

**Core OOP Principles – Abstraction & Interface Practice Problems (Any Three)**

**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.

PROGRAM:

// File: Problem1.java

abstract class Vehicle {

public abstract void start(); // abstract method

public void fuelType() { // non-abstract method

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

}

}

class Car extends Vehicle {

@Override

public void start() {

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

}

}

class Bike extends Vehicle {

@Override

public void start() {

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

}

}

public class Problem1 {

public static void main(String[] args) {

Vehicle v1 = new Car(); // Vehicle reference to Car

v1.start();

v1.fuelType();

Vehicle v2 = new Bike(); // Vehicle reference to Bike

v2.start();

v2.fuelType();

}

}

OUTPUT:

Car starts with key

Uses fuel

Bike starts with kick

Uses fuel

**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.

PROGRAM:

// File: Problem2.java

abstract class BankAccount {

protected double balance;

public BankAccount(double balance) {

this.balance = balance;

}

public abstract void calculateInterest();

public void displayBalance() {

System.out.println("Balance: " + balance);

}

}

class SavingsAccount extends BankAccount {

public SavingsAccount(double balance) {

super(balance);

}

@Override

public void calculateInterest() {

double interest = balance \* 0.04;

System.out.println("Savings Account Interest: " + interest);

}

}

class CurrentAccount extends BankAccount {

public CurrentAccount(double balance) {

super(balance);

}

@Override

public void calculateInterest() {

double interest = balance \* 0.02;

System.out.println("Current Account Interest: " + interest);

}

}

public class Problem2 {

public static void main(String[] args) {

BankAccount acc1 = new SavingsAccount(10000);

acc1.displayBalance();

acc1.calculateInterest();

BankAccount acc2 = new CurrentAccount(20000);

acc2.displayBalance();

acc2.calculateInterest();

}

}

OUTPUT:

Balance: 10000.0

Savings Account Interest: 400.0

Balance: 20000.0

Current Account Interest: 400.0

**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.

PROGRAM:

// File: Problem3.java

interface PaymentGateway {

void pay(double amount);

void refund(double amount);

}

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");

}

}

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");

}

}

public class Problem3 {

public static void main(String[] args) {

PaymentGateway p1 = new CreditCardPayment();

p1.pay(500);

p1.refund(200);

PaymentGateway p2 = new UPIPayment();

p2.pay(1000);

p2.refund(500);

}

}

OUTPUT:

Paid 500.0 via Credit Card

Refunded 200.0 to Credit Card

Paid 1000.0 via UPI

Refunded 500.0 to UPI

**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();

}

5

// 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()

}

}

**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

6



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 {

7



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 }

}

8