] m = new int[3] {4, 5, 6}

multidimensional & Jagged arrays are same. Both are arrays of array.

multidimensional ⇒ Have fixed size

Jagged ⇒ varying sizes.

⇒ int [,] multiarray = new int [3,4] ⇒ multi

⇒ int[][] jaggedArray = new int [][] { ⇒ Jagged

new int [] {1, 2, 3}
new int [] {4, 5, 6, 7}
new int [] {8, 9}

};

⊙ Indexer An indexer is a special type of property that allows a class or structure to be accessed like an array for its internal collection.

public string this [int index] {
↑ get { return names [index]; }
set { names [index] = value; }
}

names class

public, private, protected or internal.

private string [] names = new string [4]

Session 7

Generic class

```
public class P<T> {
    T data;
}
```

Generic Methods

Method declared with type parameters for its return type or parameters is called generic method.

Generic Constraint

If we want to restrict a generic class to accept only the particular type of placeholder then we need to use constraint.

```
public void showValues<x>(T value) where x : struct
{
    Console.WriteLine(value)
}
```

class → ref
struct → value

Collections : Generics & Non-generics :-

System.Collections → Namespace that contains non-generic collection types.

System.Collections.Generic → namespace includes generic collection.

- List<T>
- Dictionary<T key, T value>
- SortedList<T key, T value>
- Queue<T>
- Stack<T>
- HashSet<T>

Non-generic collections

- ArrayList
- SortedList
- Stack
- Queue
- HashTable

Generic

- ICollection
- IList
- IDictionary

} Interfaces

Session-8

⊙ **Delegates:-**

- contains memory address of methods that matches the same signature as the delegate so that it can be called safely with correct parameters types.
- Reference type data type
- can be declared inside or outside of the class

```
public delegate int mydelegate(int x, int y);
                                ↑ declare delegate
class Test Program {
    public int calculate(int a, int b) {   ← set
        return a+b;                          ← Target method.
    }
    static void main(string[] args) {
        Test Program obj = new Test Program();
mydelegate ~~Myd~~ ~~Calculate~~ d = new ~~Calculate~~ (obj.calculate)
                                    mydelegate ← delegate
                                    create instance.
Console.WriteLine(d(12,22));  ← Invoke delegate
                                d.Invoke(12,22)
```

multicast delegates ⇒ Delegate that points to multicast delegate.

- All methods are called in single call in FIFO order

`+=` add delgates `-=` remove delegates.

In multicasting of delegates, method should have return type void otherwise it will return only value of last method.

FIFO ⇒ called the function

① Function delegate

Zero or more input parameters

One out parameters

Func<int, int, int> add = addNum;

```
int addNum(int x, int y)
{
return x+y;
}
```

int result = add(10,20)

Console.WriteLine(result);

one out is must
0 or more input => 0 to 16

② Action delegate => Doesn't return value.

Action<int, int> obj1 = addNum; obj1(10,20);

```
void addNum(int x, int y) {
    Console.WriteLine(x+y);
}
```

16 Types of input

Input 1 or more
output 0

③ Predicate delegate => must take 1 input pasameters & return boolean => True / False

```
bool checkvalue(int x) {
    return x>5;
}
main() {
    Predicate<int> obj1 = checkvalue;
    Console.write(obj1(10));
}
```

④ Anonymous method A method without name called anonymous method.

- method defined using delegate keyword & can be assigned to variable of delegate type.

public delegate void mydelegate(int val);

class TestProgram {

Method defined using delegate keyword

mydelegate d = delegate(int val) {
    Console.WriteLine(val);
};

Variable of delegate type

d(10);

}

◉ lambdas :-

=> Read as "goes to" operator

sum => sum+3

Session 9: Exception Handling

Exceptions :- An abnormal condition that arises in a code sequence at run-time.

- When an exceptional condition arises, an object representing that exception is created & thrown in the method that caused the error.

Checked & Unchecked

checked — checked keyword is used in C#

```
checked {
    int n = int.MaxValue;
    Console.W(n+10);
}
```

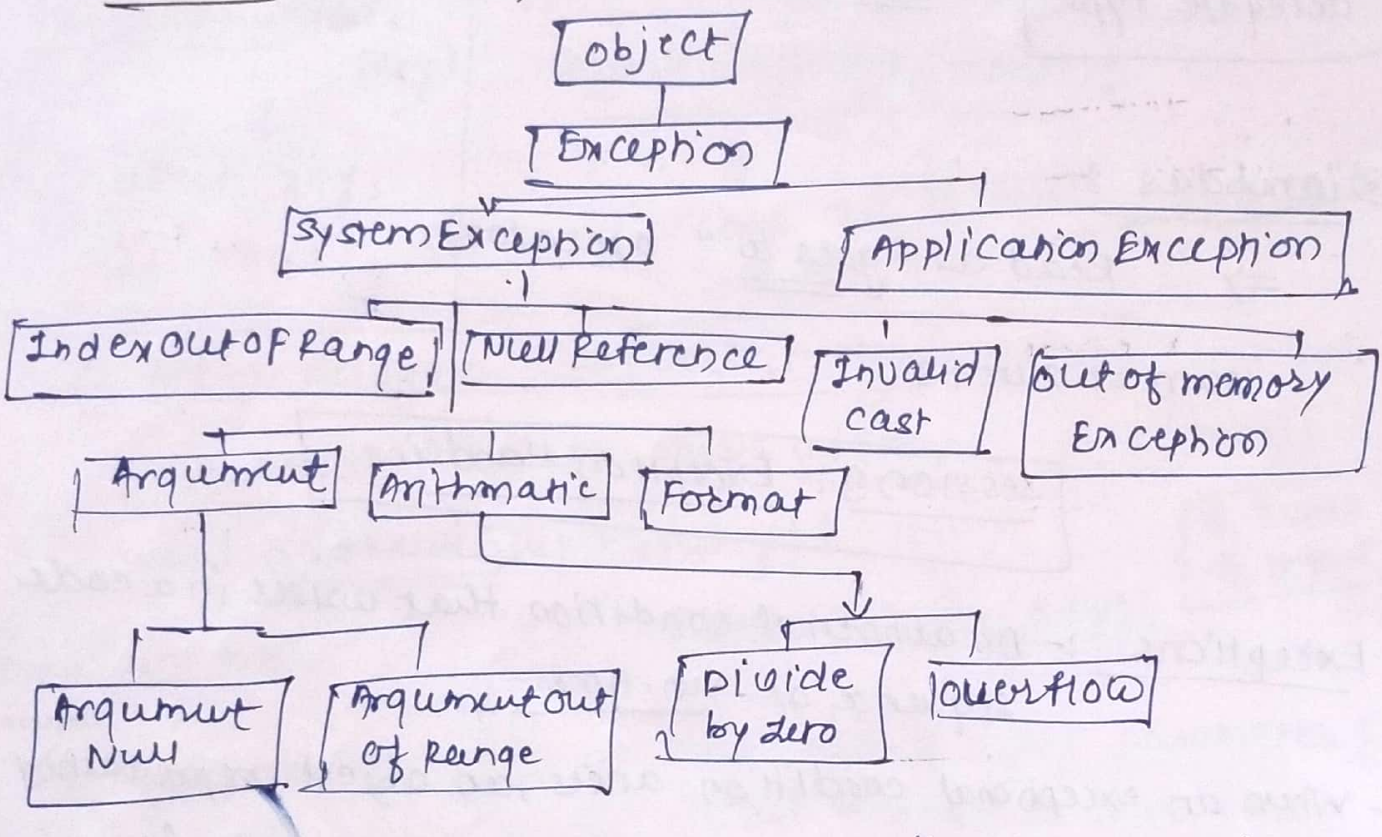
→ checked exception are exceptions that the compiler enforces you to handle or declare.

