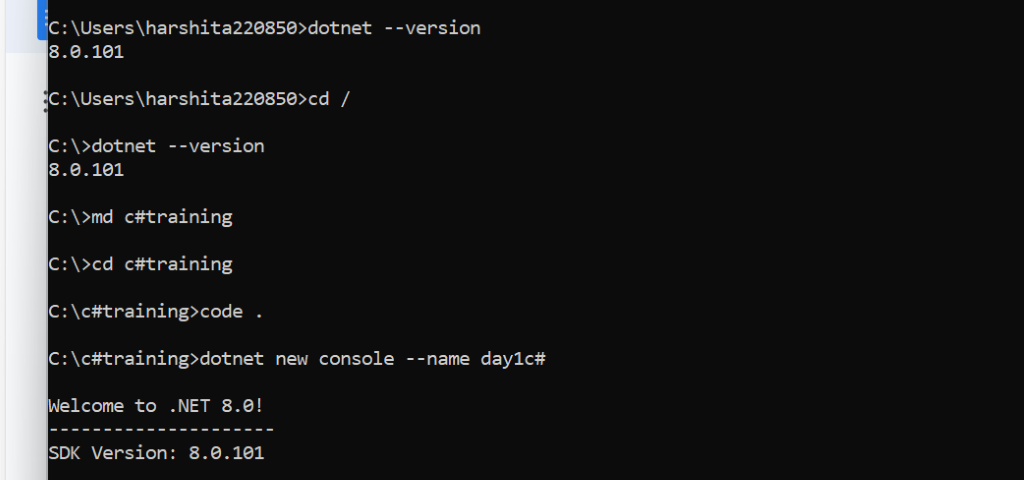
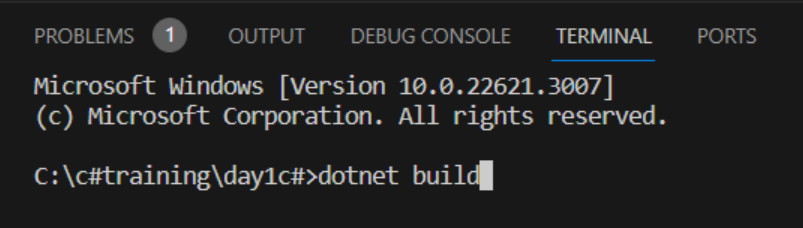
**Command prompt commands -**



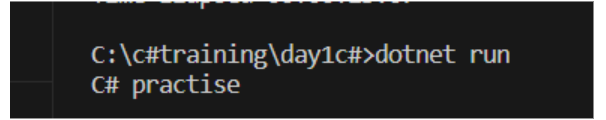
* **dotnet new console --name day1c#**

This is the command to create console application in c#

And application name is “day1c#”



* **dotnet build -** This is the command to compile the program



* This is the command to run the program

**Agenda day 1**

1. We have configured c# in vscode environment to compile and run the program
2. Types of variables in c#
3. Types of functions
4. Types of constructors

1. What is **class**?
   1. Class allows to create **user defined data type**
   2. within the class we can have **multiple kinds of variable**
      1. instance variable - **done - by two ways, constructor and functions**
      2. Instance read only variable - **(one time initialization) - done**
         * Instance read only variable is a one-time activity to set a value
         * We can set a value by constructor or parameterized constructor only, means no function or no other way except constructor or parameterized constructor
         * Example - “personid” is the best nominee to declare an instance read only
      3. static variable
         * It is said to be class level variable that's why we can access by class name
         * **\*\***If any information is common to all employees in an organization, then it is the best nominee for static variable
         * Example - “office address, office phone no.” best example of static variable
         * There are several ways we can set a value to static variable
           1. by static constructor
           2. by static function
           3. by class name.
      4. Static read only variable
      5. Constant variable
   3. **object** is a **pointer to class** known as reference pointer
   4. **class cannot be private or protected** otherwise it will produce compile time error
   5. by default, **scope of the class** is “internal”
   6. every member of the class is said to be “private” by default
   7. compiler does not allow to compile the program without “Main function”
   8. Can explain architecture of class and object \*\*\*
      1. **Stack and heap** play a major role while developing the programs in java and c#
      2. execution of function takes place in stack memory
      3. stack memory consists of 3 major things
         1. Function
         2. Local variables
         3. Object of a class (reference pointer)
      4. heap is said to be as storage memory means memory of class is allocated in heap memory
      5. **instance variable, instance read only variable** is a part of **heap memory**
      6. **static variable, statics read only variable, constant variable** are said to be **global variable** so we can **access by class name**, it is available in upper section of the heap memory
   9. Can you make a diagram to visualize the architecture of class and object in heap, stack memory

Example -

class person

{

int personid; // *it's a member/instance variable*

int personage;

public static void Main ()

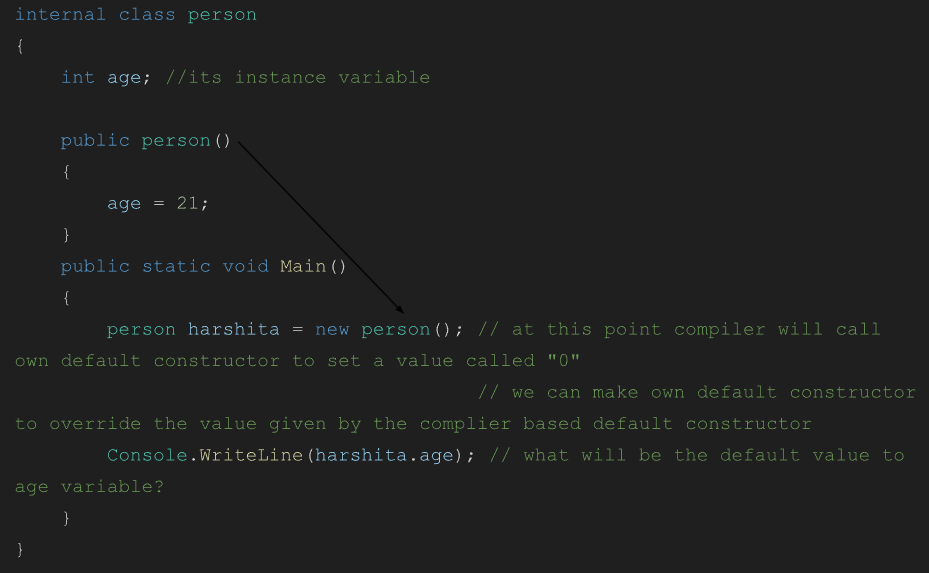
{

person harshita = new person ();

}

}

**Remark -** Every compiler calls its own default constructor to set a default value “0”.



Frontend class table

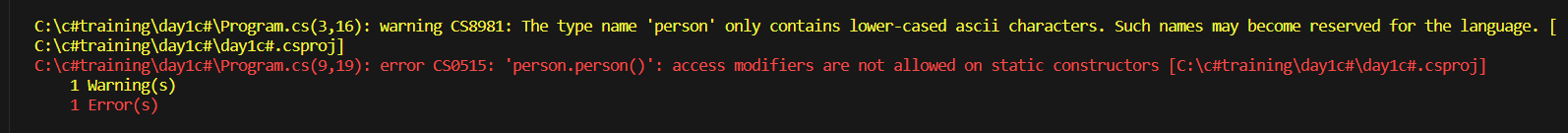
(how to design class)

(No matter in which language)

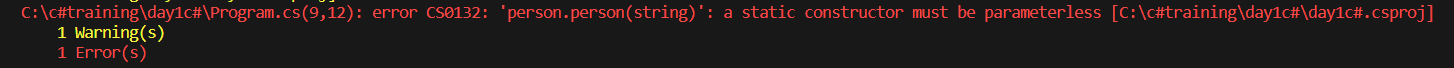
Question -> **Difference between static constructor and default constructor.**

1. Static constructor cannot have any access modifier (no public, no private etc.) -**done**
2. We cannot pass parameter to static constructor **-done**
3. It means, we cannot overload the static constructor -**done**
4. Static constructor is so special it **runs before the main function\*\* -done**
5. **Area of use** - static variable is the best nominee to initialize by static constructor because static variable and static constructor both are independent to an object -**done**
6. Default constructor can be overloaded -**done**
7. We can initialize instance variable or instance read only variable by default constructor or parameterized constructor’ -**done**
8. Default constructor can have a scope (private or public) -**done**
9. An object is needed with “new” operator to call default constructor or parameterized constructor –**done**

