**Session 12: Four Pillars (Abstraction, Encapsulation, Inheritance, Polymorphism).**

* **Abstraction:-** display only essential information. Private keyword is used to hiding. Hiding implemetetion detail from user only display the functionality. Identifying only required charachteristic of an object. Don’t need to no about the internal process. Reduce the complexity. Avoid duplication of code. Helps to increase security. Only important details will provide. Cannot display the background detail. Provide generalised view. Can be achive abstract class, method,**interface.**
  + **Eg:** -Car
* **Encapsulation:-** data hiding process.wrapping data in single unit.its protective shield that prevent data from being used. Data can be hidden from another class. Variables are private. Easy to reuse. Variable of class can be hidden to other. It can be accessed only through method or properties of their current class.
* **Polymorphism:** perform single action by difrent way. Poly means many and morph means form so it indicate that many forms. **Static Polymorphism:** Compile Time Polymorphism – linking function with object during compile time. Also known as early binding,fixed binding. **Techniques to implement**: **Method overloading:-**having multiple definition with same function name in scope(class body). The name should be same but parameter different. Cannot overload function declaration that differ only by return type. **And Operator Overloading**. **Dynamic Polymorphism:** Runtime Polymorphism **Method Overriding**:- it allows us to have virtual and abstact methods in base using derived class with same name and same parameter.
* **Inheritance:** inheriting one class to another. Crete new class from existing class. It contain base(Parent) and derived(Child) class. Reduce code complexity. **:** is used to inherit the class.
  + **Eg:**- Man>> Teacher >> Student >>Businessman
* **Types of Inheritance:Single Inheritance:**- it contain only one parent class and one child class.
* **Hirarchical Inheritance:**- one base class can be derived from more than one child class called hierarchical. This is used when one class nedded in multiple class.
* **Multilevel Inheritance:** child class is accessing one parent class and again the new class is accessing child class which is already inherited from parent.
  + **Eg: class one(Base) class two(Child)**
  + **Class one : class two >>> class three(New Child)**
    - **Class three : class one**
* **Multiple Inheritance:** one class can have more than one superclass it inherit from all patent. C# doesn’t support multiple inheritance. But we can use **Inertface** to inherit in multiple inheritance. Using class its not possible. But using interface it allowed.

S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace OOPS\_\_AllSession

{

class S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc

{

int age;

private string employeeName;

public string address;

//\*\*\*\*\*\*\*\*\*\*\*\*\* Abstraction \*\*\*\*\*\*\*\*\*\*\*

void Abstraction\_Method()

{

double salary;

const double tax = 0.1;

double netSalary;

Console.Write("\nEnter Employee Name: ");

employeeName = Console.ReadLine();

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

age = int.Parse(Console.ReadLine());

Console.Write("Enter Net Pay: ");

netSalary = double.Parse(Console.ReadLine());

//Calling Encapsulation Method//

Encapsulation encap = new Encapsulation();

encap.AccessingEncapsulatedData();

if (netSalary >= 30000)

{

salary = netSalary - (tax \* netSalary);

Console.Write($"\*\*\*\*\*\*\*\*\*Employee Detail is \*\*\*\*\*\*\*\*\nEmployee Name: {employeeName}\nEmployee Age: {age}\nEmployee Address: {address}\nSalary: {salary} \n");

}

else

Console.Write($"\*\*\*\*\*\*\*\*\*Employee Detail is \*\*\*\*\*\*\*\*\nEmployee Name: {employeeName}\nEmployee Age: {age}\nEmployee Address: {address}\nSalary: {netSalary} \n");

}

public void NonAbstraction\_Method()

{

this.Abstraction\_Method();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\* Encapsulation \*\*\*\*\*\*\*\*\*\*\*

class Encapsulation

{

public void AccessingEncapsulatedData()

{

S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc accessEncapData = new S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc();

Console.Write("Enter Employee Address: ");

accessEncapData.address = Console.ReadLine();

//Console.Write("\nEnter Employee Age: "); //Error

//accessEncapData.age = int.Parse(Console.ReadLine());

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Polymorphism \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public void PolyMorphism()

{

Encapsulation polymorPhism = new Encapsulation();

polymorPhism.Addition();

polymorPhism.Addition(30, 30);

//polymorPhism.Addition();

}

public void Addition()

{

int num1 = 40, num2 = 40;

Console.WriteLine($"Addition is For Non Parameter Method: {num1 + num2}");

}

public void Addition(int num1, int num2)

{

Console.WriteLine($"Addition is For Parameter Int Method: {num1 + num2}");

}

//public int Addition(int num1,int num2) Error

//{

// int sum = num1 + num2;

// return sum;

//}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Inheritance \*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Parent\_Inheritance

{

public string name = "Imran";

public int age = 88;

public int workingHours = 12;

int salary = 450000;

public void Parent\_InheritanceMethod()

{

Console.WriteLine($"The Person Detail is:\*\*\*\*\*\*\*\*\*\*\nName: {name}\nAge: {age}\nSalary: {salary}\nWork Hour: {workingHours}");

}

}

class Single\_Inheritance : Parent\_Inheritance // Single

{

long contactNo = 9988776655;

string address = "Mumbai Pune Highway...";

public void Single\_InheritanceMethod()

{

Single\_Inheritance baseInherit = new Single\_Inheritance();

baseInherit.Parent\_InheritanceMethod();

Console.Write($"Contat Number: {contactNo}\nAddress: {address}\n");

}

}

class Hirarchical\_Inheritance : Parent\_Inheritance // Hirarchical Inheritance

{

string[] city = { "Pune", "Latur", "Mumbai", "Dubai", "Jermany", "America", "Nashik", "Amrawati", "London", "UK", "US", "Japan", };

public void Hirarchical\_InheritanceMethod()

{

Console.Write("\nCities Are: ");

foreach(string cities in city)

Console.Write(cities);

}

}

class Multilevel\_Inheritance : Hirarchical\_Inheritance // Multilevel

{

int[] number = { 0, 11, 110, 1100, 11000, 1100000 };

public void Multilevel\_InheritanceMethod()

{

Console.Write("\nNumbers Are: ");

for(int i=0;i<number.Length;i++)

Console.Write(i);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*Multiple Inheritance \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface MultipleInheritance

{

void Multiple\_InheritanceAccess();

}

class Multiple\_Inheritance : Multilevel\_Inheritance, MultipleInheritance

{

public void Multiple\_InheritanceAccess()

{

int no1 = 20, no2 = 40;

Console.WriteLine($"Addition is :{no1+no2}\nSubstraction is: {no2-no1}");

}

}

}

OopsSessions.cs

using OOPS\_\_AllSession;

using System;

using static OOPS\_\_AllSession.S11\_\_ClassAndTypes;

namespace Oops\_\_AllSession

{

class OopsSessions

{

static void Main(string[] args)

{

//S12\_\_FourPilllars(Abstraction, Encapsulation, Polymorphism, Inheritance)

S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc fourPillars = new S12\_\_FourPillars\_\_AbsEncapPolyMorpInhrtnc();

fourPillars.NonAbstraction\_Method();

Encapsulation polymorPhism = new Encapsulation();

//polymorPhism.PolyMorphism();

Single\_Inheritance childClass = new Single\_Inheritance();

//childClass.Single\_InheritanceMethod();

Multiple\_Inheritance multiple\_Inheritance = new Multiple\_Inheritance();

//multiple\_Inheritance.Parent\_InheritanceMethod();

}

}

}