1. Problem: Payment Processing System

Imagine you are developing a payment processing system for an e-commerce application. The system should be able to process different types of payments, such as Credit Card, PayPal, and UPI.

Each payment method has a different way of processing, but they all follow a common ProcessPayment() method.

Requirements:

1. Define a base class Payment with a ProcessPayment() method.
2. Create derived classes (CreditCardPayment, PayPalPayment, UPIPayment) that override ProcessPayment().
3. Implement polymorphism by calling the overridden methods dynamically.

2. Problem: Vehicle Rental System

You are developing a Vehicle Rental System where different types of vehicles can be rented, such as Cars, Bikes, and Trucks.

Each vehicle has common attributes like Brand, Model, and Rental Price Per Day. However, different vehicles have specific rental policies (e.g., Car may require a minimum age limit, Truck may require a commercial license).

Requirements:

1. Use Abstraction (abstract class Vehicle) to define common properties and an abstract method CalculateRentalCost().
2. Use Inheritance to create Car, Bike, and Truck classes that extend Vehicle.
3. Each derived class implements its own rental cost calculation logic.
4. Demonstrate dynamic method dispatch using polymorphism.

3. Problem: Employee Payroll System

A company has an Employee Payroll System where base class Employee has a method ShowDetails(), but in the derived class Manager, the same method is redefined (hidden) using the new keyword to display additional details.

Scenario:

- The base class Employee shows basic employee details.
- The derived class Manager hides the ShowDetails() method and provides additional information such as Bonus.
- If accessed through a base class reference, it calls the base class method (method hiding).

4. Problem: Online Banking System

A bank wants to develop an **Online Banking System** where customers can:

1. **Create an account** (Savings or Current).
2. **Deposit and withdraw money** with specific rules.
3. **Check account details.**
4. **Ensure secure transactions** using **Encapsulation**.
5. **Use Inheritance** to extend features for different account types.
6. **Use Polymorphism** to handle different types of accounts dynamically.