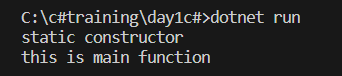
After implementation of access modifier on static constructor we get this message



After implementation of parameter in static constructor



After implementation to check which function runs first-



**CONCLUSION - DAY 1 TOPICS COVERED**

1. Class
2. Object
3. Memory architecture by making diagram
4. Stack and heap
5. What is location of an object - because object is a reference pointer and persist in stack memory
6. “new” operator means allocate the memory of class in heap, then compiler will supply own default constructor
7. We have seen how to design the class for full stack development because frontend is giving data to class and class is giving data to table in database
8. To design the class 5 kinds of variable plays a major role
   1. Instance variable
   2. Instance read-only
   3. Static variable
   4. Static read-only - **pending**
   5. Constant **- pending**
9. Static constructor

**Day 1 code -**

internal class person

{

int age; //its instance variable

readonly int personid; // its instance read-only variable

static string office\_address;

static person() // static constructor

{

office\_address = "noida";

//Console.WriteLine("static constructor");

}

public person() // default constructor -> best practise is to go for public scope

{

Console.WriteLine("default constructor");

age = 1;

personid = 101;

}

public void assignage(int a)

{

Console.WriteLine("I am a function");

age = a;

//personid = id;

}

public static void Main()

{

Console.WriteLine("this is main function");

person harshita = new person(); // at this point compiler will call own default constructor to set a value called "0"

// we can make own default constructor to override the value given by the complier based default constructor

Console.WriteLine(harshita.age); // what will be the default value to age variable?

harshita.assignage(22);

harshita.assignage(23);

Console.WriteLine(harshita.age);

Console.WriteLine(harshita.personid);

//harshita.personid = 102;

Console.WriteLine(person.office\_address); // best way to access

//Console.WriteLine(office\_address);

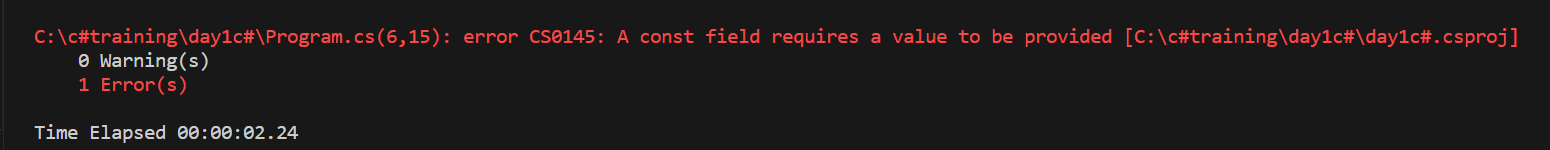
}

}

**Day 2 session**

1. Static read only
   * Static read only variable is similar to static variable means it is also a “class level variable”
   * static read only variable can only be initialized by “static constructor” means no function etc.
   * Use case – office leave policy, HR policy
2. Constant variable

* Can be declared at class level as well as function level
* For example – establishment year of company
* Can't be changed, and needs to be initialized at the point of declaration



Product class design -

Attributes -

|  |  |  |
| --- | --- | --- |
| Id | 101 | Instance read-only variable |

|  |  |  |
| --- | --- | --- |
| Product name | Surf | Instance variable |

|  |  |  |
| --- | --- | --- |
| Price | 100 | Instance variable |

|  |  |  |
| --- | --- | --- |
| Tax | 6% | Static read-only variable |

|  |  |  |
| --- | --- | --- |
| Product company name | Reliance | Static variable |

|  |  |  |
| --- | --- | --- |
| Company established year | 1992 | Constant variable |

* There are **2 classes** – product class and customer class
  + **product class** contains
    - all the **attributes of products**
    - as well as **constructors** to initialize the variables
    - and **member functions** to display the attributes
  + **Customer class** contains
    - **Main function** to **create an object** of class product and **call the member functions** through product object to display values
* To design the product class I have taken all 5 types of variables
  + “productId” as Instance read-only because it will be same for a particular product thr
* Have taken **2 constructors** – parameterized and static constructor
  + **Parameterized constructor** is used to initialize instance, instance readonly variable like -
    - product id,
    - name and
    - price as per each product
  + **Static constructor** is used to initialize static variables like
    - Tax
    - And company name

// defining the class for full-stack development and later on class will reflect the table

public class product

{

public readonly int productId;

public string productName;

public int productPrice;

public static readonly int tax;

public static string? companyName;

public const int companyEstablishmentYear = 1992;

public product(int pid, string pname, int price)

{

productId = pid;

productName = pname;

productPrice = price;

}

static product()

{

tax = 6;

companyName = "Reliance";

}

public void showProductMember()

{

Console.WriteLine(productId+" "+productName+" "+productPrice+" "+companyEstablishmentYear);

}

public static void showProductCommon()

{

Console.WriteLine(tax + "% " +companyName);

}

}

class customer

{

public static void Main()

{

// assume its a frontend written in angular

product product1 = new product(101, "surf", 100);

product1.showProductMember();

product.showProductCommon();

}

}

**Agenda day 2**

1. how function can return multiple values - **done**
2. what is out keyword
3. what is ref keyword
4. what is "struct"
5. what is "enum"

// case study 2

// we have to design the program in such a way that function can return multiple values

// STEP 1 - DEFINE THE MODEL (requirement gathering)

public class Employee

{

public readonly int Id;

public string? Name;

public int Salary;

public Employee(int id)

{

Id = id;

}

}

public class EmployeeDetails // EmployeeDetails class will define the employee

{

Employee empObj = new Employee(101);

public void details() // this function will contact to Employee class to get details

{

empObj.Name = "deepak";

empObj.Salary = 1000;

}

public void showDetails()

{

details();

Console.WriteLine(empObj.Id+" "+empObj.Name+" "+empObj.Salary);

}

}

public class Management

{

public static void Main()

{

EmployeeDetails empDetails = new EmployeeDetails();

//empDetails.details(); // breaking the abstraction model

empDetails.showDetails(); // following abstraction model

}

}

// case study 2

// we have to design the program in such a way that function can return multiple values

// STEP 1 - DEFINE THE MODEL (requirement gathering)

using System.Data.Common;

public class Department

{

public readonly int DeptId;

public string? DeptName;

public string? DeptLocation;

public Department(int id)

{

DeptId = id;

}

}

public class DepartmentDetails // DepartmentDetails class will define the Department

{

Department DeptObj = new Department(59);

public void deptDetails()

{

DeptObj.DeptName = "Digital";

DeptObj.DeptLocation = "Noida";

}

public void showDeptDetails()

{

deptDetails();

Console.WriteLine(DeptObj.DeptId+" "+DeptObj.DeptName+" "+DeptObj.DeptLocation);

}

}

public class DeptManagement

{

public static void Main()

{

DepartmentDetails deptDetailsObj = new DepartmentDetails();

deptDetailsObj.showDeptDetails();

}

}

**Feedback** - technical person says after evaluation of pre-assessment

1. need to improve the program logic and give me final version
2. rating 6/10...expecting 8/10

public class Department

{

public readonly int DeptId;

public string? DeptName;

public string? DeptLocation;

public Department(int id)

{

DeptId = id;

}

}

public class DepartmentDetails // DepartmentDetails class will define the Department

{

Department DeptObj = new Department(59);

public Department deptDetails()

{

DeptObj.DeptName = "Digital";

DeptObj.DeptLocation = "Noida";

return DeptObj;

}

}

public class DeptManagement

{

public static void Main()

{

DepartmentDetails deptDetailsObj = new DepartmentDetails();

Department Obj = deptDetailsObj.deptDetails();

Console.WriteLine(Obj.DeptId+" "+Obj.DeptName);

}

}

Case study 3 -

* Feedback – good
* rating - 8/10, expecting 9/10
* "Improve your model or requirement gathering class or say to instructor to teach major topic of c# properties"
* **What are properties** in c# then you can write actual model still model is written by participants is not "good practice" otherwise case study 2 is good.
* Object oriented programming paradigm

1. my data should be safe
2. no one can access the data without prior notification
3. it should be safe in such a way that there are no accidental alterations

* 11:30 - 12:30 - complete practical time
* case study 1 + case study 2 - revise
* **Agenda day 3-**

1. **properties** **-done -major topic for project**
2. **indexer** **-done -major topic for project**
3. out -done
4. ref -done
5. concept of inheritance

