Core OOP Principles – Abstraction & Interface

**Practice Problems (Any Three) (1;3;4 done)**

# Problem 1: Vehicle with Abstract Class

## Problem Statement:

Create an abstract class Vehicle with an abstract method start(). Subclasses Car

and Bike will extend Vehicle and provide their own implementations for start(). Demonstrate abstraction by using Vehicle references to call the methods.

// File: Vehicle.java

public abstract class Vehicle {

// TODO: Create abstract method start() public abstract void start();

// TODO: Create non-abstract method fuelType() -> print "Uses fuel"

public void fuelType() {

// ... (implementation)

}

}

// File: Car.java

public class Car extends Vehicle {

// TODO: Implement start() -> "Car starts with key" @Override

public void start() {

// ... (implementation)

}

}

// File: Bike.java

public class Bike extends Vehicle {

// TODO: Implement start() -> "Bike starts with kick" @Override

public void start() {

// ... (implementation)

}

}

// File: VehicleTest.java public class VehicleTest {

public static void main(String[] args) {

// TODO: Create Vehicle reference pointing to Car

// TODO: Call start() and fuelType()

// TODO: Create Vehicle reference pointing to Bike

// TODO: Call start() and fuelType()

}

}

Sol:

// Vehicle.java

abstract class Vehicle {

public abstract void start();

public void fuelType() {

System.out.println("Uses fuel");

}

}

// Car.java

class Car extends Vehicle {

@Override

public void start() {

System.out.println("Car starts with key");

}

}

// Bike.java

class Bike extends Vehicle {

@Override

public void start() {

System.out.println("Bike starts with kick");

}

}

// VehicleTest.java

public class VehicleTest {

public static void main(String[] args) {

Vehicle v1 = new Car();

v1.start();

v1.fuelType();

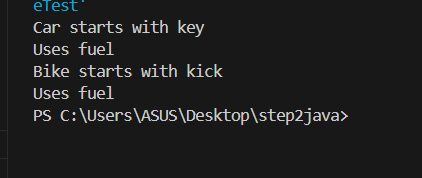
Vehicle v2 = new Bike();

v2.start();

v2.fuelType();

}

}



# Problem 2: Bank Account with Abstract Methods

## Problem Statement:

### Design an abstract class BankAccount with abstract method calculateInterest(). Subclasses SavingsAccount and CurrentAccount should implement it differently. Demonstrate abstraction by handling different account types.

**Understanding:** Abstract class with both abstract and non-abstract methods.

// File: BankAccount.java

public abstract class BankAccount { protected double balance;

// TODO: Create constructor to set balance public BankAccount(double balance) {

// ... (implementation)

}

// TODO: Create abstract method calculateInterest() public abstract void calculateInterest();



// TODO: Create non-abstract method displayBalance() -> print balance public void displayBalance() {

// ... (implementation)

}

}

// File: SavingsAccount.java

public class SavingsAccount extends BankAccount { public SavingsAccount(double balance) {

super(balance);

}

// TODO: Implement calculateInterest() -> interest = balance \* 0.04 @Override

public void calculateInterest() {

// ... (implementation)

}

}

// File: CurrentAccount.java

public class CurrentAccount extends BankAccount { public CurrentAccount(double balance) {

super(balance);

}

// TODO: Implement calculateInterest() -> interest = balance \* 0.02 @Override

public void calculateInterest() {

// ... (implementation)

}

}

// File: BankTest.java public class BankTest {

public static void main(String[] args) {

// TODO: Create BankAccount reference -> SavingsAccount

// TODO: Call displayBalance() and calculateInterest()

// TODO: Create BankAccount reference -> CurrentAccount

// TODO: Call displayBalance() and calculateInterest()

}

}

# Problem 3: Interface for Payment Gateway

## Problem Statement:

### Create an interface PaymentGateway with methods pay() and refund(). Implement this interface in CreditCardPayment and UPIPayment. Demonstrate multiple payment methods using interfaces.

**Understanding:** Interface implementation and abstraction through contracts.

// File: PaymentGateway.java public interface PaymentGateway {

// TODO: Declare method pay(double amount) void pay(double amount);

// TODO: Declare method refund(double amount) void refund(double amount);

}

// File: CreditCardPayment.java

public class CreditCardPayment implements PaymentGateway {

// TODO: Implement pay() -> "Paid via Credit Card" @Override

public void pay(double amount) {

// ... (implementation)

}

// TODO: Implement refund() -> "Refund to Credit Card" @Override

public void refund(double amount) {

// ... (implementation)

}

}

// File: UPIPayment.java

public class UPIPayment implements PaymentGateway {

// TODO: Implement pay() -> "Paid via UPI" @Override

public void pay(double amount) {

// ... (implementation)

}

// TODO: Implement refund() -> "Refund to UPI" @Override

public void refund(double amount) {

// ... (implementation)