- derived from exception class

② Unchecked Exceptions ⇒ compiler doesn't enforce you to handle.

```
unchecked {
    int n = int.maxValue;
    C.W.(n+10);
}
```

Exception Hierarchy



Try → Holds the code that may throw an exception

catch → catches the exception thrown by try block

finally → This will execute whether exception is thrown or not

throw → used to throw exception manually.

③ Events

class that sends or raises an event ⇒ publisher

class that receives or handles the event ⇒ subscriber

publisher An object that contains the definition of the event & the delegate.

⇓

This calls the method in subscribes class

subscribes object that accepts the event & provides an event handler

Declare delegate

public delegate string MyDelegate(string str);

Declare event

event MyDelegate MyEvent; publisher class

Raise the event

MyEvent.Invoke("India");

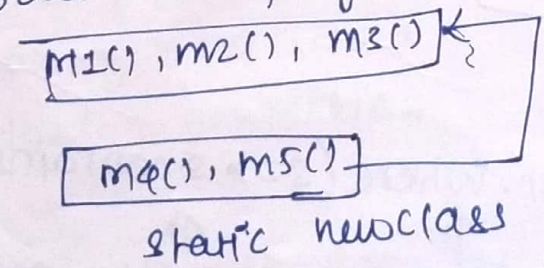
session 10: partial classes & LINQ

⊙ Anonymous Types

var student = new { rollNo = 100, name = "Alex", add = "pune" }

properties :- read only, can't be initialized with null, anonymous func or a pointer type.

⊙ Extension methods :- Add new methods in the existing class without modifying the structure.



⊙ partial class

public partial class student {

}

① All files in must be same assembly

② splitting impl. in diff .cs files

partial method ⇒ declaration in one partial class, definition part in another partial class.

⇒ public partial void displayStudent()

[Both classes are partial]

[Nested partial Types are allowed]

LINQ - Language Integrated Query

Two ways to write LINQ query to [IEnumerable] collection or [IQueryable] data sources.

1. Using Query syntax
2. Using Method syntax

[IEnumerable
IQueryable]

① Using Query syntax

var a = from object in datasource ← Initialization
where condition ← condition
select object ← selection. or select Group By

[from items in list
where items >= 44
select items]

var can be used to store result

② Using method syntax

var result = strlist.Where(s => s.contains("Tutorials"));

Extension method ↑ Lambda expression.

Compile time error checking

LINQ to object supporting IEnumerable<T> for accessing in-memory data collections without any need of LINQ provider (API)

Deferred execution query is not executed when declared It executed when query object is iterated over a loop.

PLINQ (parallel LINQ) :- parallel implementation of LINQ to object

→ PLINQ enables query to automatically take advantage of multiple processors.

→ Increase speed of LINQ to objects

Session 11- Reflection & File IO

Creating shared assembly => shared assembly => GAC resides in

GAC => Global Assembly Cache |

Creating custom attributes

Traditional classes that are derive from System.Attribute

a) applying the ~~attr~~ Attribute Usage Attribute
b) Declaring attribute class
c) Declaring constructors
d) Declaring properties.

AttributeTargets
Inherited
Allow multiple

Attributes are metadata extensions that give additional info to the compiler about elements in the program code at runtime

Reflection objects used for obtaining type info. at runtime of classes, interfaces, enums.

System.Reflection

Reflection Allows

① Inspect types of properties, methods & events.

② Instantiate Types can create instances of types

③ Access & modify fields

Type class is used

```
Type type = typeof(myclass)

MethodInfo[] methods = type.GetMethods();
                       GetProperties();
```

④ File IO and streams :-

System.IO

```
string path = " . . ";
string[] lines;
If (File.Exists(path)) {
        lines = File.ReadAllLines(path)
                ReadAllText(path)
File.WriteAllText(path, lines);
File.WriteAllLines(path, lines);
File.AppendAllText
File.AppendAllLines
```

Stream ⇒ sequence of bytes passing through comm. path.

Input stream — Reading data from file

Output stream — Writing into file

TextWriter (parent) → StreamWriter (child class) — writing into files

```
StreamWriter sw = new StreamWriter("a.txt")
String str2 = Console.ReadLine() ⇒ Taking input
sw.WriteLine(str2); → To write a line in buffer
sw.Flush(); → To write in output stream
sw.Close(); → To close the stream.
```

Close()
Flush()
Write()
WriteLine()

StreamReader — Implements → TextReader

Seek() ⇒ Read / write at specific location from file.
↳ move the cursor

Close()
Peek()
Read()
ReadLine()
Seek()

BinaryWriter
BinaryReader

StreamWriter class — Implements → TextWriter

StreamReader class — Implements → TextReader

Stream ⇒ sequence of bytes passing through comm. path.

Input stream
Reading data from file

output stream
Writing into file

TextWriter (parent) → StreamWriter (child class)
writing into files

StreamWriter sw = new StreamWriter("a.txt")

Close()
Flush()
Write()
WriteLine()

String str2 = Console.ReadLine() ⇒ Taking input
sw.WriteLine(str2); → To write a line in buffer
sw.Flush(); → To write in output stream
sw.Close(); → To close the stream.

StreamReader → Implements → TextReader

Seek() ⇒ Read/write at specific location from file.
↳ move the cursor

Close()
Peek()
Read()
ReadLine()
Seek()

Binary writer
Binary Reader

StreamWriter class → Implements → TextWriter

StreamReader class → Implements → TextReader

⊙ Working with directories & drives :- | System.IO

Directory class → provides static method → creating, copying, moving & deleting directories

```
string[] drives = Directory.GetLogicalDrives();
```

✓ unary operators take one operand & can be overloaded

✓ binary take 2 → can be overloaded

✓ comparison → overloaded

✗ conditional logical operators → can't overloaded directly

✗ Assignment → can't overloaded.

.	::	→	sizeof()	::

Session 12 - Threading and Task

Threads:- Programs lines of execution **OR** smallest unit of processing that can be scheduled by an operating system.

Process	Thread
- Represents an application	- Represents module of app.
- Heavy weight component	- Light weight component
	- Executed inside process
- separate memory area occupied.	- common ~~are~~ memory area required / shared

`System.Threading.Thread` → class created → Lifecycle start

States in Thread

1) Unstarted → Instance of thread class created
2) Runnable (Ready to run) → start() called
3) Running
4) Not Runnable → sleep() or wait called
5) Dead (Terminated) → completed

Thread class properties	Thread class methods
- Current Thread	- Abort()
- Is Alive	- join()
- Is Background	- resume()
- Name	- sleep(int 32)
- Priority	- ~~sleep()~~ start()
	- suspend
	- yield()

Types of Thread

① Foreground Thread ⇒ keeps running until finish work even if main thread leaves.

② Background Thread ⇒ leaves its process as main thread leaves

Parameterized Thread start :-

Thread (ParameterizedThreadStart) constructor used to initialize new instance of thread class.

- Defines a delegate which allows object to pass to thread.
- Gives ArgumentNullException if parameter of this constructor is null

ThreadStart ⇒ It is a delegate which represents a method to be invoked when this thread begins executing. Thread (ThreadStart) constructor

Thread synchronization

Thread pool :- It manages & maintains a pool of worker threads, allowing for efficient & scalable execution of asynchronus operation.