--->PROPERTIES

* should have similar name as the variable
* should have same return type as the datatype of variable
* should have public access modifier
* "get" and "set" are 2 accessors and can be accessed by class object
* get to return the value
* set is to set a "value" to the class variable

---> INDEXER

* indexer is enhancement version of properties
* limitation of properties is properties can bind single private variable so if we go for developing the project in c# we have 10 private variables it means we need to have 10 properties which can degrade the performance of the application
* why not can we design the application in such a way that 1 property can bind multiple private variables
* indexer is an array of properties
* CASE STUDY - if class consist of few private variables, then properties is good idea otherwise, we need to implement indexer

// code with one property

public class Applicant

{

private string? skills;

public string Applicant\_Skills // Applicant\_Skills is wrapper of skills variable

{

get // get is accessor

{

return skills;

}

set // set is accessor

{

skills = value; // "value" is a keyword to assign data to private variable

}

}

}

public class HR

{

public static void Main()

{

Applicant deepa = new Applicant();

deepa.Applicant\_Skills = "database"; // at this point set accessor will be called

Console.WriteLine(deepa.Applicant\_Skills); // at this point get will be called

}

}

//code with 2 properties

public class Applicant

{

private int age;

private string? name;

public int Applicant\_Age

{

get

{

return age;

}

set

{

age = value;

}

}

public string Applicant\_Name

{

get

{

return name;

}

set

{

name = value;

}

}

}

public class ApplicantInfo

{

public static void Main()

{

Applicant infoObj = new Applicant();

infoObj.Applicant\_Age = 22;

infoObj.Applicant\_Name = "Harshita";

Console.WriteLine(infoObj.Applicant\_Name+" "+infoObj.Applicant\_Age);

}

}

//INDEXOR IMPLEMENTATION

public class Applicant

{

private string[] myData = new string[3];

public string this[int i]

{

get

{

return myData[i];

}

set

{

myData[i] = value;

}

}

}

public class HR

{

public static void Main()

{

Applicant deepa = new Applicant(); // deepa will be referred as "this"

deepa[0] = "101"; // at this point set will be called and deepa[0] = this[0] = myData[0]

deepa[1] = "laptop"; // at this point set will be called and deepa[1] = this[1] = myData[1]

deepa[2] = "120"; // at this point set will be called and deepa[2] = this[2] = myData[2]

Console.WriteLine(deepa[0]+" "+deepa[1]+" "+deepa[2]);

}

}

* "**dynamic**" keyword - is a datatype in c# which can hold any type of data

--->OUT and REF

// CASE STUDY - 1. how many ways a function can return multiple values

//2.to open account in a bank (a.)2 photographs (b.)aadhar card (c.)account open

// code showing two return statements are not possible

public class bank

{

public string accountOpen()

{

return "carrying 2 photos";

return "carrying aadhar card"; // cant return 2

}

}

public class customer

{

public static void Main()

{

bank bankObj = new bank();

string data = bankObj.accountOpen();

Console.WriteLine(data);

}

}

**// OUT IMPLEMENTATION**

public class Numbers

{

public void nosReturn(out int A, out int B, out int C)

{

A = 100;

B = 200;

C = 300;

}

}

public class Customer

{

public static void Main()

{

Numbers obj = new Numbers();

obj.nosReturn(out int a, out int b, out int c);

Console.WriteLine(a + " " + b + " " + c);

}

}

**// OUT IMPLEMENTATION**

public class Bank

{ // adc is giving to a and pc is giving to p ,so we move from topto bottom

public void accountOpen(out string adc, out string pc)

{

adc = "aadhar card available";

pc = "carrying 2 photographs to open account";

}

}

public class Customer

{

public static void Main()

{

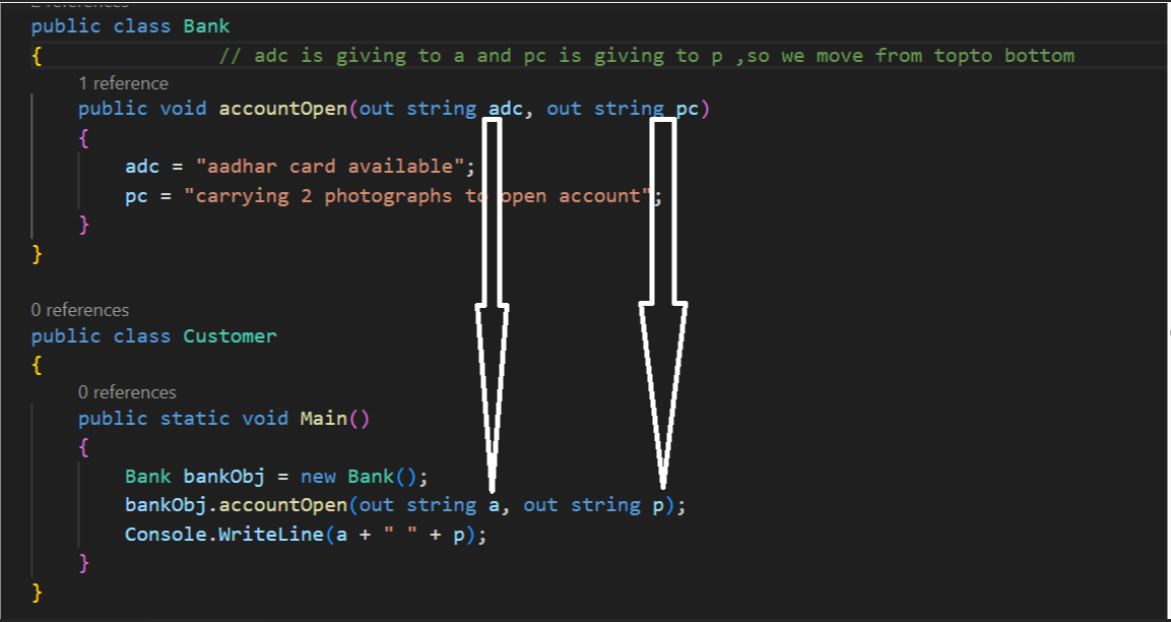
Bank bankObj = new Bank();

bankObj.accountOpen(out string a, out string p);

Console.WriteLine(a + " " + p);

}

}



**// this program shows out is not bidirectional**

public class Bank

{ // adc is giving to a and pc is giving to p ,so we move from topto bottom

public void accountOpen(out string adc, out string pc)

{

adc = "aadhar card available";

pc = "carrying 2 photographs to open account";

}

}

public class Customer

{

public static void Main()

{

Bank bankObj = new Bank();

string p = "one photograph";

bankObj.accountOpen(out string a, out p);

Console.WriteLine(a + " " + p);

}

}

**// ref implementation - string**

public class Bank

{

public void accountOpen(ref string aadhar)

{

aadhar = aadhar + "1234";

}

}

public class Customer

{

public static void Main()

{

Bank bankObj = new Bank();

string aad = "5678";

bankObj.accountOpen(ref aad);

Console.WriteLine(aad);

}

}

**// ref implementation - int**

public class Bank

{

public void accountOpen(ref int Amt)

{

Amt = Amt + 1;

}

}

public class Customer

{

public static void Main()

{

Bank bankObj = new Bank();

int amt = 100;

bankObj.accountOpen(ref amt);

Console.WriteLine(amt);

}

}

**AGENDA DAY 4 and 5**

* Inheritance -done
* concept of interface \*\*\*
* introduction to collection and generics \*\*\*
* runtime polymorphism
* What is use of “**base**” keyword in method hiding -done
* What is use of “**new**” keyword in method hiding -done
* What is use of **Constructor chaining** in inheritance -done
* What is role of constructor and its execution -done
* What is the use of virtual and override keyword.
* What is use of sealed keyword in run time polymorphism\*\*\*

Pending topics

* Interface
* Runtime polymorphism
* introduction to collection and generics \*\*\*
* lab time - 6:00 - 7:00
* after break - 2:05 - 2:30 - lab time

INHERITANCE

* INHERITANCE is based on generalization principle
* Generalization principle means we can go for classification
* Base class will always be generalized, and derived classes will be specialized, i.e. derived class will be classified under the base class, and there should be an association between them.
  + Example – base class -> engineer, derived class -> different branches like civil engineer
* Why do we go for inheritance – one class is not sufficient to define all the types
* Base class properties will be available in derived class
* **Virtual object** – when derived class object is created a base class hidden object is created internally first called virtual object, hence due to creation of this object base class constructor is invoked first before the derived class constructor
* Constructor will never be inherited

