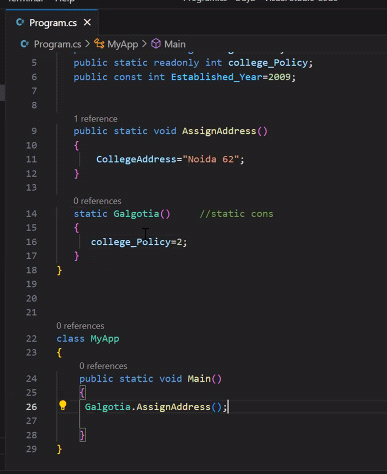
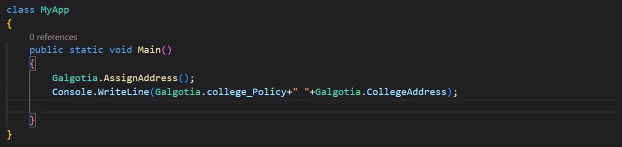
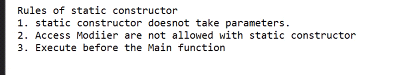
**DAY-2**

Static constructor is started before main function





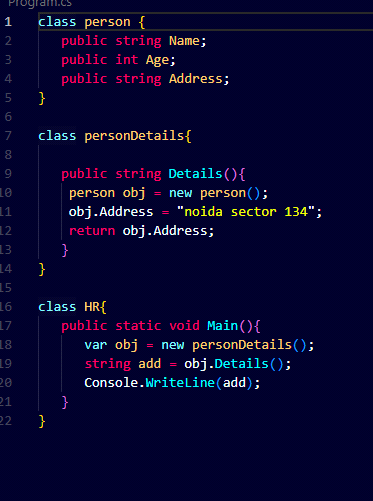
Static constructor run on priority before main.



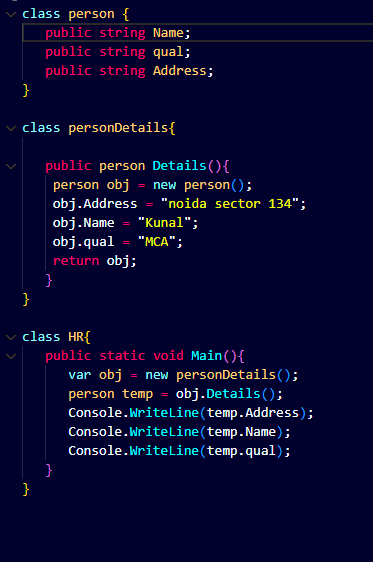
-------------------------------------------------------------------------------------------------------------------

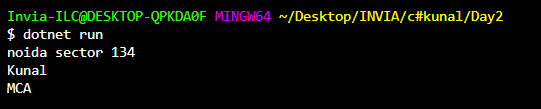
Returning multiple values

returning one value through function



Returning class using object in functions





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

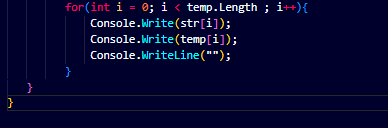
**Q. How function can return values(Array, collection, generaics)**

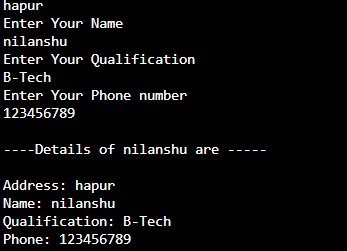
recommended: use generics

**Namespace** is a collections of classes similiar to package in JAVA.

* system is the root namespace in dotnet.
* we can make nested namespace in dotnet.
* if we are writing **system.collection** then we can only access classes of collection and not that of the system.
* **System** namespace is included by default in our program.cs file.

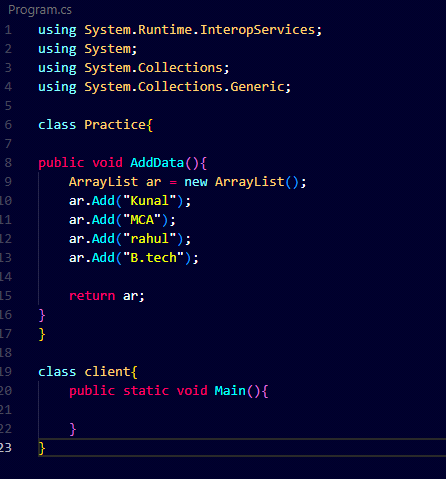






----------------------- Collection and Generics

Creating arrayList in c#



* Prefer **Foreach** as its better than **FOR** Loop.
* use **var** instead of primitive data type as they are better for memory managment when working on large scale.
* reference type(class,object) are stored in **HEAP**
* primitive type(string, double, int) are stored in **Stack**

All the data stored in Arraylist is stored in HEAP as a Object and it can't be called as **Console.WriteLine(arr[0])** as it should be converted to its primitive type first and move the item from Heap to Stack.

