Keywords : -- static ,  this ,

OOPS intro

Pillars of oops

Advt and Disadvt of OOPs

Method  , properties , Namespace

Folder creation

Strategies  --- >   OOPS   ---> Principles

Structured Programming

Procedural Programming  or Modular Programming -- When we create different set of functions(Operations to perform) i.e a Modular Programming

OOPS Programming -  When we create different set of functions(Operations to perform but here we call as object .. Each object has its relevant functions

In oops the Software --- is a collection of objects containing functions and data related to those functions but in Modular  programming the system was  a collection of functions

Problems in Modular Programming

1. Reusability --  We must write the same code or logic at multiple places, increasing code duplication.
2. Extensibility -- wouldn't be able to extend easily with additional features.
3. Simplicity -- we usually end up with many functions and code in scattered manner.
4. Maintainability : It was difficult to manage and maintain the application code.

OOPS came into the picture to overcome the problems we had in Modular Programming:

1. Reusability :: Class & Objects
2. Extensibility :: Inheritance  --- Aggregation  / Composition
3. Simplicity :: Abstraction , Encapsulation and Polymorphism
4. Maintainability :: If we combine all

**Abstraction** :: Hiding or removing unnecessary things , providing the essential features without including   the background details  … For eg : Google search bar  , ATM Machine ( Logical thinking)

**Encapsulation** : Binding the data and functions together into a single unit .( Physical Implementation)

 For eg:

Private String Salary   -- public getSalary(); --- ReadOnly

Private String employee name  --- public String SetName(String name){ Read / Write  } ,, public getName()

**Inheritance**  :  The class from which the members are transferred is called  Parent / Base / Super

                          The class that inherits the Parent/ Base / Superclass members is called Derived / Child /      Subclass

**Polymorphism** : We can say that the same function will have or show the different behaviour by taking different types of values or with a different number of values  … The ability to take more than one form.

1. **Static Polymorhism/ Compile-time Polymorphism / Early binding. ( Overloading)**

Person p = new Person(int a ,float b);

p.display(20,30.5);

Class ABC

{

Void display();

Void display(int a);

Void display(int a , float b);

}

1. **Dynamic Polymorphism/ Run time Polymorphism/ Late Binding ( Overriding)**

Person p = null;

P = new Father();

p.jd();

Person p = new Employee();

p.jd();

Person p1 = new Employee();

p.jd();

Person p = new SocialWorker();

Person p = new Person();

Father f = new Person();

Class Person

{

Private int I;

Private Void jd()

{}

}

 class Father extends Person

{

Void jd()

{}

}

Class Employee extends Person

{

Void  jd()

{}

}

15 mins break

CLASS & OBJECTS :

Class is a blueprint --- an architecture  -- structure  --- Objects which are belonging to that class has it's own properties and behaviour

In a govt sector

The residential building they are offering to their employees , the structure or a blue print of all the flats are same

2 bed room sets -- MIG

3 bed room sets --HIG

The design which we create is a class and the houses created with the help of that design are objects

A class is a user defined data type that represents both state and behaviour

State  -- Properites

Behaviour -- action

A cLass is composed of three things  --- name ,  attributes and operations

Public class <Classname>

{

Int a=20 , int b;   // states

   public int input(int a , int b )  // behaviour

{

Return a+b;

}

}

}

<classname> <objectname> = new <classname()>;

<objectname>.input(20,30)

Types of Classes

Abstract

Partial

Concrete

Sealed

Static

// Substitute of Copy Constructor

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OopsPrograms

{

    internal class Person

    {

        public string Name;

        public int Age;

        // default constructor

        public Person(string name , int age) {

            Name = name;

            Age = age;

        }

        public Person(Person p)

        {

            Name = p.Name;

            Age = p.Age;

        }

        public void Display()

        {

            Console.WriteLine($"Name : {Name} and Age : {Age}");

        }

    }

    class MainProgram

    {

        static void Main(string[] args)

        {

            Person person1 = new Person("Niti", 35);

            person1.Display();

            Person person2 = new Person(person1);

            person2.Display();

            person2.Name = "Aditi";

            person2.Age = 20;

            person2.Display();

        }

    }

}

Keywords  :  private , public , protected , abstract , static i. e..    Modifiers

Access Specifier :  - special kind of modifiers which is used to define the scope of a type(class, Interface, Delegates) and its Members( Variables, properties,

Constructor and methods.