//inheritance - base and derived class constructor

// code 1 – day 4 – concept of virtual object of base class

public class Engineer

{

public Engineer()

{

Console.WriteLine("I am base class constructor");

}

}

public class Mechanical: Engineer

{

public Mechanical()

{

Console.WriteLine("I am derived class constructor");

}

public static void Main()

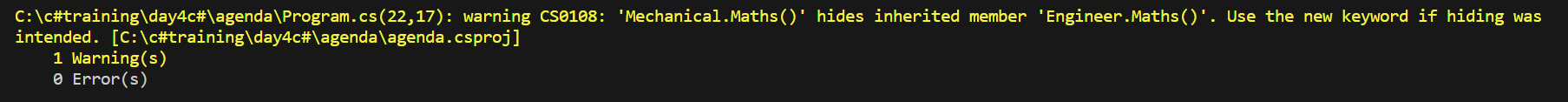
{

Mechanical obj = new Mechanical();

}

}

“new” keyword – for method hiding

//inheritance - method hiding

// code 2 – day 4 – new keyword for method hiding

public class Engineer

{

public Engineer()

{

Console.WriteLine("I am base class constructor");

}

public void Maths()

{

Console.WriteLine("I am maths - base class");

}

}

public class Mechanical: Engineer

{

public Mechanical()

{

Console.WriteLine("I am derived class constructor");

}

new public void Maths()

{

Console.WriteLine("I am maths - derived class");

}

public static void Main()

{

Mechanical obj = new Mechanical();

obj.Maths();

}

}

//inheritance - method hiding - base keyword

//code 3

public class Engineer

{

public void Maths()

{

Console.WriteLine("I am maths - base class");

}

}

public class Mechanical: Engineer

{

new public void Maths()

{

base.Maths(); // explicitly calling base class method

Console.WriteLine("I am maths - derived class");

}

public static void Main()

{

Mechanical obj = new Mechanical();

obj.Maths();

}

}

// why constructor chaining

// code 4

public class Manager

{

public Manager(string report)

{

Console.WriteLine("Yes, I am receiving your report "+report);

}

}

public class TL : Manager

{

public TL(string report)

{

Console.WriteLine(report);

}

public static void Main()

{

TL tlObj = new TL();

}

}

// REMARK - Compiler will not supply own default constructor, if programmer has taken parametrized constructor

// compile time error

**// why constructor chaining - CASE STUDY 2**

// code 5 – no error as default constructors present and base class default

// constructor and derived class parameterized constructor called

public class Manager

{

public Manager()

{

Console.WriteLine("default constructor from Manager class");

}

public Manager(string report)

{

Console.WriteLine("Yes, I am receiving your report "+report);

}

}

public class TL : Manager

{

public TL()

{

Console.WriteLine("default constructor from TL class");

}

public TL(string report)

{

Console.WriteLine(report);

}

public static void Main()

{

TL tlObj = new TL("Yes I prepared a report");

}

}

**// why constructor chaining - CASE STUDY 3**

// code 6 – to be able to access parameterized constructor of base class from

// derived class object

public class Father

{

public Father()

{

Console.WriteLine("default constructor from Father class");

}

public Father(string np1)

{

Console.WriteLine("Yes, I am receiving your newspaper "+np1);

}

}

public class Child : Father

{

public Child()

{

Console.WriteLine("default constructor from Child class");

}

public Child(string np) : base(np) // this is called constructor chaining

{

Console.WriteLine("Child class : " + np);

}

public static void Main()

{

Child tlObj = new Child("Times of India and aajtak");

}

}

**DAY 5**

Runtime polymorphism : Polymorphism is a method in which derived class can call base class as well as base class can call derived class. It can also fix inheritance.

* Loose coupling
* Extendibility
* An option
* Break dependencies

Sealed : It stop overriding

**Interface : It is used to separate the declaration from their implementation so that we can achieve high end of security.**

We have two ways to implement interface

1. Implicit interface
2. Explicit interface

Day 6

* What is collection ?

**Case study** – I want to design the application in such a way that we can have a variable which can hold multiple candidate's name

1. **Solution** -> If I use array in c# we can hold multiple names but limitation is we have to define the size of the array and we dont know initially total no of candidate list. If I cross the array size it will give runtime exception because declaring the array means we have to define the size so, array is static by nature and for multiple candidates we dont know total no of names.
2. Collection consist multiple inbuilt classes which provides dynamic array so we dont need to mention the size of the array initially means I want dynamic array and not static array
3. We have multiple types of dyanamic array available in collection namespace
4. Collection is a namespace in C# which provides several classes
5. **Example**: **ArrayList**- is dynamic by nature so we dont need to mention the size initially. Array is static by nature whereas ArrayList is dynamic by nature. ArrayList is class in collection namespace which provides multiple functions to solve the problem.

We can add, update, delete, search using ArrayList

**LIMITATION OF COLLECTIONS**

1. Collections are not strongly typed\*
2. Collections support boxing and unboxing\*
3. Boxing and unboxing degrade the performance of the application due to type casting.
4. Example – to demonstrate boxing and unboxing
5. Collection was introduced in c# 1.0 whereas generics was introduced in c# 2.0. Generics base classes are enhancement of collections.

Object is a root datatype in c#

Whenever we create an object of ArrayList class to class the add function , it internally hold the data in object type, later on applying the indexing to extract the data to assign string variable it gives compile time error.

* What is generics?

1. Generics are strongly typed
2. Generics are type safe
3. Generics avoid boxing and unboxing
4. Finally no conversion takes place if we go for using generics based classes
5. Generics is a namespace which consist multiple classes
6. Example
7. List<> is a generic inbuilt class, enhancement

* Difference between collection and generics?

Remark : in this example List<> is a generic class, <>needs datatype so that we can add similar data only thats why list is a strongly

Assignment – how to update a record, delete, search, count

**What is boxing?**

* Assigning the value type data to object type(reference type) is called boxing.
* Ex: Object y = 10 // 10 is a value type and y is reference type.
* Ex: string name = obj[2] // will give name

**Value Type :**

int, char, float, double, float, boolean, etc are value type are occupying stack memory

**Reference Type :**

object, string, interspace, etc are value type because occupying heap memory.

Assignement: Convert arraylist example to list with all the operation

**Day 7**

1. Advance part of generics
2. Advance part of collection
3. Generic class, Generic function
4. Exception handling
5. Introduction to threading
6. Enum keyword
7. Struct keyword
8. Implicit and Explicit interface

How to implement generics in project

Case Study:

1. I want to design an application to maintain the customer list
2. We can maintain customer id, customer name and customer phone no.

Remark:1

1. You are creating properties for accessing the private variable
2. Poor approach or old approach for creating the properties
3. Use c# 3.0 feature to create properties

Solution-

Remark- 2

1. Unprofessional approach to add the record

Remark –3

1. There is no interface, no abstraction, no encapsulation, no proper structure to develop on project level.

Solution : Creating the professional application

Folder Structure

Models

Functionality

Repository

Limitation of Employee Repository application

1. Only client.cs file will access the employee repository, assume that client.cs is Lokesh
2. What about Gaurav client and Ashwini client

Solution:

I want to create reusable repository known as library or dll so that multiple client can take the reference of library.

So we need to create a library in dotnet then we can consume the library in multiple application.

* There is no Main in libarary, because client will take the reference and he will have the main function.
* One library can have multiple console i.e multiple console

**Day 8**

1. Abstract Class
2. Struct
3. Enum
4. Dynamic and Var difference
5. Exception Handling introduction

What is Abstract class?

* Abstract is a keyword we can place before
* Class
* Function
* If a class is marked as abstract we cannot create an object of that class
* If a function is declared before abstract we cannot have the body of that function
* It means every function declared within the interface are abstract by default
* Why we go for interface ?

Reference Type

* Struct is a keyword to create a structure or user defined data type
* Memory of struct is allocated in stack memory
* Struct does not support instance or instance read only variable because these type of variable goes to heap memory
* Struct support static variable, static read-only variable, constant variable
* Struct does not support inheritance means struct is by default sealed
* Area of use—If any function of class does not use the instance variable why do we create class to have function, why not we can have struct to use function so that we can say struct is meant for creating the light weight object because no instance variable or instance read only variable.