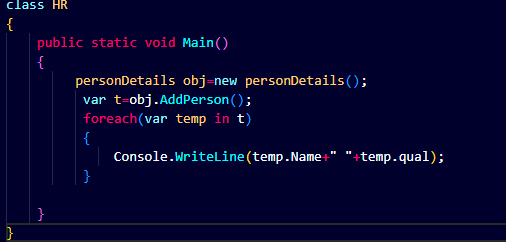
Collection promote boxing and unboxing.

* major drawback of collection is that it supports boxing and unboxing.
* Boxing --> The process of converting a Value Type variable (char, int etc.) to a Reference Type variable (object) is called Boxing. from Stack to Heap.
* Reverse of boxing is unboxing.
* Collection is obselete and now Generic is used as there are performace issue due to boxing and unboxing in collection.
* **string age = Convert.ToString(temp[4]);** here we are converting from heap to stack(unboxing).
* Convert is the default class used to convert data from refernce type to primitive type.

----List

* class of Generic similiar to array list but without boxing and unboxing.
* **List<>** here "<>" is called generics.
* **List<person> objList = new List<person>();**
* **objList.Add(new person{ }) ;**



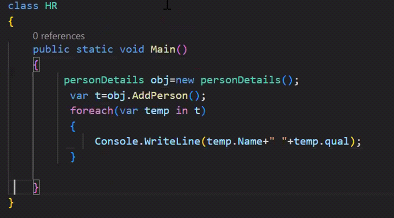


DAY 3

class Person is called is the **requirement** as it store the data

Class personDetails is called as **Buisness logic layer or service logic layer** as it contains all the logic for the data fetched from the Person Class.

The class HR class is called the **UI layer** as it contains the user interface for the user.



* Main is single threaded function.
* One class in one File.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Program.cs

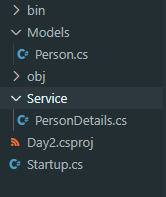
1. Person  
   Create folder ... Model(class person.cs) ... Namespace --> Model
2. PersonDetails  
   Create folder ... Service(class PersonDetails.cs) ... Namespace -->Service
3. HR(Starter)

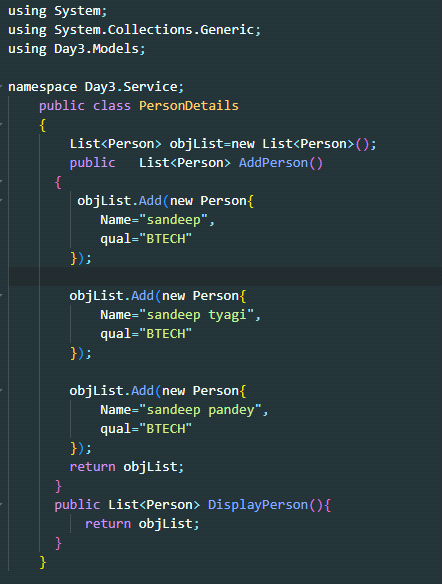
Create folder ... StartUp(HR.cs) ... Namespace --> StartUp

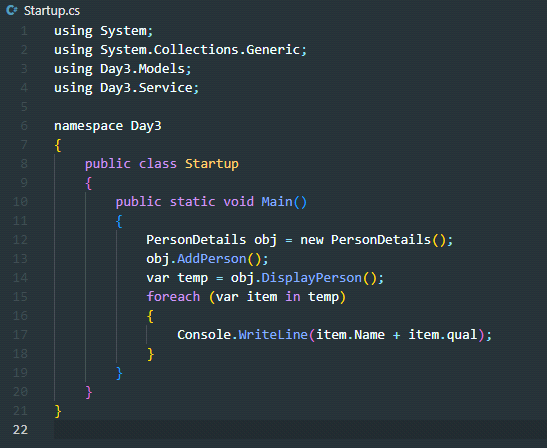
Currrent C# version is 9.0 and i it we dont have to enclose the data in a namespace using {}

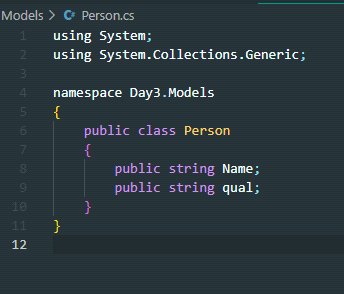
For display or formatting or formatting never use **Concatination** and instead use **Interpolation**

**------------------------------------------------------------------------------------------------**

** File Structure**







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

Solution --> Modules + Console Application

Modules can take reference of each other.

To create a new Solution Directory.