- Thread reuse
- Dynamic sizing
- Task queing.
- Improved performance
- Asynchronus programming (async, await)

① Synchronizing Critical data using lock :-

lock keyword used

[Lock current Thread] → [Execute the Task] → [Release the lock]

- other threads does not interrupt the execution.

```
lock (this) {
        Console.WriteLine(Thread.CurrentThread.Name);
}
```

② synchronizing Critical data using Monitor class

Monitor { Monitor.Enter, Monitor.TryEnter, Monitor.Exit } — similar to lock

③ Synchronizing Critical data using Interlocked

Interlocked.Increment
Interlocked.Decrement
Interlocked.Add
Interlocked.Exchange

⊙ Working with Tasks :- (Task parallel library TPL)
used to implement Asynchronus programming

To create a Task ⇒
use Task.run method with a delegate or lambda expre.

```
Task myTask = Task.Run(() => DoWork());
myTask.Wait()
```

Ran(), Wait(), ContinueWith()

Task Return Type

Async Await ⇒ Asynchronus programming.

⇒ An async method can contain the await keyword if it typically returns a Task or Task<T> indicating that it is an asynchronus operation.

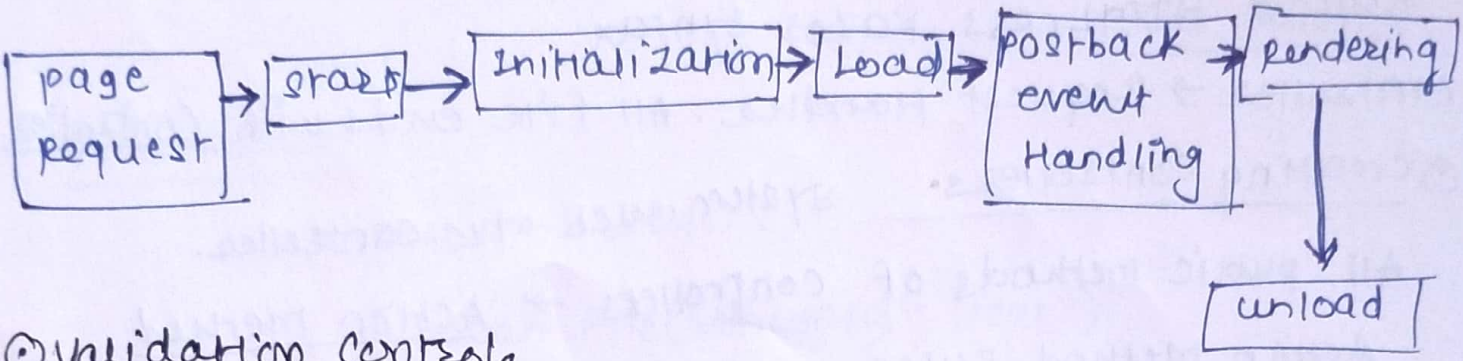
⇒ Await keyword used withing async method to asynchronously wait for the completion of a task without blocking the calling method.

Session 18: Intro to ASP.NET

ASP.NET

ASP.NET - server side Technology.

⊙ ASP.NET page life cycle



⊙ Validation Controls

① Required Field Validator

② Range Validator → Type, MinimumValue, MaximumValue.

③ Compare Validator → Type, ControlToCompare, ValueToCompare, Operator.

④ Regular Expression Validator → Validation Expression

⑤ Custom Validator → user defined

⑥ Validation Summary → Displays a report of all validation errors occured in web pages.

Session 14: Intro. to Asp.NET MVC

System.Web.mvc

View → HTML, css, Razor syntax.

controller → Request Handler. All file ends with Controller

⊙ Creating controllers — System.Web.Mvc.Controller.

All public methods of controllers → ACTion method

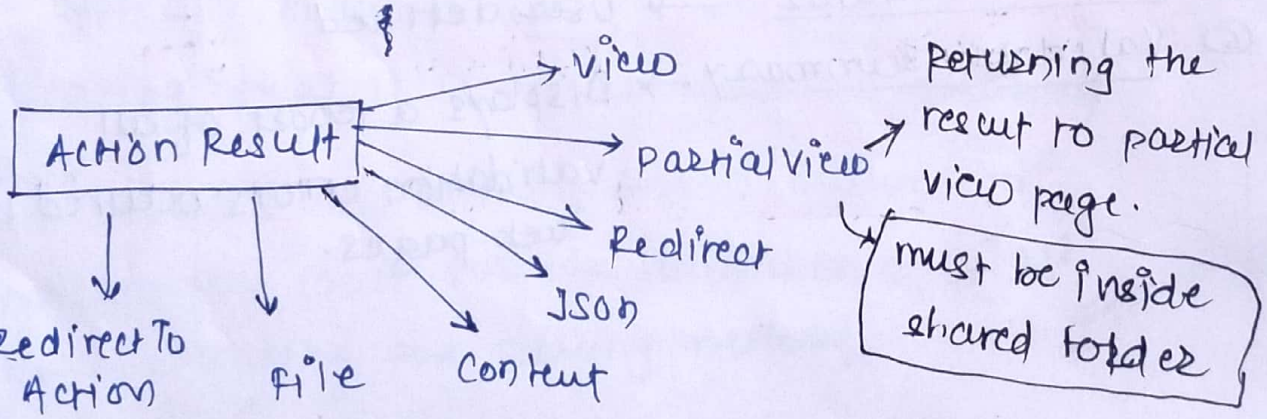
ACTion Method Rules

1) Must be public cannot be private or protected.
2) cannot be overloaded
3) cannot be static

BaseController

⇓

public class StudentController: Controller{ }



RedirectToACTion("GetStudents", "Student")

(optional) ← Controller ← ACTion

If not given finds ~~current~~ action in current controller.

```
public JsonResult GetAll(){
    List<Student> students = new List<Student>();
    return Json(students, JsonRequestBehavior.AllowGet);
}
```

Get method

File Result

```
ActionResult Index(){
    return File("web.config", "text");
}
```

web.

ActionResult

① RedirectToAction() ② File() ③ Content()

ViewResult

- View()

PartialViewResult

PartialView()

RedirectResult

Redirect()

JsonResult

Json()

Session-15 - Understanding views & state mgmt

① **Razor View Engine** :- write HTML + C#

ASP.NET MVC supports below Razor View Files ⇒

① .cshtml → C# code
② .vbhtml → Visual Basic code
③ .aspx → Asp.net web Form
④ .ascx → Asp.Net web Control

@ character

HtmlHelper class → @Html object

@Html.DisplayNameFor()

⊙ Standard HTML Helper Methods

@Html.ActionLink()
BeginForm()
:
TextBox()
CheckBox()
RadioButton()

⊙ Strongly-Typed HTML Helper

@Html.HiddenFor()
LabelFor()
TextBoxFor()

⊙ Validations using Data Annotation Attributes :-

System.ComponentModel.DataAnnotations

annotation representation ⇒ [Required]

1. Required
2. StringLength
3. Range
4. RegularExpression
5. CustomValidation
6. EmailAddress
7. MaxLength
8. MinLength
9. Phone

[Range(18,45)]

State management process where developers can maintain status, user state, & page information on multiple requests for same or diffn pages with web app.

① Client-side state mgmt :-
- Information is stored in customer system.
- e.g. ViewBag, TempData, ViewData, Cookies, Querystrings

② Server-side state mgmt
- store all information in server's memory
- e.g. session, Application

ViewBag ⇒ [Controller] → Data → [View]

ViewBag.Msg = "Hello" → @ViewBag.Msg
ViewBag.Emp = empl → @ViewBag.Emp

TempData ⇒
→ persist data for the duration of single HTTP request & is then automatically cleared.
→ After accessing data once → It will be cleared in next request
[Keep()]
→ passing data from [one action method] → [Another action method]

Cookies → plain Text File stored by client (Browser) tied to specific website.

Querystring → string variable that is inserted at the end of the page URL

⟹ RedirectToAction("Index", "Emp", new {Empid = 1, Ename = "abc"})

Viewpara :- [controller] → data → [view]

⊙ Server side state management :-

① session ⇒ store & retrieve values for a user.