**Day 9**

* Threading
* Delegate
* Garbage Collection
* Introduction to TPL(Task Parallel Library)
* Introduction to Reflection

**Var**

* Var is used for providing the short hand notation of user defined datatype or fundamental data type.
* Example -- var obj = new CustomerDetails();
* Var keyword is used only within the function, not function parameter and not at class level
* Whenever we declare variable with var keyword data has to assigned
* Example-- var x;
* Var support compile time checking known as early binding
* Example – var x = “Noida”;
* Remark- x is string type decided by var keyword at compile time
* If var is decided x is a string type later on you cannot change its type
* **dynamic**" keyword - is a datatype in c# which can hold any type of data

**Delegate**

Use :

* To achieve high abstraction model
* To achieve anonymous model : Eliminate the code of function and wrap in the delegate object is called delegate block
* To achieve lambda programming
* Event creation
* Callback\*\*\*
* To create Thread
* Multicast delegate

Case Study

// Delegate

public class MyApp

{

public static void A()

{

Console.WriteLine("Function a");

}

public static int B()

{

return 1;

}

public static string C()

{

return "c";

}

}

// .....................................

public class ClientA // A wants only String C function of that library

{

public static void Main()

{

}

}

Rules of using delegates

1. Delegate says show me function signature first-(public static string C())

2. It is advisable to declare the delegate before all the classes

3. Object of delegate is loosly coupled with classes and tightly

coupled with signature

Anonymous Method Delegate

* Small Delegate for Anonymous
* Capital Delegate for Threading

**Day 11**

Public int CustomerId {get; set;}

When we write public property without mentioning the private variable

1. Compiler internally creates private variable and variable name is almost similar to the public properties
2. We can have proof if we apply ildsm we can break meta data to find the private variable of that public properties.

Topics

1. Exception Handling
2. File Handing
3. Serialization\*
4. Callback in delegate
5. Event in delegate
6. Garbage collection

* When we are creating an object of StreamWriter class or StreamReader class
* Object is persist in memory till application is not closed
* Why not we can define the scope of an object of StreamWriter class or StreamReader class so that if we write or read operation is completed.
* We can destroy the object immediately to save the resources or memory
* ‘using’ is a keyword in c# for multiple uses
  + We can import the namespace by ‘using’ keyword. (eg - using System)
  + ‘using’ behaves as a block so that we can create an object within the using block and if we exit from using block automatically object should be destroyed.
  + Now implimenting the garbage collection so that we can destroy the object as soon as

Exception Handling

1. Meaning of exception is logical error
2. If exception comes we cannot continue in a program
3. If we apply condition exceptions can be minimized. Committed Exception can be minimized by validation and if-else conditions.
4. Some exceptions like divide by zero, array out of bound

**Architecture of thread**

T1 (thread) ..................................................................................................F1 (function)

T2 (thread)....................................................................................................F2 (function)

Thread obj = new Thread(F1);

obj.Start();

Thread obj2 = new Thread(F2);

obj2.Start();

// fortunately main thread is free or idle and can perform other task like F3 function

**What is threading and why threading is needed?\***\*\*

* System.Threading is a namespace in C#
* “Thread” is a class consists of several functions to start the thread and to stop the thread
* Clr provides single thread to Main() to perform the task and to develop the enterprise application single thread is not sufficient to perform multiple activities simultaneously, it is said to be blocking operation
* Thread is a class, and we can create an object to use parameterized constructor to assign the function signature which is long running by nature so that we can run the function over separate thread, we will not use the main thread
* Meanwhile main thread can call other function and new thread is also performing separate task to implement multithreading to achieve asynchronous programming

* Multithreading is good option when we go for developing enterprise application, when we have multiple function needs to be run simultaneously like downloading data and at same point of time fetching the record from file or database
* Multithreading can produce concurrency issue means if multiple threads are executing the same function there is a chance likely to occur concurrency problem known as deadlock situation so, ensure that use threads properly and don't try to assign multiple threads to single function as concurrency problem can arise

SQL Server

Ex 1

declare @name char(20)

set @name='Hemant'

print @name

Ex 2

Declare @age int

Set @age=20

if @age<10

begin

print 'correct'

End

Ex 3

declare @ctr int

set @ctr=1

while @ctr<10

begin

print @ctr

set @ctr=@ctr+1 //ctr++

End

int MyTask()

{  
Console.WriteLine("Hello World”);

return 2;

}

create function MyTask()

return int return

as

begin

return 2

end

print dbo.MyTask

We are calling the function bye dbo.MyTask

-- Factorial

/\*

CREATE FUNCTION dbo.Factorial (@iNumber INT)

RETURNS INT

AS

BEGIN

DECLARE @i INT

IF @iNumber <= 1

SET @i = 1

ELSE

SET @i = @iNumber \* dbo.Factorial(@iNumber - 1)

RETURN (@i)

END

SELECT dbo.Factorial(4) AS Factorial0f4

\*/

**DAY 14- SQL Day 4**

* Architecture of function in database -done
* Important rules by defining the function in database -done
* How to pass parameter to function -done
* Concept of returns and return keyword in function -done
* Integrate function with database table to perform the SQL query -done

**Agenda Day 15- SQL Day 3**

* Architecture of procedure\*\*\*
* How procedure is different from a function
* What is input procedure, output procedure
* Difference between input and output procedure
* What are limitations of return keyword
* How to perform transactional query using procedure
* Integrate procedure with database table to perform major SQL query
* How to call function from procedure
* How to call one procedure to another procedure
* How to handle the transaction in procedure
* How to handle the error handling in procedure

2:00 – 3:00 -> lab time

Like operator, in, between, and, or, not, where, orderby, groupby

Make 3-4 program for above each topics

Function

* We cannot use declare keyword in funtion parameter

Case Study

We have to create a function in database

1. Accept username and password, check in membership table if it is matched return 1 otherwise return 0
2. Write a program in SQL server to count number of records in membership table by applying the function

create table membership

(

username varchar(20) primary key,

passwords varchar(20)

)

insert into membership(username, passwords) values('Hemant', 'H123')

insert into membership(username, passwords) values('Amar', 'A234')

create function loginValidate(@usr varchar(20), @pwd varchar(20))

returns int

as

begin

declare @result int

select @result=count(\*) from membership where username=@usr and passwords=@pwd

if @result=1

return 1

return 0

end

print dbo.loginValidate('Hemant','H13')

SQL Day 3 (SQL procedure is most important for developing the database)

1. Stored procedure is special kind of function
2. It is an extended version of function
3. It is also said to be pre-compiled statement\*
4. Different between function and procedure

a. Procedure support DML operation (insert, update, delete).

i.) I can return multiple values

Ii.) I support out keyword to return multiple values

Iii. I able to pass parameter to procedure known as input procedure

Iv. Transactional keyword like begin transaction, rollback tran, commit tran

b. Function does not support DML operation or transaction

i. I can fetch data, search data support aggregation (max, min, sum, count)

Ii. Function says i can return one value

Iii. I do not support out keyword so i can’t return multiple values

Iv. I support function parameter2

v. I do not support transaction(begin tran, rollback tran, commit tran)

........................................................................................................................................

Function cannot call procedure whereas procedure can call function

Exec is a keyword used to execute the procedure

create proc myProc3

as

begin

return 'Noida'

end

declare @add varchar(20)

exec @add=myProc3

print @add

**Remark**

In the above example, obviously we will get the compile time error because procedure returns integer by default

**Solution**

So we have to use out keyword

create function fetchEmployeeRecord()

Returns

Remark : what will be the return type of returns

Generally we use out keyword in 2 cases

1. To return non-numeric values
2. To return multiple values
3. Out is similar to return keyword but return is specific due to by default int whereas out is generic return to return any kind of data
4. If we are using ‘out’ keyword it has complete different syntax to call the procedure

Advanced Stored Procedures

How we can perform

Assignment : input, output, return, multiple output, input with output

Serialization, delegate action, func, delegate, exception handling, thread, array, list, hash set,

Day 16

* Joins
* Data-Centric Applications and ADO.NET
* Connecting to data sources
* Performing connected database operations
* Building Datasets

Agenda

What is **ADO.NET** and its architecture

**System.Data.SqlClient**

**SqlClient** is a namespace consisting of several classes to connect backend

* + - 1. SqlConnection
         * We need to pass connection string to connect with backend using parameterized constructor
         * Open and close are the function of SqlConnection class
      2. **SqlCommand**
         * This class plays a major role to perform the operation
         * We can pass query like insert, update, delete, select with the help of parameterized constructor
         * To perform insert, update, delete known as DML operation, we have the function known as **executeNonQuery()** of SqlCommand class
         * ExecuteNonQuery() function return type: integer read function returns as a boolean value (True or False)
         * To fetch, search, concerned to retrival we have the function in SqlCommand class: **executeReader()**
         * ExecuteReader() function has return type: **SqlDataReader** class -- read is a function of SqlDataReader class and
         * To sum, count, max,min, average etc we have **executeScalar()** function
         * ExecuteScalar() function return type: object
      3. SqlDataReader
      4. Are the major classes to connect db

public MyApp()

{

myCon = new SqlConnection("Data Source=NL10-DF2670;Initial Catalog=db;Integrated Security=True;Encrypt=False;");

myCon.Open();

}

Day 17

1. Fetch entire records of single table
2. Fetch search record from single table
3. Fetch all record from multiple table
4. Fetch search record from multiple table and flash appropriate message with proper indexing architecture.