**dotnet new sln --name PersonRepositoy**

To create a new Library.

**dotnet new classlib --name Service**

similiar for Models.

To add library to Solution

**dotnet sln add Models\Models.csproj**

Similarly add service library

**dotnet sln add Service\Service.csproj**

**\*\* benefit of adding:- If we compile the program then all the added application got compiled automatically \*\***

Now after making the backend we have to create our frontend application which in this case is our console application.

**dotnet new console --name UI**

now add the UI to the solution

**dotnet sln add UI\UI.csproj**

Now we have to Take the reference of one library to another library

Taking refernce of Model library in Service

**dotnet add Service\Service.csproj reference Models\Models.csproj**

Adding Model and Service reference in UI(console Application)

**dotnet add UI\UI.csproj reference Models\Models.csproj**

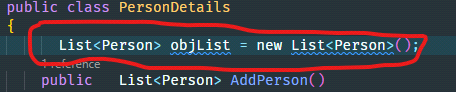
**dotnet add UI\UI.csproj reference Service\Service.csproj**

Now run **Dotnet Build** to build the Application.

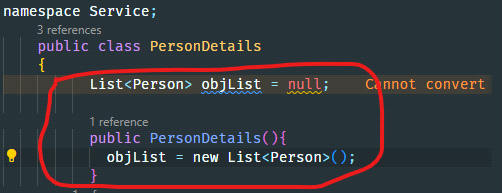
**Dotnet restore** to restore any missing package.

DAY 4

**\*\*Declare variables but Don't allocate memory at the start of the scope of the memory\*\***



Instead allocate memory using constructor of that class.



**Q. Why we need runtime polymorphism?**

Run time polymorphism is also known as **Dynamic Method Dispatch** as the method functionality is decided dynamically at run time based on the object.

\*\* Keep updating your tech stack \*\*

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

**Encapsulation and Abstraction**

**\*\* Segregate declaration from implementation in the Code \*\***

if there is no segeregation between declaration and implementation then there is no encapsulation and if no encapsuation then no abstraction.

I have taken the class PersonDetails.cs defining all the functions, it means **Declaration and Implementation are tightly Coupled**.Oveall There is no **Encapsulation** and **Abstraction**. they shoud be loosly coupled by using interface.

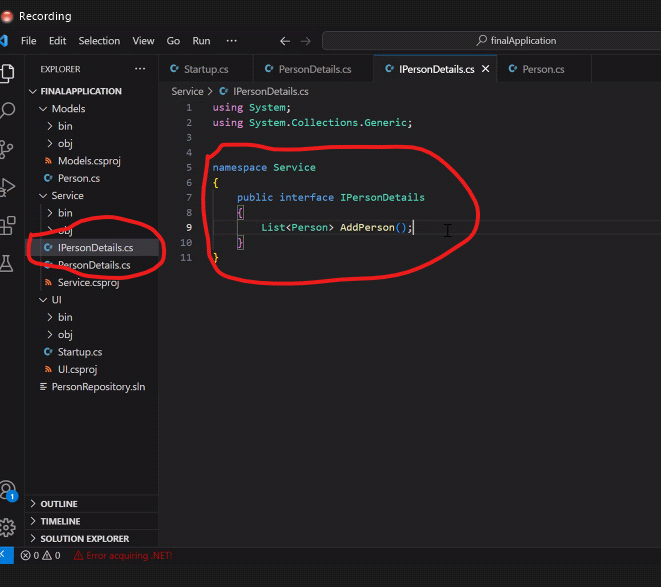
-------------------**Interface in C#**------------------------------------

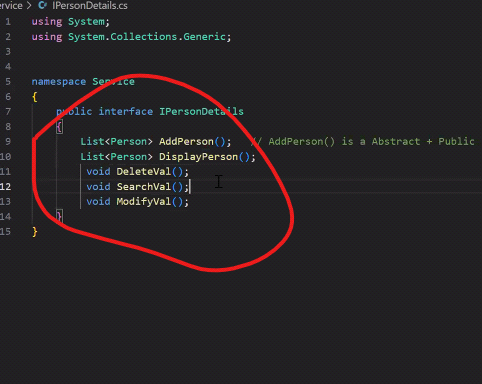
Interface is a placeholder where we can have only function declaration.

* Interface is a placeholder.
* interface is the keyword.
* interface can't have any kind variable but class can have.
* Interface can have only function declaration.
* function in interface are by default abstract(function which dont have any body are called abstract) and public.