Session ["empid"] = Convert.ToString(obj.Emp2D)

② Application :- store in memory of ~~user~~ server

Faster than storing & retrieving data from DB.

session → single user | Application - All users

HttpApplicationState class

Global.asax file used for handling app. events

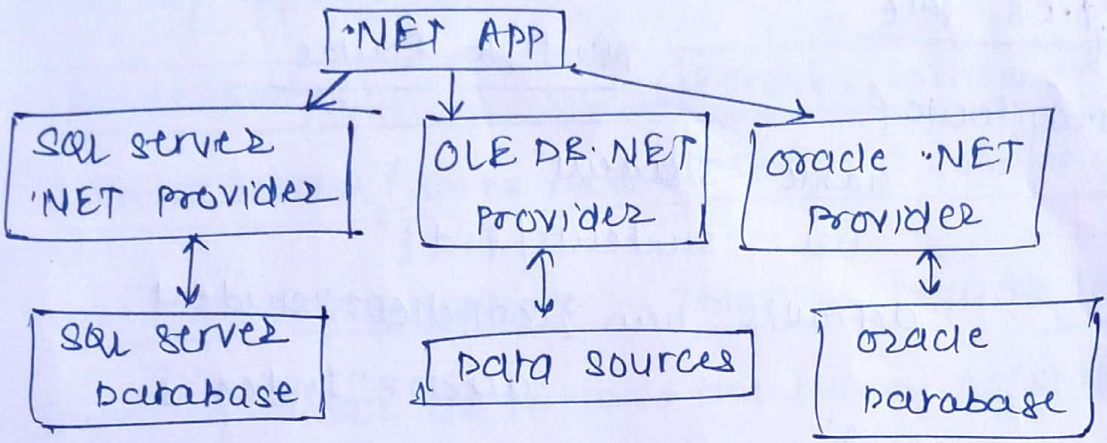
All routes are registered in Global.asax

Application_Start()

Session 16 : Data management with ADO.NET, Routing & Request Lifecycle.

ADO.NET ⇒ Data Access Technology

Allow .NET applications to connect to data sources, execute commands & manage disconnected data.



ODBC - open Database Connectivity.

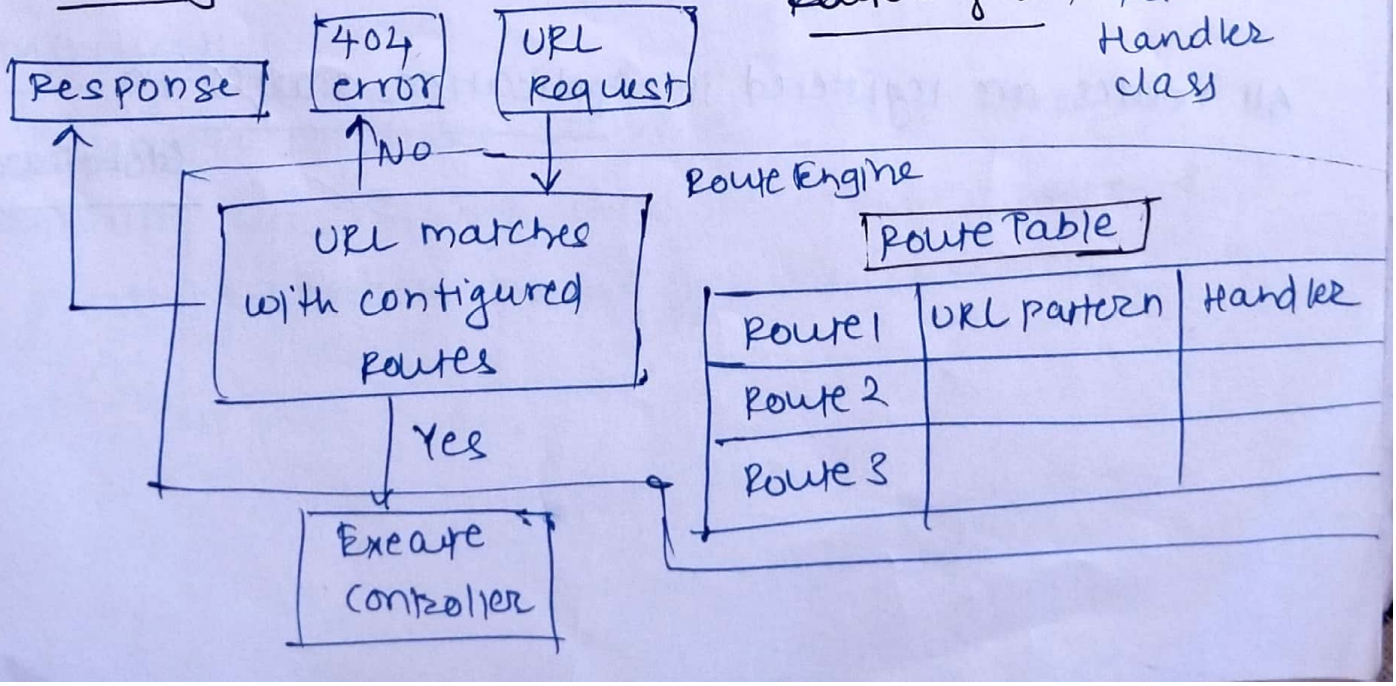
ADO.Net objects

① Connection
② Command
③ DataReader
④ Data Adapter → Dataset
⑤ Data Table

Route Table → Routes stored

Route Engine → determine Handler class

Routing



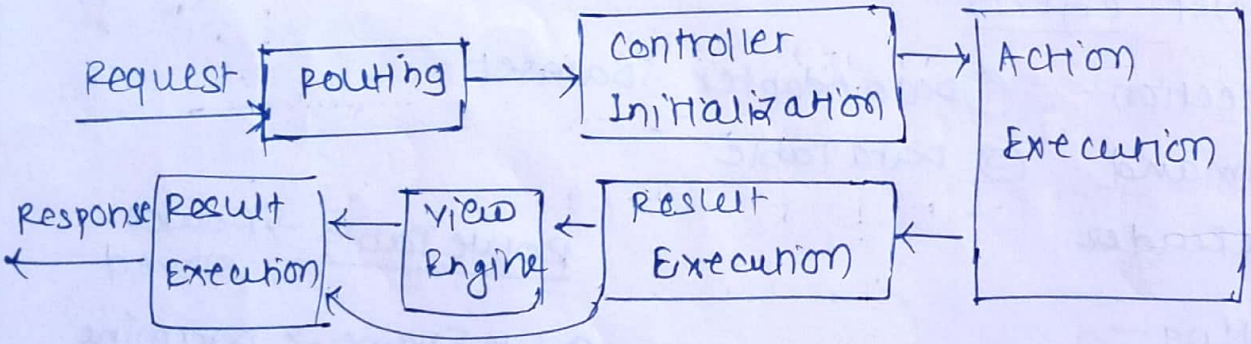
```
routes.MapRoute {
        name: "Default",
        url: "{controller}/{action}/{id}"
        defaults:
    new {controller = "Home", action = "Index", id = UrlParameter.
                                                  Optional }
);
```

RouteConfig.cs file

: Multiple Routes

```
routes.MapRoute (
        name: "student"
        url: "students/{id}"
    defaults: new {controller = "student",
                   action = "Index" }
);
```

Request life cycle



All routes are registered in Application_Start() in Global.asax

Session 17 : Layout, Bundle, Minification & mvc security.

Layout ⇒ Layout.cshtml By default

- Eliminates duplicate coding
- Enhances development speed.
- Easy maintenance

Shared / Layout.cshtml

RenderBody()	Renders portion of child view
RenderSection (string name)	Renders a content named section.

Bundling & minification :- Improve request load time

Bundling allows us to load the bunch of static files from server in single HTTP request.