Remark – employee, student, course

**Inner Join** – fetching the common record in both table

select \* from student, course

where student.studentID = course.studentID

ADO.NET provides two ways to connect with database

* Connected mode
* Disconnected mode

1. In case of connected mode, SqlConnection, SqlCommand, SqlDataReader class plays a major role
2. Connected mode means there is a constant connection between backend and frontend
3. If record changes in database table frequently, we need to have constant connection with frontend otherwise frontend will not have information regarding new record is added in database
4. Weather forecast, news channel, air traffic signal are the best example of connected mode. we need to sync every time with database otherwise frontend is not aware about new record or if any transaction is done in backend.
5. Implementation
   1. We write the connection string with SqlConnection Class
   2. We write the query with SqlCommand Class
   3. We fetch the data by Sql DataReader class
   4. Sql DataReader class act as a pointer so that if any updation is occour in database it reflect back to SqlDataReader class
   5. SqlDataReader object as a pointer which points to database table known as connected mode

**System.Data** is a Root namespace which consists several classes to work in disconnected mode

1. DataSet
2. DataTable

Remark – These are the major classes to work in disconnected mode

SqlClient namespace consist of several classes

1. SqlConnection
2. SqlCommand
3. SqlDataReader

4. SqlDataAdapter

Final Architecture of disconnected mode:

1. SqlConnection
2. SqlDataAdapter
3. Dataset
4. These are major classes to work in disconnected mode

**Architecture of Disconnected Mode**

1. We can pass connection string by SqlConnection class
2. We can write the query by SqlDataAdapter class
3. SqlDataAdapter class is carrying the function called **fill()**.
4. Fill() function copy entire table or download the entire table in Dataset class
5. Once all the records is added in dataset we can work locally in dataset or we can fetch a record from dataset because dataset is replica of dataset table we will not feel the absence of database so we can say frontend is getting data from dataset known as disconnected mode
6. SqlDataAdapter class plays a major role to make it disconnected because it takes requested table from database and download entire records in dataset so frontend will get data from dataset not directly from database
7. Disconnected means we have a local copy of database table in dataset we will not feel the absensce of database
8. Ex: Laptop battery behaves as a dataset and laptop charger called adapter takes required current from AC and give it to battery, now battery is charged will not feel the absence of adapter or direct current. We can work locally with battery laptop this process is said to be disconnected mode.
9. Whenever adapter apply the Fill() function it automatically establishes the

**Project Level Implementation**

If record in database does not change frequently why do we remains constantly connected with database.

**Day 18**

**HTML, CSS, JavaScript**

Case Study:

1. Creating employee form and taking all the possibility of control
2. Apply CSS on HTML tag for better look and feel
3. Apply JavaScript on employee form to check form validation

**JavaScript**

1. JavaScript is a language which runs on client side known as browser
2. JavaScript is primarily meant for implementing client side validation
3. The most important part in javascipt is to define the function
4. The purpose of making the function in JavaScript to retrieve the controls id to validate whether it is blank or not or satisfy the criteria
5. Document.getElementByID(“.txtname”) is to retrieve the id that to check data entry is done or not
6. There is no proper data type in javascript
7. Var is a keyword to declare the variable but we cannot use in function parameter. We can use within the function

DEMO:



Integration of javascript in HTML

<body>

<form id="empForm" onsubmit = "myValidate()">

<input type="submit" value="Submit">

</form>

</body>

<script>

function myValidate()

{

alert("Hello World")

}

</script>

Remark

Submit is a datatype in HTML, if click on button onsubmit event will be called and onsubmit is attached with function

**Day 19-20**

ASP.NET training

1. What is ASP.NET
   1. It is a web development framework to build web application
   2. Web application is a collection of web pages like employee page, registration page, etc
   3. Every web page has 2 parts
      1. Aspx – UI part of web page to design the web page, apply css, apply js
      2. CS- it is code behind of aspx. Programming part, logic part, ado.net part is to be done, event part

Implement all the CRUD operations in ASP.NET Web form

Agenda

1. State Management architecture
2. Security in ASP.NET
3. We have to design the application in such a way that user cannot directly access Dashboard Page
4. Need to access the dashboard page only after login
5. Unprofessional code is written for this application and not acceptable
6. There is no **layering** in this application
7. There is no proper exception handling
8. There is no session to authenticate or security

select \* from membership where username='" + uname + "' and passwords='" + pwd + "'

**ADVANCE TRAINING**

**Day 1**

* Language Enhancements in c#3.0, 4.0, 5.0 and .NET 4.5
  + Dynamic
  + Var
  + Using
  + File handling operations
  + ADO.NET concepts improvement
  + State management(Session)
  + Enhancement part of threading(how to handle concurrency issues in multi threading)
    - Lock – This is a shorthand notation of monitor class
    - Monitor- this is an old technique to avoid concurrency
  + State Management Architecture
    - Session is inbuilt object in ASP.NET to maintain the value for multiple requests or visit from one to another page, unless and until you don't logout from website
    - What is the default timeout of session 20 minutes.
    - Example
    - This means when we go for developing the website, we can have multiple web pages
      * Sign-up
      * Sign-in
      * Home or dashboard – just after login
    - If a user logged in our site his name should be displayed if he visits any web pages and he can sign-out or logout at any point
    - Sign out means i am destroying the session explicitly or kill the object
    - Syntax of creating session -> syntax of retrieving session- > syntax of logout

Case Study for logout

Logout means we have to kill the session and redirect to login page

Case Study 1:

How i can picturize the concurrency problem in multithreading

Case Study 2:

How can i solve

**Day 2**

Agenda

1. What is EF Core
   1. Dotnet core is a new framework of microsoft
   2. It is a open source framework means we can install on mac, linux, etc
   3. It provides own local server(kestrel) to run the application
   4. Kestrel is a local server known as development server to run the application
   5. Dotnet core provides 2 major products
      1. Core MVC
      2. Core WEB API(there is no presentation layer or view) - It provides end point or URL known as route to display data in browser
      3. Core Web API is also said to be “HTTP service” known as restful service
      4. Example- HTTP:localhost:7200/employeedetails
         1. HTTP:localhost:7200/employeedetails is a route known as end point
         2. If we request the route or URL from browser, will get data
         3. HTTP:localhost:7200/employeedetails is created by CORE WEB API and to connect database we will use EF CORE
         4. It means there are 2 ways we can connect to database
            1. ADO.NET(.NET)...2002---ASP.NET, ADO.NET, MVC(2011)
            2. .NET CORE...2017--CORE MVC, CORE WEB API\*, EF CORE, SQL Server
            3. Finally microsoft introduced

.NET---2002-2016—ASP.NET, ADO.NET, basic c#

.NET CORE---2017-till date—Advance c#, generics

<https://jsonplaceholder.typicode.com/users> (written in CORE WEB API+ EF CORE+ SQL SERVER )

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

ASP.NET, MVC, JSP, PHP, REACT.JS, ANGULAR

Tuesday- able to create service and will explain the files structure of Core Web API