The interface file must start with a Capital "I".In our case name is

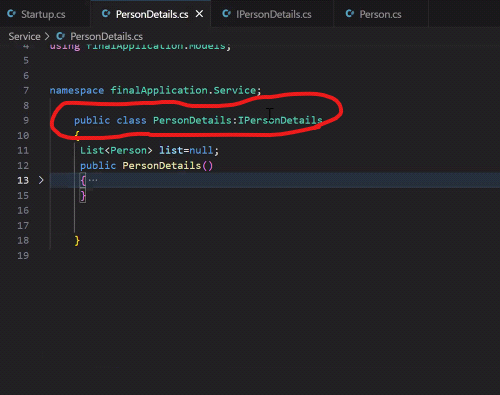
**IPersonalDetails.cs**

****

****

In class we can define and declare the function but in Interface we can can only define the function but can't declare.

to define Interface in our class we have to write :**IPersonDetails**(INTERFACE NAME)



If we remove the function added in the interface from the class then it will show us error(eg. to open a new account in bank we need 4 document and account will not open with less than 4 documents like this system will not allow us to run the program if the programs in interface and class doesen't match.

**Interface file should be made in a seperate directory"Functionality"**

Service

|\_PersonDetails

Create new Library Functionality for Interface

**Terminal:- dotnet new classlib --name Functionality**

Adding functionality to the solution

**Terminal:- dotnet sln add Functionality/Functionality.csproj**

Adding functionality refernce to service

**Terminal:- dotnet add Service/Service.csproj reference Functionality/Functionality.csproj**

Adding model reference to functionality

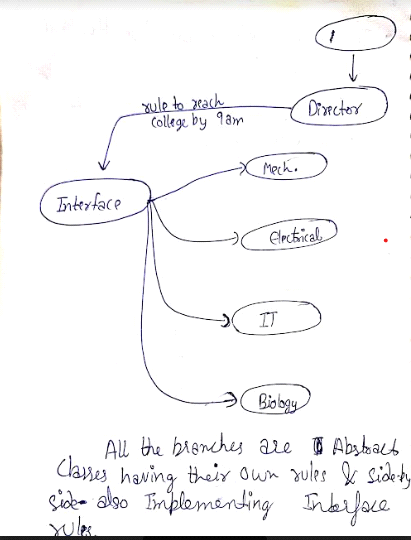
**Terminal:- dotnet add Functionality/Functionality.csproj reference Models/Models.csproj**

now move the Interface IpersonDetails.cs to Functionality library and delete default class1.cs

**Day-5**

**------- Abstract Class**

* if a class is marked as a Abstract then it itself behave as a interface for the derived(Child) Class.
* Absract can have Abstract as well as non-abstract methods but an interface can only have Abstract class.
* By interface we can acheive **Diversity in inquaity** means interface functions are implemented by **Dissimiliar type** Functions.
* By Abstract we can implement only functions with **Similiar Type.**
* If a class is marked as abstract and it is must to overide it in the **derived class**.(**runtime polymorphism).**
* **Eg.** Class **Traffic\_speed** is a a interface which need to be followed by everyone irrrespective of the type, make, owner of the transport but the class **Two\_wheeler, Three\_Wheeler,Four\_Wheeler** are all abstract classes following the interface by also follow their individual class rules.
* Interface can bind multiple classes with dissimiliar Type.



**------- Dynamic V/S Var**

In **Dynamic** we need to Initialize and dont need to declare the value there and can be done later on.

In **Var** we need to Initiate and Declare the Value together and it fixed throughout the Program.

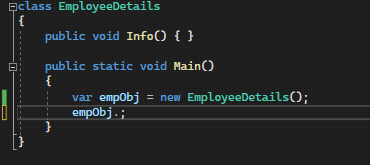
**Explation Evaluation -**  means where our output is being evaluated and we don't know Whether the output is Int, Double, Float or other so **here we use Dynamic Type.**

**Dynamic uses runtime Memory Allocation**  and **Var uses Compile Time Memory Allocation.**

Dynamic is introducted in c# 4.0 and Var is defined in C# 3.0 before c# 4.0 C# is said to be statically typed language but after Introduction of Dynamic it become dynamically typed language.

If you use Dynamic for **alaising** then after writing obj **dot**  we cant see the parent methods but same not true for Var.

This process of not showing anything after using **dot** in the empObj in case of using dynamic data type(empObj in this case) is called as **Dynamic Behaviour and Late Binding**.



