Module 3

What to learn

Object Oriented Programming
Inheritance
Polymorphism
Encapsulation
Abstraction

Practice Exercise

Practice 1
Do the hands on the things provided in video and url https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/intro-to-csharp/object-oriented-programming
Practice 2

Inheritance

Single Inheritance:
Create a base class Vehicle with properties Make and Model. Derive a class Car that adds a property FuelType. Write a program to initialize a Car object and display all its details.

Method Overriding:
In the Vehicle class, add a method DisplayInfo() that prints the Make and Model. Override this method in the Car class to include FuelType. Call both versions using a Car object.

Constructor Chaining:
Demonstrate constructor chaining between the Vehicle and Car classes. Pass values for Make, Model, and FuelType while creating a Car object.

Multilevel Inheritance:
Add a new class ElectricCar derived from Car, with a property BatteryCapacity. Demonstrate the initialization and display of all details across three levels of inheritance.

Real-World Scenario:
Create a Person class with Name and Age. Derive a Student class that adds StudentID and Course. Derive a GraduateStudent class that includes GraduationYear. Implement a program to manage graduate student details.

Practice 3

Polymorphism

Compile-Time Polymorphism (Overloading):
Create a class MathOperations with overloaded methods Add that:

Accepts two integers.
Accepts three integers.

Accepts two doubles.
Test all overloads in a program.

Run-Time Polymorphism (Virtual Methods):
Create a base class Shape with a virtual method CalculateArea(). Derive Circle and Rectangle classes that override CalculateArea(). Use polymorphism to calculate areas for different shapes.

Abstract Class with Polymorphism:
Create an abstract class Employee with an abstract method CalculateSalary(). Derive classes FullTimeEmployee and PartTimeEmployee that implement this method. Use polymorphism to calculate salaries for both types.

Interface-Based Polymorphism:
Define an interface IDiscount with a method ApplyDiscount(). Implement this interface in RegularCustomer and PremiumCustomer classes with different discount rates. Demonstrate polymorphism by applying discounts to both customer types.

Real-World Scenario:
Create a Payment class with a virtual method ProcessPayment(). Derive CreditCardPayment and UPIPayment classes that override this method. Use a List<Payment> to process multiple types of payments in a single program.

Practice 4

Encapsulation

Encapsulation with Properties:
Write a class BankAccount with private fields AccountNumber, Balance, and AccountHolder. Use properties to encapsulate these fields and add validation for setting Balance.

Encapsulation with Business Logic:
Add a method Deposit and Withdraw to the BankAccount class. Ensure the Withdraw method prevents overdrafts and updates the balance.

Read-Only Properties:
Create a Product class with a read-only property DiscountedPrice that is calculated as Price - Discount. Ensure Discount is validated using encapsulation.

Encapsulation with Method Calls:
Write a class Order that contains private fields for OrderID, Items, and TotalAmount. Use encapsulation to manage adding items and updating the total amount.

Real-World Scenario:
Create a Student class with private fields for StudentID, Name, Marks. Use methods to add marks for subjects and calculate the percentage. Ensure marks cannot exceed a specified range.

Practice 5

Abstraction

Abstract Class:
Create an abstract class Appliance with abstract methods TurnOn() and TurnOff(). Derive Fan and WashingMachine classes that implement these methods.

Interface Implementation:
Define an interface IReport with a method GenerateReport(). Implement it in PDFReport and

ExcelReport classes. Demonstrate abstraction by using only the interface reference to generate reports.

Abstract Class with Business Logic:
Write an abstract class Shape with a method CalculatePerimeter() and an abstract method CalculateArea(). Implement it in Square and Triangle classes. Demonstrate abstraction by calculating areas and perimeters for both shapes.

Real-World Scenario with Abstraction:
Create an abstract class Loan with properties LoanAmount and InterestRate. Add an abstract method CalculateEMI(). Implement it in HomeLoan and CarLoan classes, each with their specific EMI formula.

Combining Interface and Abstract Class:
Create an abstract class Vehicle with a property Speed and an abstract method DisplayDetails(). Create an interface IFuelEfficiency with a method CalculateEfficiency(). Implement both in a HybridCar class.

Assignment Exercise

Assignment 1
Implement all the oops concept for Employee payroll system.
Assignment 2
Create above assignment's class hierarchy, implement inheritance and polymorphism.
Assignment 3

Assignment: Inventory Management System

Objective:

Build a **C# Console Application** for an **Inventory Management System** that demonstrates **Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism**. The system should manage different types of products, calculate inventory value, and handle business-specific logic like discounts and tax calculations.

Requirements:

1. Encapsulation

Encapsulate fields such as ProductID, Name, Price, and Stock within a base class Product using properties.
Add validation for Price and Stock to ensure they are non-negative.

2. Abstraction

Create an **abstract class** Product with:
- Abstract method CalculateTax() to calculate tax based on product type.
- Abstract method ApplyDiscount() for different discount rules.
- Concrete method CalculateTotalValue() to calculate the total value of stock (Price * Stock).

3. Inheritance

Derive the following classes from Product:
- Electronics

Specific tax calculation: Tax = 18% of Price.

Discount: 10% discount on products priced above 5000.

Groceries

Specific tax calculation: Tax = 5% of Price.

Discount: No discount for groceries.

Clothing

Specific tax calculation: Tax = 12% of Price.

Discount: 15% discount on products priced above 2000.

4. Polymorphism

Use polymorphism to calculate and display tax, discount, and inventory value for different product types using a list of Product references.

Online Reference

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abs

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/poly

.NET Core Web API

WEB API (old)

Authentication And Authorization (WEBAPI)(old)

FullStackDevelopment_WIth_Dotnet_AND_Angular