Wednesday-

Thursday-

Friday- Core Web API, Advanced C#, EF CORE

Monday- Frontend by CORE MVC

**SERVICE SIDE**

**IMPLIMENTATION**

1. C# + EF CORE = produce the output in console (CRUD)
   1. Installing 4 package to work in EF Core
      1. Microsoft.EntityFrameworkCore
      2. Microsoft.EntityFrameworkCore.SqlServer
      3. Microsoft.EntityFrameworkCore.Design
      4. Microsoft.EntityFrameworkCore.Tools

b. **EF Core achitecture**

* + 1. EF Core provide 2 major classes
       1. **DbContext**
          1. DbContext class supports DML operations and has major function **SaveChange()**
          2. SaveChange() - for insert, update, delete
          3. Most of the functions of DbContext are **virtual**
          4. On configuring(), **OnModelCreating()**
          5. Remark—DbContext class behaviour is similar to SQLCommand of ADO.NET

2.  **DbSet<>**

a. DbSet is a generic class

b. We can fetch, search, joins

c. We can apply the **LINQ** query to perform the operations

d. DbSet does not support SQL query so we have to use LINQ Query

e. LINQ query is only a solution in EF core to fetch,search and filter, etc.

f. EF Core does’t support SQL Query

g. Diff between SQL query and LINQ query

h. DbSet is used to generate table

i. public DbSet<Employee> Employees { get; set;}

Employees will be table name and columns will be provided by employee class

Diff between SQL query and LINQ query

dotnet tool install --global dotnet-ef --version 8.0.2 // Command to configure EF core

There are 2 ways in EF Core we can connect to backend

* 1. Code first approach
     1. In case of code first approach, database and table will generate dynamically
     2. We have defined the employee class
     3. We have defined the database class
     4. Now we can apply migration technique to generate database and tables dynamically
     5. We have to execute two commands for migration
     6. “**Dotnet ef migration add phase1**”--- after executing this command migration folder will be created and within the migration folder, script file will be available
  2. DB first approach(recommended for project)
     1. In case of Db first we have database and table

1. Now integrate C# + EF Core WEB API =output in browser

**Day 3**

**Agenda**

1. Now migration is done
2. Database is created successfully
3. Now we have to perform database operations using DbContext and DbSet<>

Now we are converting C# + EF CORE in CORE WEB API

1. What is CORE WEB API(main target in exl)
2. CORE WEB API architecture
3. What is controller
   1. Controller is a mechanism to create endpoint for GET, PUT, POST, DELETE
4. What are HTTP verbs
   1. [HttpGet("exltraining")]
   2. exltraining is a url of training() to access from browser
5. What is tight coupling
6. What is loose coupling
7. What is inversion of control(IOC)
8. What is Dependency injection
9. CORE WEB API request pipeline
10. What is middleware
11. Complete CRUD operation to create end points
12. How to validate the models
13. What is fluent API pattern
14. What is swagger
    1. Swagger is a user interface to display the output or we can directly display the output in browser.
    2. By browser we cannot post the data, update data or delete data, we can only surf.
    3. Swagger is a extended version of browser because we can perform, get, post, put, delete
15. What is DTO(Data Transfer Object)

* dotnet new webapi --name EmployeeService
* dotnet build
* dotnet watch run
* dotnet add package microsoft.entityframeworkcore
* dotnet add package microsoft.entityframeworkcore.tools
* dotnet add package microsoft.entityframeworkcore.design
* dotnet add package microsoft.entityframeworkcore.sqlserver
* dotnet restore

What is program.cs file?

What is the use of appsettings.json file

What is launchSettings.json file?

What is the use of EmployeeService.csproj file?\*\*\*

//////////////////////Dummy Endpoint WEB CORE API

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at <https://aka.ms/aspnetcore/swashbuckle>

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

var summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

app.MapGet("/weatherforecast", () =>

{

var forecast = Enumerable.Range(1, 5).Select(index =>

new WeatherForecast

(

DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

Random.Shared.Next(-20, 55),

summaries[Random.Shared.Next(summaries.Length)]

))

.ToArray();

return forecast;

})

.WithName("GetWeatherForecast")

.WithOpenApi();

app.Run();

record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)

{

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

}

///////////////////////////////////////////WE have removed certain lines because dotnet core is giving dummy endpoint with fake records

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at <https://aka.ms/aspnetcore/swashbuckle>

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.Run();

**How to create end point for GET, POST, PUT, DELETE**

Example of creating endpoint for GET operation

[HttpGet("exltraining")]

public IActionResult training()

{

return Ok("Training for basic and advanced .NET") }

To create the endpoint so that we can display in browser certain things need to be configured

builder.Services.AddControllers();

app.MapControllers();

**If we have multiple HTTP GET or more than 1 HTTP GET then we have to apply Route**

Recommendations – Don't pass URL by [HttpGet] but use [Route(“”)]

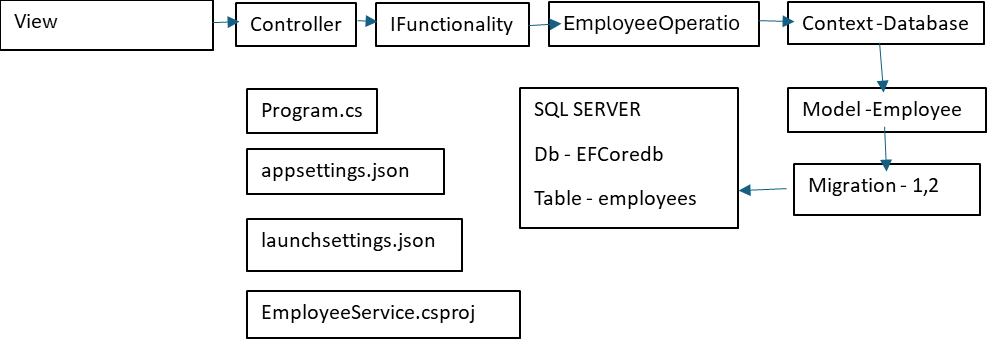
Agenda

Integrating EF Core + C# + Core WEB API

Operations – Insert, Fetch

**Application Architecture-Written in CORE WEB API**

**Application Architecture-Written in CORE MVC**



**DAY 5**

Remark – complete unprofessional application

1. Why controller is creating an object of EmployeeOpeartion (tight coupling)
   1. It violets the single responsibility principle
2. Why employee operation is creating an object of context class

* Name the case study
* Architecture with data flow digram – request response diagram
* 1-2 min explaination of architecture
* Technology – Dotnet core Web API + EF Core
* Code first or DB first
* EF Core tools one time configuration

Assignment

While posting the data from swagger, we should validate

DAY 6

* Implementation of IOC and DI
* Delete, Update, Search
* Model Validation

Remark – OIC and DI

We have identified in the application where new operator was used i.e. in 2places

We commented those two lines in controller and employeeoperations class

No we will apply IOC means register the dependent classes in .NET CORE framework so that .NET CORE framework will take care for instantiation

Then we have to apply Dependency Injection technique for dependent classes

There are 3 ways we can achieve Dependency Injection technique

Constructor Parameter(Recommended)

We need to pass dependent classes(EmployeeOperation, DatabaseContext) to constructor parameter

Method Parameter

Properties

**Model Validation**

* **While** we are posting the data from swagger, it is controller responsibility to filter the data then pass to next layer i.e. EmployeeOperation which will then pass to DatabaseContext which in return passes data to database table
* There are multiple ways we can apply model validation in .NET CORE
  + **Data Annotation**
  + **Fluent Model Validation**, to validate the incoming request (employee)

DTO(Data Transfer Object)

* While we are posting the data from controller to model, we should not use the actual model name ‘Employee’
* Employee class refer to table known as entity
* We can make a clone of Employee class so that we can pass clone class name by function parameter of controller
* Later on clone class will give data to actual class (Employee) for security reason
* Clone class is said to be DTO

**What is view?**

* Architecture wise, application wise, implementation wise no difference between CORE WEB API and CORE MVC
* Only we can attach the view known as UI like aspx page in ASP.NET to create UI
* CORE WEB API is meant for making service whereas CORE MVC is web application similar to ASP.NET

**Middleware**

* We have inbuilt middleware in dotnet core which runs before rendering the output in browser
* Example – app.MapControllers();

**Steps to implement Model Validation**

* Install the package to support fluent model validation
* **dotnet add package Fluent.Net --version 11.9.0**
* RuleFor(x => x.ename).NotEmpty().WithMessage("Employee Name cannot be empty");
* RuleFor(x => x.esalary).GreaterThan(0).WithMessage("Salary must be greater than 0");
* Now we have to register the validator in program.cs file
* **dotnet add package FluentValidation.AspNetCore**