* **Case study-1:**

**\* Var ---** Early Binding

**\* Dynamic ---** Late Binding

* **Case study-2:**

**\* Var ---** Function parameter in C# does not allow "VAR keyword"

**\* Dynamic ---**Function parameter in C# allow "Dynamic keyword"

* **Case study-3:**

**\* Var ---**  can be used for function return type

**\* Dynamic ---**can't be used for function return type

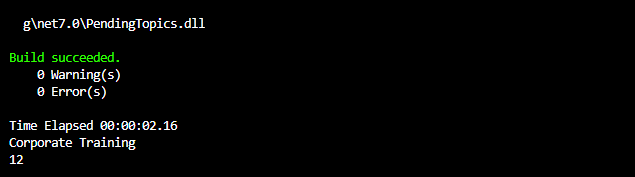
* **Case study-4:**

**\* Var ---**  can't be used at class levl, function parameter, return type

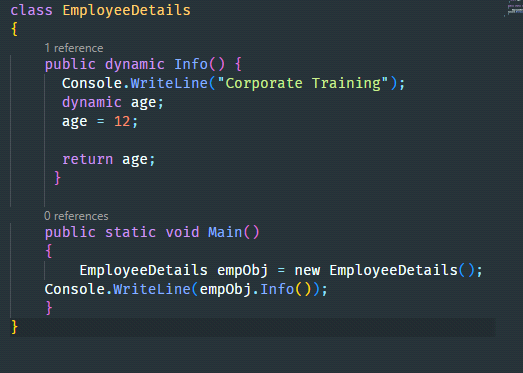
**\* Dynamic ---**can be used

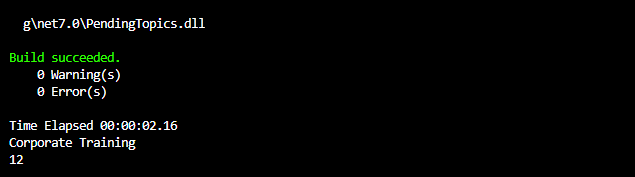
* using **VAR**





* By using **DYNAMIC**





**Things to kept in mind after creating a class**

* Object is a Base Class of the Employee.
* Scope of classs is Internal.
* Every Variable, Function declared in class is by default **Private**.
* Object Class Provides low level Services to Derived class.
* Garbage Collection, Cloning are the major feature of Object Class as a low level Service.

------------------ Day-6 ------------------------

* **Inheritance:** Inheritance is a fundamental concept in object-oriented programming that allows us to define a new class based on an existing class. The new class inherits the properties and methods of the existing class and can also add new properties and methods of its own. Inheritance promotes code reuse, simplifies code maintenance, and improves code organization.

There should be a logical relationship between miultiple classes. If there is a relation then  **only we can use inheritance.**

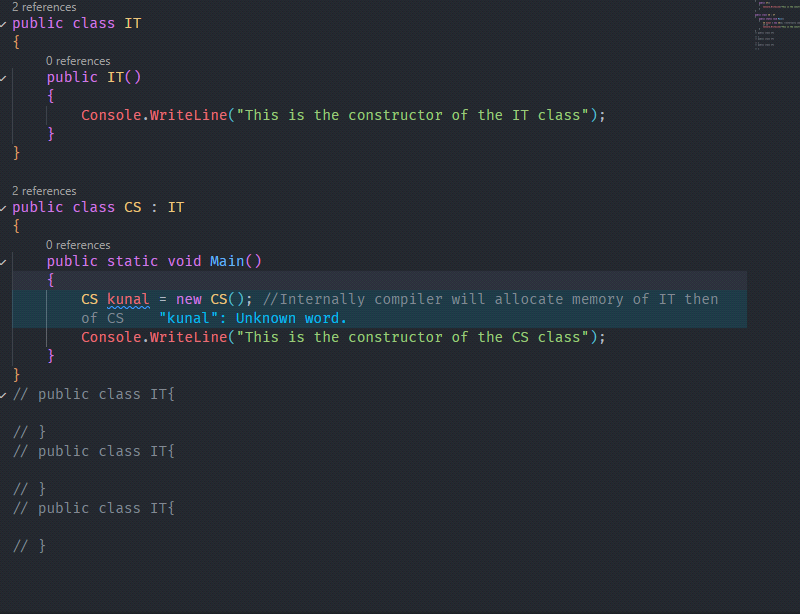
1. Every derieved class is the **extended Version of the base Class.**
2. Every base class should have larger scope than the Derived class. eg. class **CollegeDepartment** can't be the base class for the Derived class **College.** but true if we interchange their positions.
3. \*\* When we make Object of the Derived class then internally the system make the object of the base class first and then of derieved class this base class object is called as **Virtual Object.**
4. **C#/Java** Doesen't Support **Multiple Inheritance.**
5. Object of the base class is embedded within Derieved Class.
6. If the base class is also a inherited Derived class for another base class then the Root base class object is alloated memory first then its next derieved class and at last the node or the bottom derieved class.