Bundle Types

① script Bundle :- Responsible for Javascript minification of single or multiple script files.

② style Bundle :- Responsible for css minification of single or multiple css files.

minification :-

Technique optimizes script or css file size by removing unnecessary white space & comments & shorthing variable names into one characters.

⑥ Error Handling in MVC with log Entry :-

HandleError is default built in exception filter

Web.config file

MVC security

In visual studio, Authentication ⇓

1> None
2> Individual user account
3> MS identity platform
4> Windows.

CSRF – cross site Request Forgery Attacks

CSRF attacks are possible against web sites that use cookies

Anti-forgery Token / Request verification Token

Two Token

1 Token cookie

1 Token (inside form) Hidden form Fields as form data

HtmlHelper.AntiForgeryToken()

server

request include both Tokens

Yes → Access granted

No → NOt Access

OAuth- Open Authorization

Token based authorization & authentication.

[Authorize] ⟶ for security

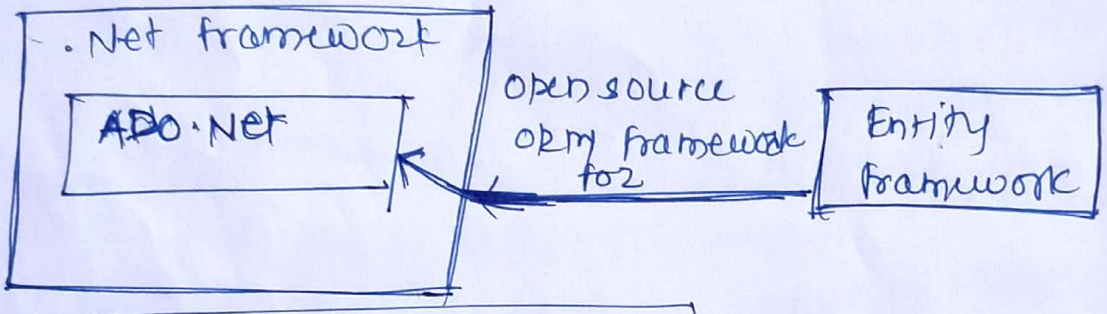
[AllowAnonyms] ⟶ didn't provide security.

session 18 : Entity Framework & ASP.NET mvc core

Intro. to Entity Framework

Entity Framework is an object Relational mapper (ORM)

mapping between [object in your s/w] To the → [columns & Tables of relational database]



ORM ⇒ [creating database connections, Executing commands, Taking query results] → convert into → [Application object]

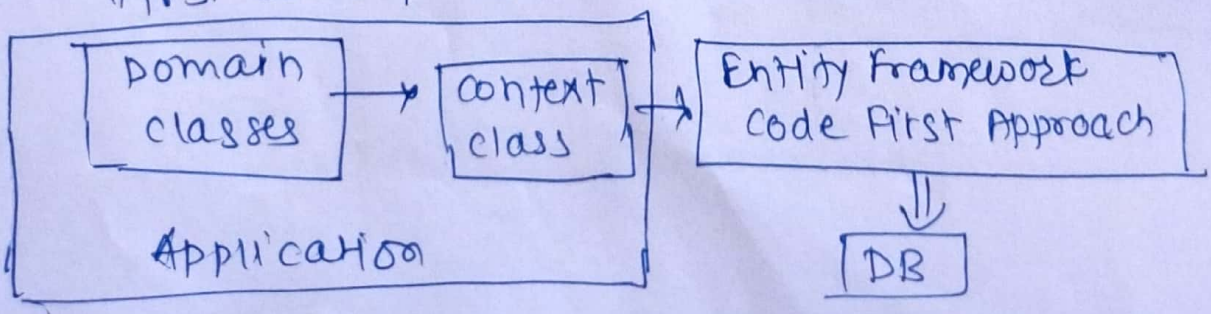
Automatically materializing

Approaches of Entity Framework

1. code First
2. database First
3. model First.

1. code First

First code & then connects to database



Database First

Database First then classes, properties, DBContext acc. to the database

Model First

Data Annotation Attributes

[Key], Required, MinLength, MaxLength, StringLength.

Table, Column, Index, ForeignKey, NotMapped.

↑ should have index

↑ {MappedBy in Java}

ASP.Net MVC	MVC Core
N	- No web.config & global.asax file - Don't need IIS for hosting - wwwroot folder for static files. - Inbuilt dependency Injection (DI) - Resves & client side dependency management of packages.

Session 19 : Localization in MVC & deploying MVC application

~~Localization~~ Internationalization

Process of enabling your code to run correctly all over the world

① Localization :-

② Globalization :-

System.Globalization namespace

CultureInfo class

↓

currentCulture property

⊙ Deploying MVC Application :-

→ Visual studio

solution explorer → publish

select IIS server & click next

Then select web deploy package & click next

Provide location of your project

then publish

IIS → Internet Information services.

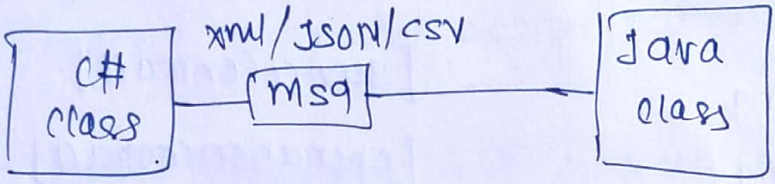
Session 20: Windows Communication Foundation (WCF)

Services :- logical encapsulation of self-contained business functionalities.

WCF :- Framework for building service oriented application. (SOA)

SOA- Service Oriented Architecture :-

Two non-compatible applications can communicate using a common language.



WCF application consists of

- WCF service
- WCF service host
- WCF service client

Fundamental Concepts of WCF

1. Message
2. End point
3. Binding
4. contracts
5. Hosting
6. WCF client
7. channel
8. SOAP. - simple object Access Protocol.
   XML document ⇒ Header + Body section

Contracts

1. service contract
2. Message contract
3. Data contract

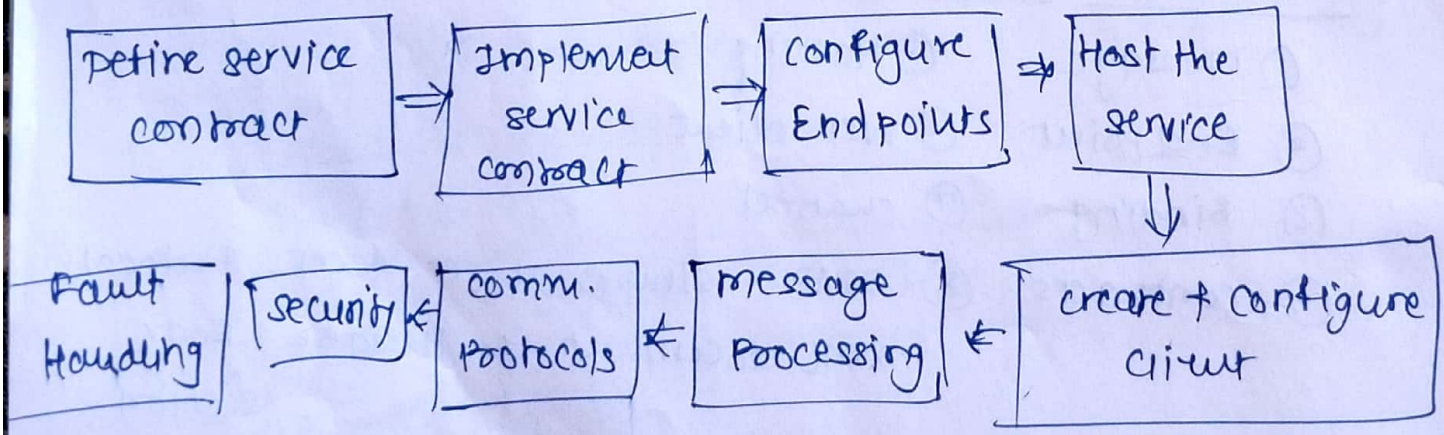
- Address (A) ⇒ where the service is
- Binding (B) ⇒ How to talk with service
- contract (C) ⇒ Tell us that what can service do for me

Types of Bindings

1. Basic Http Binding
2. Net Msmq Binding
3. Net Tcp Binding
4. Net Peer Tcp Binding
5. Ws Http Binding
6. Ws Dual Http Binding
7. Net Named Pipe Binding

[service Contract]

[operation contract]



Session 21 - Web API

API - Application programming interface

⇒ Set of rules that defines how applications or devices can connect to & communicate with each other.