**DAY 7**

**Standard or best practices to create controller**

* Controller name should have a suffix as “controller”
  + Example- HemantController.cs
* HemantController will be inheriting from ‘**ControllerBase**’ class
* ControllerBase class encapsulate the feature of Http Verbs(HttpGet, HttpPost)
* Function name in controller does not make any sense unless and until we don’t define route
* HemantController is a class does not make any sense until we don’t define route before the class
* HemantController + route + name of function + route = endpoint

[Route("api/[controller]")]

* Api is a route prefix and [Controller] is a keyword
* [Controller] refers to EmployeeController class

[Route("AddEmployeeDetails")] refer to function name addEmployeeDetails

**Implementation of IOC and DI**

* We have identified 2 classes are tightly coupled in this application
* DatabaseContext, EmployeeOperation
* We have to register the databaseContext class and employeeOperation class in .NET core for implicit creation of an object
* Example -
* We need to apply DI for tightly coupled classes
* DatabaseContext – we have different syntax to register in .NET CORE
* EmployeeOperation – we have different syntax to register in .NET CORE
* Because employee EmployeeOperation is implementing the interface IFunctionality

**DAY 8**

* **DTO**
* **DB FIRST**
* **Migrate the project to VS 2022**  ---done

**DTO**

* Model is said to be POCO(Plain Object) class means no extra annotations or extra responsibility which follow SOLID principle
* Model later on represent to table after applying migration
* Model is said to be entity
* In real life situation model is similar to actual documents of my marksheet
* We have to have clone of actual marksheet to apply for jobs
* DTO is similar to clone of actual model
* FINAL ARCHITECTURE OF DOTNET CORE WEB API
* DTO is also know as view model

Function written in controller is known as action

**DB FIRST approach**

* Ensure that database and tables are created already
* Now we have to create new application in core web api to perform all the operations
* Working of DB FIRST APPROACH
  + Ensure that database and tables are created already
  + Context classes will be created implicitly
  + Model class will be created implicitly
  + Migration is required
  + We have created a Models folder but classes and context will be created implicitly we do not need to add ourself
  + Code First—Class-->Migration-->table
  + DB First – table--> Scaffolding(tool)--> Class
  + Scaffolding is a tool is to be use in DB first approach to generate classes and context
  + dotnet ef dbcontext scaffold "data source=NL10-DF2670;Initial Catalog=efCoredbApiProject;Integrated Security=True;Encrypt=False; " Microsoft.EntityFrameworkCore.SqlServer -o Models
  + Remark – While we are posting data for POST, PUT, DELETE and client says do not expose actual model to action parameter of controller
    - Apply DTO and we can write multiple DTO depending on scenario
    - If we pass DTO as action parameter then obviously we have to apply model validation on DTO level and not Model level
    - Finally we will apply DTO validation but not annotation which break single responsibility principle

**DAY 9**

* CORE MVC
* Model Binder
* Exploring ASP.NET MVC
* Working with controller in ASP.NET MVC
* Using the Razor View Engine
* Building a resilient ASP.NET MVC 5 Web application
* Working with Data

Case Study 1

Perform CRUD operation with project level architecture using code first approach

Difference between CORE WEB API and CORE WEB MVC

1. CORE WEB API is said to be a RESTFUL service means HTTP service, we can access from the browser
2. There is no view or RAZOR view engine in CORE WEB API to load the view
3. CORE WEB API is primarily meant for creating service
4. CORE MVC is said to be web application like ASP.NET, PHP
5. It is said to be server side application because C# runs on server side not client side.
6. Razor is a programming model is to be implemented in view
7. Extension of view is .cshtml file
8. We can do programming on view in C# language with the help of razor programming
9. Example
   1. .cshtml + C# = Razor programming
10. This is said to be razor programming

@{

string name = "Hemant";

}

@name

**DAY 10**

Class--->Function--->View--->Output in browser(CORE MVC)

Class--->Function--->Browser--->CORE WEB API

Assignment – Viewbag, viewdata, tempdata, viewmodel

**DAY 11**

Remark- We need to complete the update, delete, insert

We have taken the prior permission from exl to complete typescript and angular with depth knowledge

AGENDA DAY 11

* What is typescript?
* How to configure typescript
  + Node is a platform to build or support **JS** based applications
  + React, Angular, Vue.js....these are application which runs on node platform like windows is a platform for Dotnet Core etc
  + Now we can install angular framework or react on node platform
* Why do we learn typescript
* Why exl has given typescript for angular

Remark

1. Constructor can’t be overloaded in typescript
2. Constructor is a keyword to create constructor in typescript

DAY 12

* Angular 17
* Architecture of Angular 17
* File Structure
* Architecture of component in angular
* What is component
  + Component is a feature of angular to create tag known as vocabulary
  + Class-->new type-->
  + Object is a mechanism to access new type
  + Component helps to create new tag
  + Typescript is a language to access new tag

Browser(client) Web(Server)

HTML Angular, CORE MVC

DOM(Document Object Model)

* DOM is a collection of objects and JavaScript is a language to access those object available in DOM
* Objects are based on nested class architecture or nested principle(object).

Navigator

Windows

Documents

Form

(all objects concern to UI like button, checkbox, textbox, etc)

Browser+DOM+Angular = Browser+DOM+New Tag

AGENDA DAY 13

* What is binding in angular
* Types of binding in angular
* What is models in angular

**Case Study:** How to fetch a record (data entry) in angular user interface using data binding.

* In the app folder in explorer, we will make a folder ‘Models’
* In this Models folder, make a file Employee.ts in which we will make our Employee class.
* Now similarly, we will make a folder “Services” , in which we make a function and add the records.
* Now in app.component.ts, We have a class called AppComponent in which we define a function
* And using this function, we use **Data Binding** in app.component.html to fetch the records on the UI.

Case Study 2

How to show multiple employee records in angular UI

1. Important directives(ngif, ngfor, ngelse) in angular
   1. We will use these directives in UI(similar to razor view engine in core mvc)

**DAY 14**

* **Inversion of control(IOC)**
  + Tight coupling- why component is creating an object of service class
  + Angular as a framework support inversion of control to solve the dependencies or we can say , we can give dependent classes to angular framework so that memory allocation part will be done by angular
  + @Injectable()
  + If any class is marked as a injectable. We don’t need to create an object of that class, then we can apply DI
  + providers: [EmployeeService]
  + Providers is a attribute of component directive, we can register EmployeeService class.
* **Dependency injection(DI)**
  + Dependency injection is a technique to achieve IOC and there are three ways we can apply DI
    - Constructor Parameter
    - Method Parameter
    - Properties
* Instance management
* Types of instance management
  + Percall\*\* providers: [EmployeeService]
    - If we register the if we register the Employeeservice class in providers attribute of component , then every request new object will be created known as percall.
  + Singleton providers: [provideRouter(routes),EmployeeService]
    - If we register the if we register the Employeeservice class in App.Confige.ts file, then that object will be global for all the components and said to be Singleton.
* Routing(most important for project POV)

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

Advance Part 2

1. Routing
2. Injectable
3. Form Programming
   1. Template driven approach
   2. Reactive approach

**ANGULAR PROJECT**

* Full Stack Development / end to end project with syllabus
* We are creating a POC for EXL so that we can show what we learnt in this training

**DAY 15**

We have to create two endpoints

1. Create user Account
2. Create login for that account

Empid,name,salary,designation,phone

Ng g c Register

DAY 16

* We have to focus on angular project to connect with core web API to perform complete operations to create full stack architecture

Configure bootstrap in angular

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

"node\_modules/bootstrap/dist/js/bootstrap.min.js"

Reactive approach is professional approach in angular to create a UI

What is ngOnInit()?

Remark :

* If any class support DI, then that class must be marked as an injectable or ProvidedIn
* ProvidedIn is a keyword introduced in angular12 or enhancement of injectable
* If any class support DI, that class also support the DI for the given component

What is observalble?

What is rxjs?

What is CORS?

* Cross Origin Resource Sharing
* If we don’t enable CORS in Core Web API, we can’t access from frontend-Angular or React

DAY 17

**Agenda:**

* Routing
  + Routing is one of the most important feature in Angular
  + Routing means how component can produce URL or Route
  + Register Component.....URL
  + Login Component....URL
  + E-Shop/Grocery Component.....URL
* Security
* How to handle exception handling?
* Navigation in project
* We will work on Employee module
* Route Navigation—can you explain the navigation
  + Signup-->Signin-->Dashboard(Employee-list)

Employee List with pagination

Proper tabular format

Logout functionality