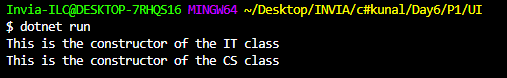
eg. If **GrandFather** is the base class for the derieved class for **Father** class which is inherited by **Son** class then when we make object of the Son class then internally first the object of the Grandson class is allocated memory then of Father and then at last of son class**.**

**Q. What is the role of constructor in Inheritance ?**

**How Can we check whether base class object is created first then object of the derieved class ?**

**Ans.** First make the constructor of the base class and then run the program with the below code.

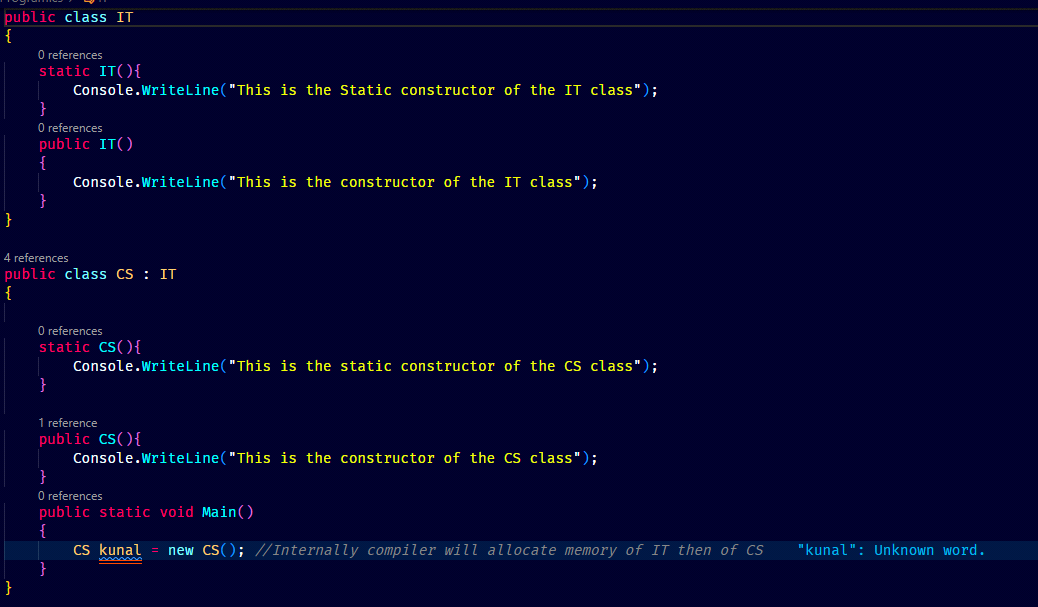


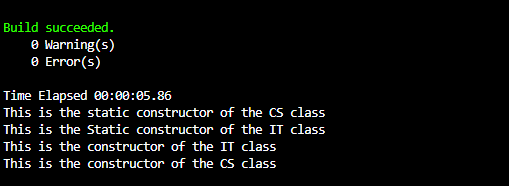


As we can see from the above output that the constuctor of the base class is created first and the ConsoleWriteLine is called in Base Class then the Console.WriteLine of the Derieved class is called,**This Proves that the object of the base class is allocated memory first then of the derieved class in Heap.**

**\*Constructor can never be Inherited\***

**Q. What is the Role of Static Constuctor in inheritance ?**





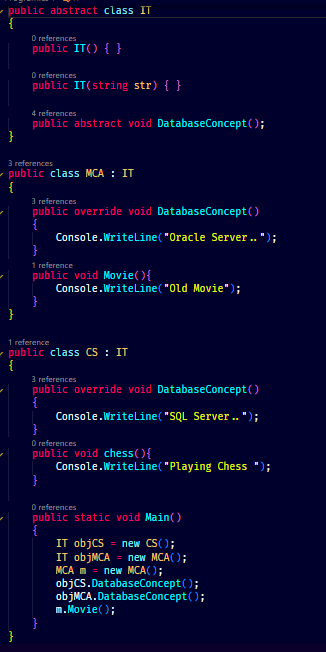
First the static constructor of the derived class is called as the obj is of the Derived Class then the static constructor of the base class is called(**Static constructor start before main**) and after that object of the base class is allcoated memory which call the default constructor of the base class and then atlast the default constructor of the rived class is called.

**\*\*Class is By default INTERNAL type and we can add PUBLIC but we can't use PRIVATE and PROTECTED to declare class as it will give compile time error\*\***

* **Practical Implementation of abstract class**

1. \*\*It is must to **Overide the Abstract method in the derieved Class**. If a class is marked as abstact then it is intended to be extended by the derieved class but not initiated.
2. When we make a derived class object with respect to the Base Class this is known as Runtime Polymorphism.
3. Requirement analysis of any application and for this . This process is called as Abstraction.**To provide Abstraction to the end user we need Private Keyword**.
4. The methods which are not essential to the user are need to be hided to the user by makng the not essential class private and this process is called as **Abstraction.**
5. If there is no abstraction then there is no Encapsulation.