⊙ Dependency Injection (DI) & Inversion of control (IOC)

control of event organizations to some other place ⇒ IOC

IOC - uses Bollywood design pattern

main class should not have concrete implementation of an aggregated class rather it should be depends on abstraction of class.

SOLID
↑
Dependency Inversion ⇒ { High level classes should not depend on low level but rather both should depend on abstractions.

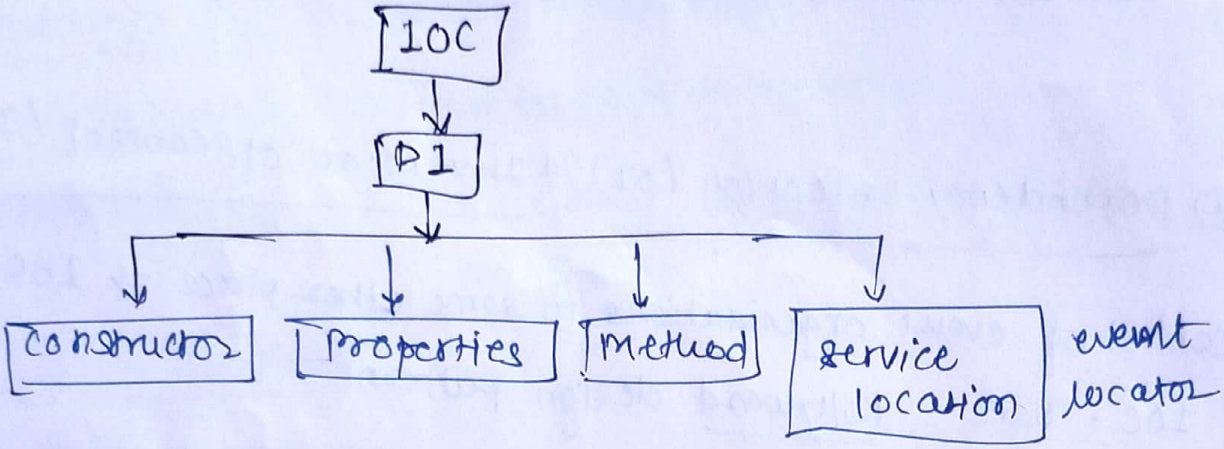
```
class college {
        TechEvent techEvents = new TechEvents();
    }

interface TechEvents {
        void organizingEvent()
    }
```

(X)

Dependency Injection

How one object knows about another abstracted dependent object



Injection via constructor

```
class College {
        private IEvent _events;
        public College(IEvent ie) {
                _events = ie;
        }
```

reference → _events; object → ie

object of concrete class (Event) passed to the constructor of dependent class.

Injection via property

```
class College {
        private IEvent _events;
        public IEvent MyEvent {
                set {
                        _events = value;
                }
        }
```

Injection via method

```
class College {
        private IEvent _events;
    public void getEvent(IEvent e) {
            _events = e;
    }
}
```

Injection via event locator

```
class college {
        private IEvent _events;
    EventLocator e1 = new EventLocator();
    public college(int index) {
            _events = e1.LocateEvent(index);
    }
}
```

How they are loosely coupled After DI :

In traditional must have to create instances of its dependencies directly using new keyword

→ ~~to~~ class College {

```
        private Event _event = new Event();
    }
```

With DI, no longer to creates its dependencies instead they injected from outside

```
Person prn = new Person(45, "chetan", "XYZ");
string fname = prn.GetFirstName();
string lname = prn.LastName;
Console.WriteLine(fname);
```

person is class

Reference Type :- class, interface, delegate, event.

Value Type :- primitive type.

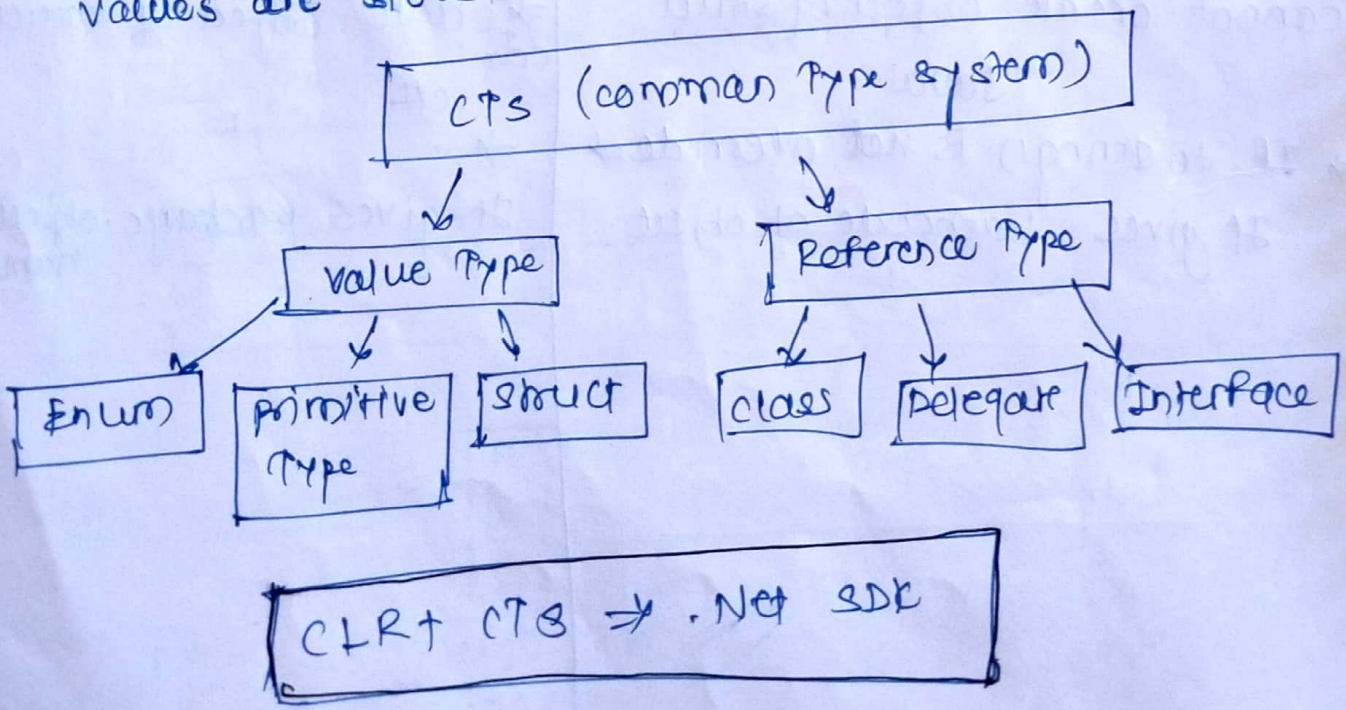
namespace Graphics :- create a package (define)

Object Type

values are stored on stack.

Reference Type

values are stored on heap



⊙ multi language support ⇒ 25 language ⇒ CTS

CTS ⇒ CLS + FCL

CTC ⇒ CLS + {base class libraries}

.net = CLR + CLS + BCL + multiple .net complaint lang. compilers.