Different types of AS in C#

1. Private
2. Public
3. Protected
4. Internal
5. Protected Internal
6. Private protected(7.2 onwards)

Public class Employee  <Type>  we can define that class as Public or internal and by default it's internal

{

Private int id;      // Variable

Public string Name {get; set;}   //Properties

Public Employee(){} //Constructor

Protected void Display(){} // Method

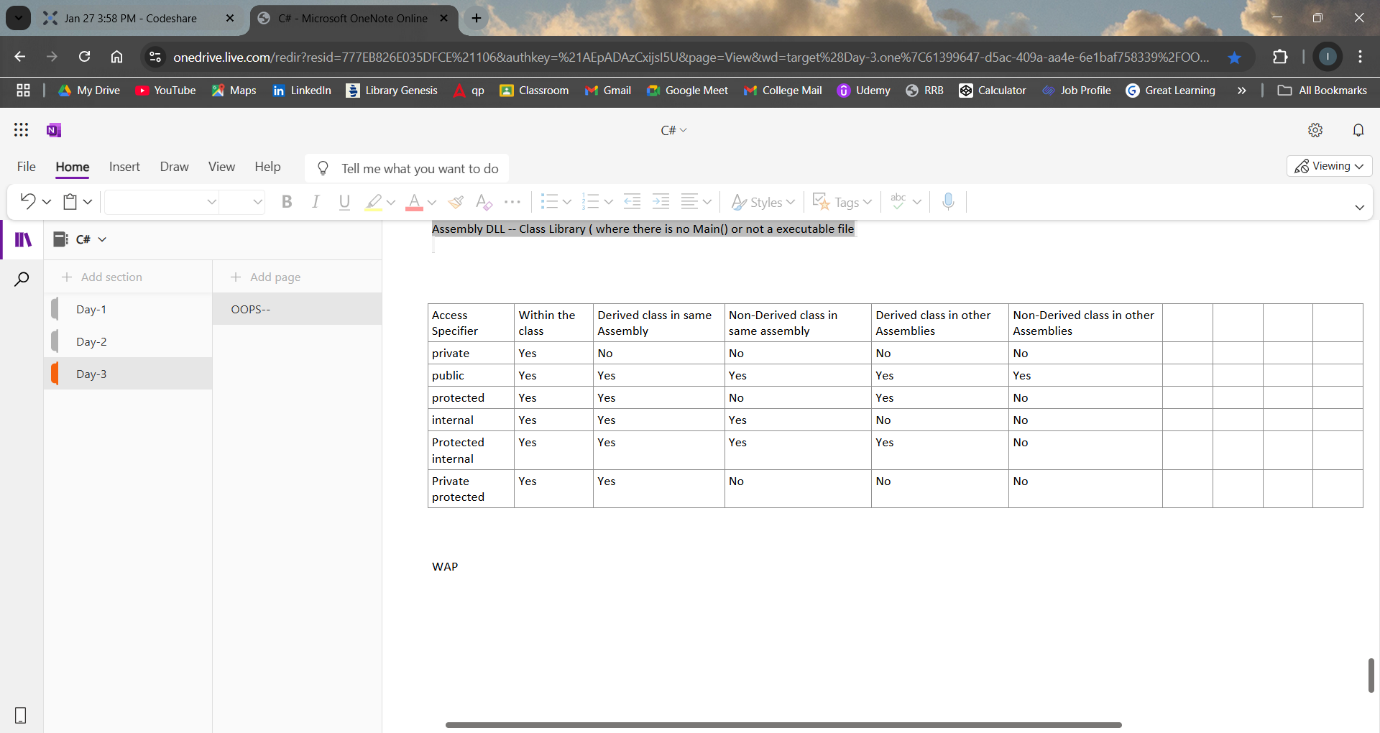
}

Assemblies :  are the building block of .Net Framework applications and also a fundamental unit of deployment . It is a precompiled .Net Code that can by run by CLR

Two types of Assemblies

Assembly EXE -- Console Application ( having Main() and it's a executable file

Assembly DLL -- Class Library ( where there is no Main() or not a executable file



switch (choice)

{

case 1:

Console.Write("Enter Student ID: ");

int id = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Student Name: ");

string name = Console.ReadLine();

Console.Write("Enter Student Age: ");

int age = Convert.ToInt32(Console.ReadLine());

students.Add(new Student(id, name, age)); // Using ArrayList's Add() method

Console.WriteLine("Student added successfully.");

break;

case 2:

Console.WriteLine("\nList of Students:");

if (students.Count == 0)

{

Console.WriteLine("No students found.");

}

else

{

foreach (Student student in students)

{

Console.WriteLine($"ID: {student.Id}, Name: {student.Name}, Age: {student.Age}");

}

}

break;

case 3:

Console.Write("Enter Student ID to remove: ");

int removeId = Convert.ToInt32(Console.ReadLine());

Student studentToRemove = null;

foreach (Student student in students)

{

if (student.Id == removeId)

{

studentToRemove = student;

break;

}

}

if (studentToRemove != null)

{

students.Remove(studentToRemove); // Using ArrayList's Remove() method

Console.WriteLine("Student removed successfully.");

}

else

{

Console.WriteLine("Student ID not found.");

}

break;

case 4:

Console.WriteLine("Exiting program...");

break;

default:

Console.WriteLine("Invalid choice. Please try again.");

break;

}