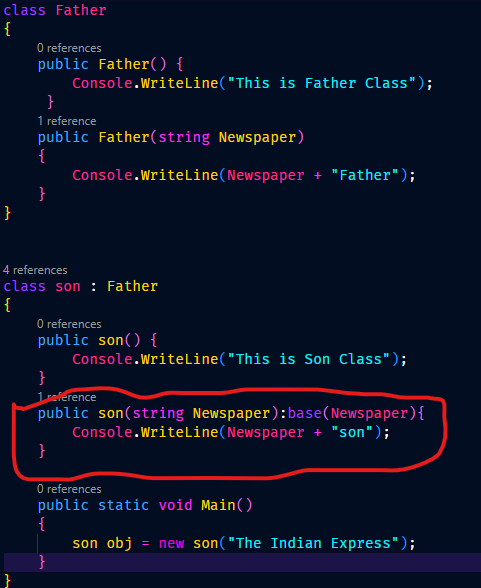
**Practical Implementation of Runtime Polymorphism**

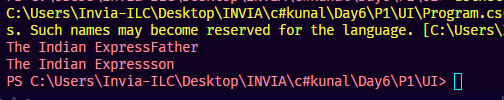
****

In the above code we can see that **IT objCS = new CS();** this line create a object ‘objCS’ of the IT Class(Parent) and allocte memory in the child class CS.It can only access the parent’s override methods like **DatabaseConcept()** in this case. This Promotes Abstraction as by writing objCS. We can only see the DatabaseConcept method and not the **Chess()** Method.

* **Constructor Chaining**

1. **Constructor chaining** is the process to call the parameterized Constructor of the Base Class with the help of the derieved class otherwise it will always call the default constructor of both classes.
2. As we can see from the below code that the **Function Public Son** passes the value of the String **Newspaper** to the base class parameterized constructor and Base class constructor is executed first and then the child class.

****

****

**--------------------------------- Day-7 ----------------------------**

Milestones for the next few Classes :-

1. **Properties**
2. **Generics**
3. **Indexer**
4. **Delegate**
5. **Types of Delegates**

* **Properties**
  + - **Limitation of Properties**
    - **Indexer**
    - **Generics(90% used in projects)\*\***

List

Dictionary

HashSet

Hashtable

* If there is no data Hiding, No Information hiding then there is no Encapsulation also called as **Wrapper**.
* To Achieve the encapsulation, **Function 🡪 Properties**
* When we want to access private variable of a class from other classes then we make the object of the class and we have to make a function(public) and we can use **Getter** and **Setter** to alter the value of the private variable.This is called as **Encapsulation**.
* If the variable is private then it **can’t be accessed using object Dot** so it promotes encapsulation and we need to make a wrapper function to use Get and Set methods.
* If we use **INTERNAL** as variable type then the scope of the variable is like is **PUBLIC** for the classes in the same directory or with the same solution and **PRIVATE** for other application classes.
* If we use **PROTECTED** The it act as **PRIVATE** for all the classes, Only the inherited classes can access the parent class members and variables.

**ORDER OF ACCESSIBILITY**

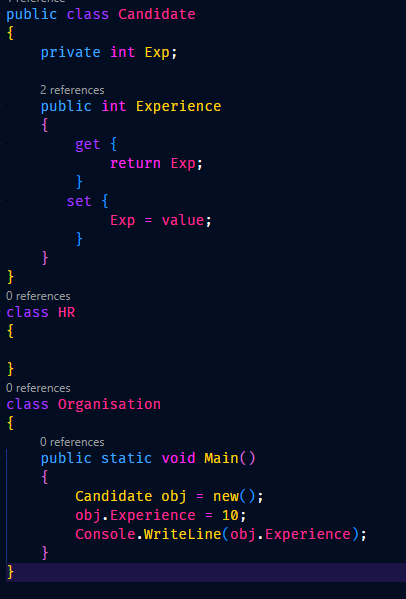
**|\_PUBLIC**

**|\_ INTERNAL**

**|\_ PROTECTED**

**|\_PRIVATE**

* **Wrapper Can be Made in two Ways:**
  + **By using Function**
  + **By using Properties**
* **Properties**
  + **VALUE** is the temporary variable in the setter method which temporarily store the value which is send thorough the obje ct dot notation.
  + **Properties** is used to get and set private variable values of the class.