}

}

// File: PaymentTest.java public class PaymentTest {

public static void main(String[] args) {

// TODO: Create PaymentGateway reference -> CreditCardPayment

// TODO: Call pay() and refund()

// TODO: Create PaymentGateway reference -> UPIPayment

// TODO: Call pay() and refund()

}

}

Sol:

// PaymentGateway.java

interface PaymentGateway {

void pay(double amount);

void refund(double amount);

}

// CreditCardPayment.java

class CreditCardPayment implements PaymentGateway {

@Override

public void pay(double amount) {

System.out.println("Paid " + amount + " via Credit Card");

}

@Override

public void refund(double amount) {

System.out.println("Refunded " + amount + " to Credit Card");

}

}

// UPIPayment.java

class UPIPayment implements PaymentGateway {

@Override

public void pay(double amount) {

System.out.println("Paid " + amount + " via UPI");

}

@Override

public void refund(double amount) {

System.out.println("Refunded " + amount + " to UPI");

}

}

// PaymentTest.java

public class PaymentTest {

public static void main(String[] args) {

PaymentGateway p1 = new CreditCardPayment();

p1.pay(500);

p1.refund(200);

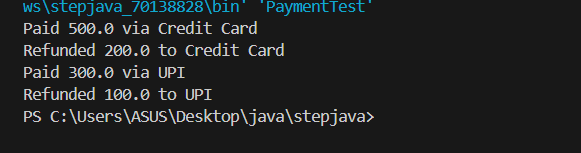
PaymentGateway p2 = new UPIPayment();

p2.pay(300);

p2.refund(100);

}

}



# Problem 4: Multiple Interfaces with Devices

## Problem Statement:

### Create two interfaces: Camera with method takePhoto() and MusicPlayer with method playMusic(). A class SmartPhone should implement both. Demonstrate multiple interface implementations.

**Understanding:** Multiple inheritance via interfaces.

// File: Camera.java public interface Camera {

// TODO: Declare method takePhoto() void takePhoto();

}

// File: MusicPlayer.java public interface MusicPlayer {

// TODO: Declare method playMusic() void playMusic();

}

// File: SmartPhone.java

public class SmartPhone implements Camera, MusicPlayer {

// TODO: Implement takePhoto() -> "Taking photo with smartphone" @Override

public void takePhoto() {

// ... (implementation)

}

// TODO: Implement playMusic() -> "Playing music on smartphone" @Override

public void playMusic() {

// ... (implementation)

}

}

// File: DeviceTest.java public class DeviceTest {

public static void main(String[] args) {

// TODO: Create SmartPhone object

// TODO: Call takePhoto()

// TODO: Call playMusic()

}

}

Sol:

// Camera.java

interface Camera {

void takePhoto();

}

// MusicPlayer.java

interface MusicPlayer {

void playMusic();

}

// SmartPhone.java

class SmartPhone implements Camera, MusicPlayer {

@Override

public void takePhoto() {

System.out.println("Taking photo with smartphone");

}

@Override

public void playMusic() {

System.out.println("Playing music on smartphone");

}

}

// DeviceTest.java

public class DeviceTest {

public static void main(String[] args) {

SmartPhone phone = new SmartPhone();

phone.takePhoto();

phone.playMusic();

}

}

A screen shot of a computer

AI-generated content may be incorrect.

# Problem 5: Abstract Employee Class with Bonus Calculation

## Problem Statement:

Create an abstract class Employee with data members name and salary. Add an abstract method calculateBonus(). Subclasses Manager and Developer should

implement the method differently. Demonstrate abstraction with real-world employee roles.

**Understanding:** Abstract class, common data members, constructor, and abstract method implementation.

// File: Employee.java

public abstract class Employee { protected String name; protected double salary;

// TODO: Constructor to initialize values public Employee(String name, double salary) {

// ... (implementation)

}

// TODO: Abstract method calculateBonus() public abstract double calculateBonus();

// TODO: Non-abstract method displayDetails() public void displayDetails() {

// ... (implementation)

}

}

// File: Manager.java

public class Manager extends Employee {

public Manager(String name, double salary) { super(name, salary);

}

// TODO: Implement calculateBonus() -> salary \* 0.20 @Override

public double calculateBonus() {

// ... (implementation)

}

}

// File: Developer.java

public class Developer extends Employee {

public Developer(String name, double salary) { super(name, salary);

}

// TODO: Implement calculateBonus() -> salary \* 0.10 @Override

public double calculateBonus() {

// ... (implementation)

}

}

// File: EmployeeTest.java public class EmployeeTest {

public static void main(String[] args) {

// TODO: Create Employee reference -> Manager("Alice", 50000)

// TODO: Create Employee reference -> Developer("Bob", 40000)

// TODO: Call displayDetails() and calculateBonus() for both

}

}