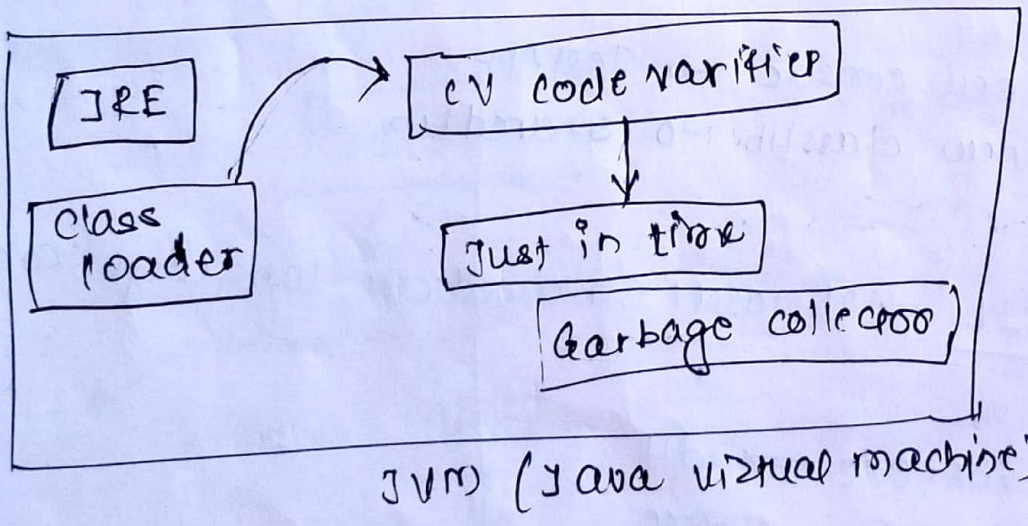
↑

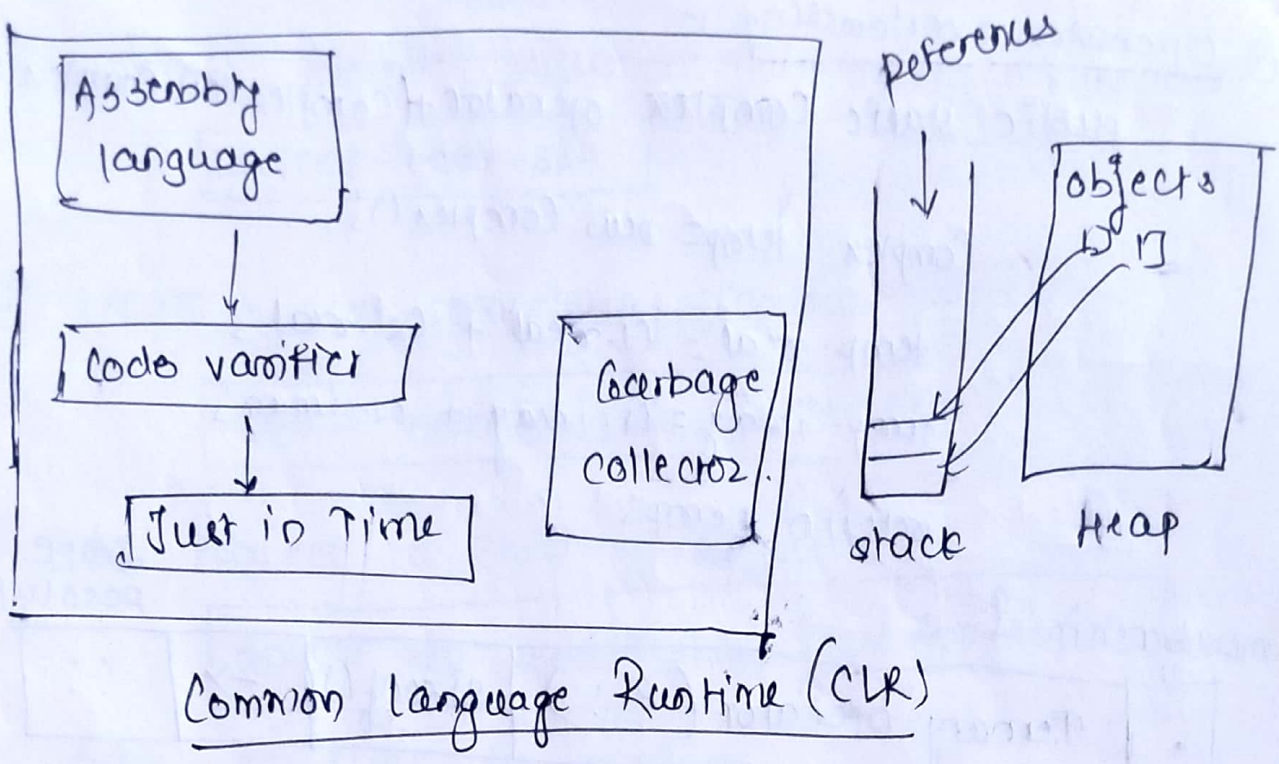
{base class libraries}

namespace :-

It is a collection of -

- class ⇒ account, savingAccount, salaryAcc
- interface ⇒ IBanking service
- delegate ⇒ Accountoperation
- event ⇒ Underbalance, overbalance
- enumeration ⇒
- structure (struct) ⇒ ministratement





Primary Thread PT

Garbage Collector HP - Harry Potter

Importance {C > PT}

constructor called by => primary Thread
Destructor called by => Garbage collector

* Destructor is always called by Garbage Collector (Daemon Thread) before object is going to be removed from heap.

```
List<Account> accounts = new List<Account>();
foreach(Account theAccount in accounts) {
    result = theAccount.GetBalance();
}
```

⑥ Operator overloading :-

```
public static Complex operator +(Complex c1, Complex c2){

    Complex temp= new Complex();
    temp.real = c1.real + c2.real;
    temp.imag = c1.imag + c2.imag;
    return temp;
```

membership ↓ | Scope resolution ↓

.	Ternary operator (? :)	sizeof()	->	::

can't be overloaded

⊙ Object oriented Programming in C# :-

.net solution :-

1. create a blank solution
2. create a new console application project.
3. Add cs files to project.
4. ~~Add cs files~~

OR

```
public int id;

public int Id {
    get { return this.id; }
    set { this.id=id; }
}
```

```
public string
fullname{
    get;
    set;
}
```

① create blank solution

```
dotnet new sln
```

sln ⇒ solution

② create a new console application project

```
dotnet new console -o TestApp
```

③ Add project to existing solution

```
dotnet sln add. TestApp |Test App.csproj
```