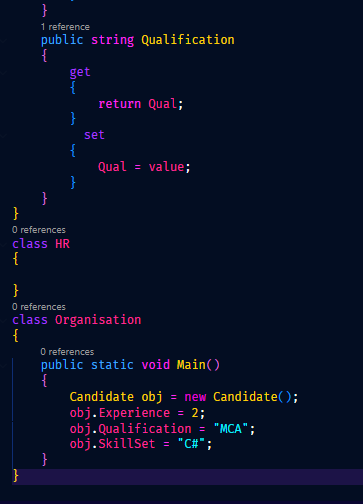


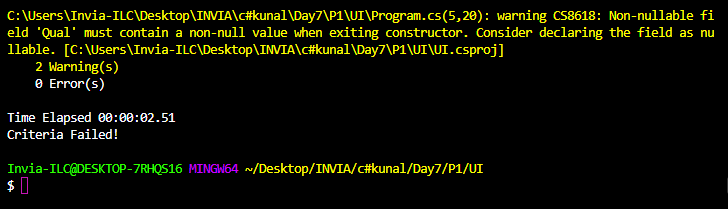
output.png

(18th Aug 2023) Friday presentation Topic:- Static constructor, constructor chaining, properties, Indexer, Access Modifier.

* \*\*Properties can bind only single value. Means If I have 3 PRIVATE Variable.
* By properties we can apply **validation** and by the function we can **perform a Task.**
* **Properties** are meant for **Validating the input data.**
* Properties v/s Functions
  + - Properties and functions both are meant for achieving the Encapsulation.
    - When we develop the Application we need to validate the data then call the function to submit the data to the database.

****

****

****

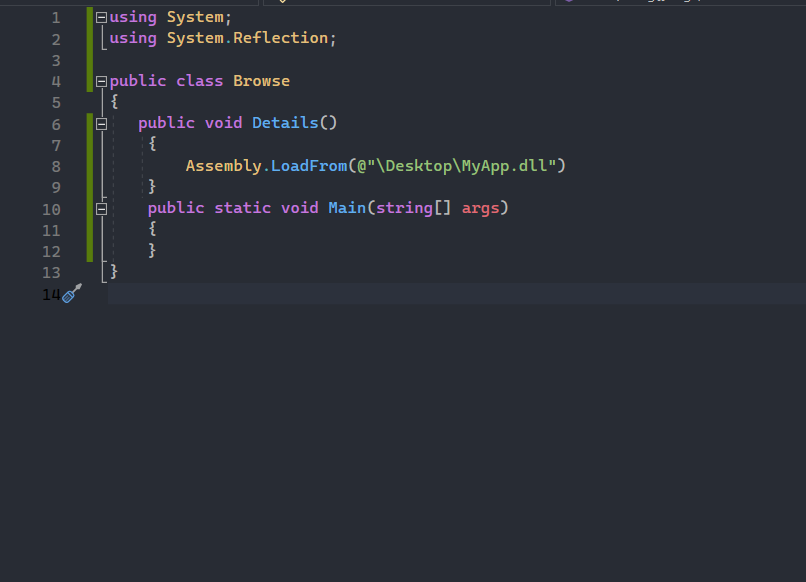
**INDEXER**

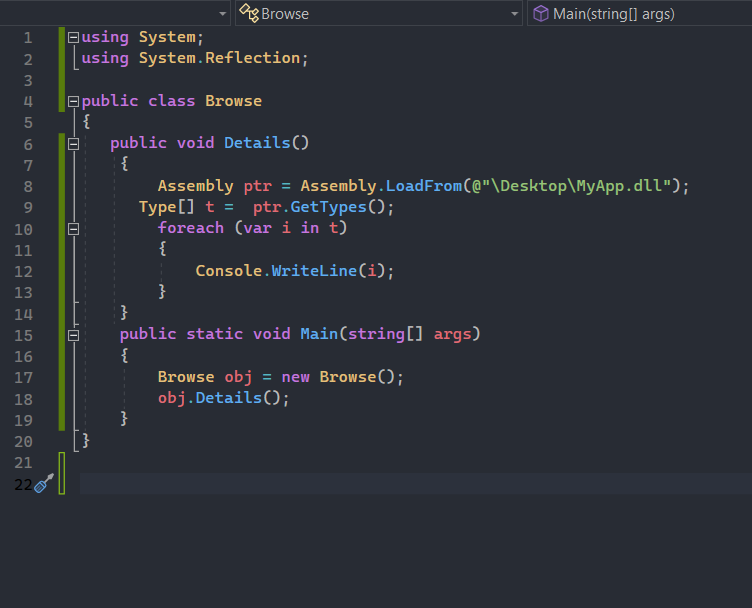
* **­­­­­Indexer** is a **array of properties**.
* Syntax Wise Complex.
* We can bind multiple private Variables with the help of indexer.
* Rules
  + Remove all the variables.
  + Declare variable dynamic array.
  + Remove all the Properties.

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

**Day-14**

Code for The .dll File to be used as class library



 In the Above code we are using that MyApp.dll file to be imported using **Assembly.LoadFrom(location).**