④ Add cs files to project

```
dotnet build
```

⑤ Build solution ⑥ Run project.

```
dotnet run
```

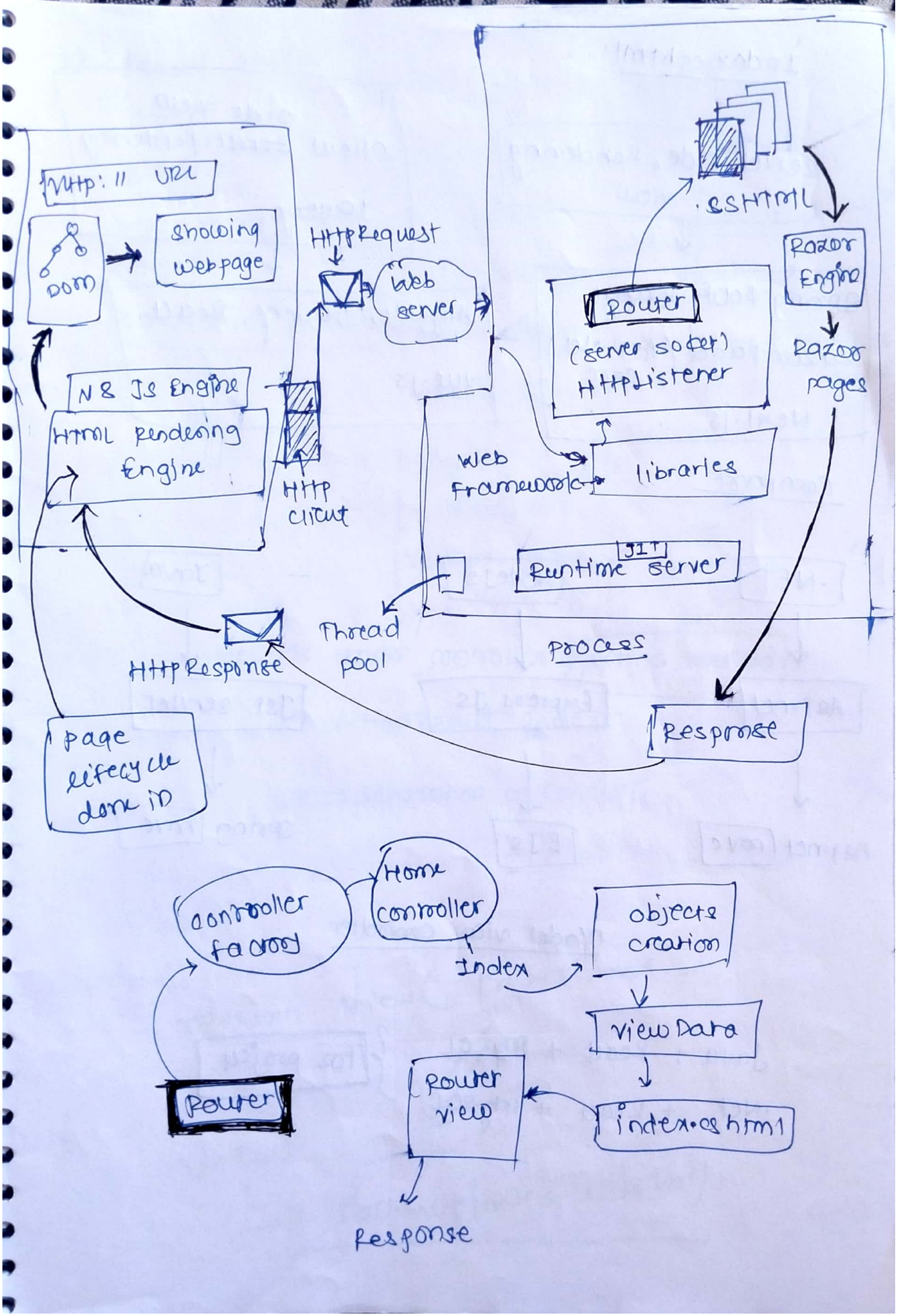
```
public override string(
    return base.ToString() + WorkingHours);
}
```

making method abstract ⇒ Enforce overriding in child class (Java)

make class abstract

making method virtual ⇒ may or may not override in child class

don't need make class abstract



Index.cshtml

Server side Rendering view

Spring BOOT, Django, Razor pages (Asp.Net core) Next.js

Client side view ~~Server~~ Rendering ~~jQuery~~,

Angular, jQuery, React, vue.js

Compare

.NET → Asp.net → Asp.net mvc

NodeJs → Express JS → EJS

Java → JSP servlet → Spring mvc

Model view controller

Java + React + mySQL

.Net + React + mySQL

} For project

Date : 23/12/2023

dotnet new console
dotnet new classlib
dotnet new webapi
dotnet new winforms
dotnet add reference
dotnet build
dotnet run
dotnet watch
dotnet new mvc
dotnet new webapp
dotnet new page
dotnet new sln
dotnet sln add.

data members :-
- member functions
- member variable
- events
  - paint
  - MouseUp
  - MouseDown
  - MouseMove
  - KeyDown
  - KeyUp

⊙ Attach EventHandler with event :-

this.paint += new PaintEventHandler(OnPaint)

this.MouseDown += new MouseEventHandler(onMouseDown)

this.keyDown += new KeyEventHandler(onKeyUp onKeyDown)

this.KeyUp +=

Built / default Interface

IDisposable, IEnumerator.

⊙ Garbage collector function :-

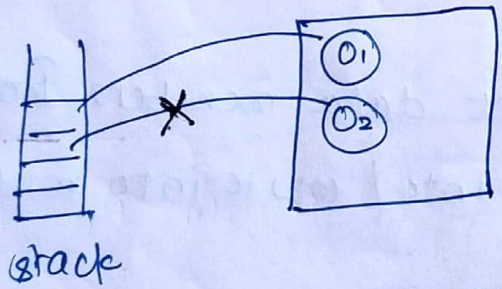
1. GC.SuppressFinalize(this)
2. GC.WaitForPendingFinalizers()
3. GC.Collect()

Garbage collector call finalize() function before object gets destroyed from heap.

orphanized object :- Efficient memory management

GC works of mark & sweep algorithm.

Deterministic finalization



stack ⇒ object reference
Heap ⇒ object stored

If object's reference removed from stack the object becomes — orphaned object

O2

Resource Allocation done ✓ ⇒ Release is not done ✗

GC.Collect() ⇒ Forcefully calling GC

If unreferenced object found

⇓

mark it

⇓

Remove it

GC.WaitForPendingFinalizers() ⇒

Day 4 | Dotnet | C# keywords

Dispose

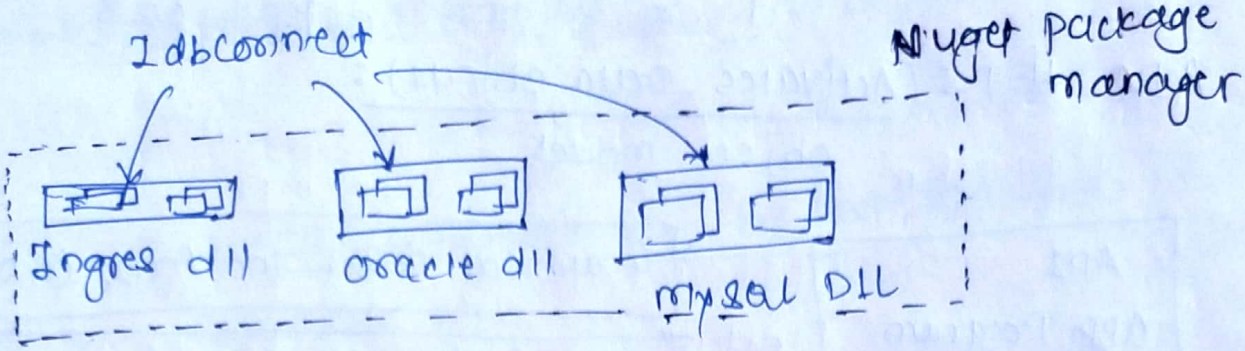
⇓

(done by primary thread)

finalize & destructor

⇓

(called by GC)



sqlconnection	oracleconnection	mysqlconnection
↓	↓	↓
open	open	open
close	close	close.

- ~~dotnet new sln dbsolution~~
- dotnet new console app dbApp
- download mysql connector (dev.mysql.com)
- dotnet add package ~~mysql~~ (in dbApp)
  MySQL.Data

~~working~~ using mysql.data.myclient.
using mysql.data.mysqlclient

---

step 1) define connection string
step 2) create connection object
step 3) set connection string to connection object
step 4) defined sql query
step 5) create command object
step 6) open connection
step 7) Execute command
step 8) retrieve result into DataReader
step 9) Iterate data from reader object
step 10) show data
step 11) close connection