Hands-on No. : **3a**

Topic : **OOPs- Inhertiance,Polymorphism,Abstraction**

Date : **04.08.2025**

Solve the following problems

Question No.	Question Detail

1	Create a class hierarchy to represent different types of vehicles. Implement a common method start() in each vehicle type to show specific behavior.
	Requirements:
	• Create a base class Vehicle with a method void start().
	• Derive the following classes from Vehicle:
	  ○ Car
	  ○ Bike
	  ○ Truck
	• Override the start() method in each derived class to display a unique message.
	• In the main program, create an array of Vehicle references pointing to objects of different types and call start() on each.
	Sample Output
	Car is starting with a key.
	Bike is starting with a kick.
	Truck is starting with a button.

2	Design an abstract class Employee that enforces subclasses to implement their own salary calculation.
	Requirements:
	• Create an abstract class Employee with:
	  ○ Attributes: name, empId
	  ○ Abstract method: double calculateSalary()
	• Create the following concrete subclasses:
	  ○ FullTimeEmployee: Has a fixed monthly salary.
	  ○ PartTimeEmployee: Paid hourly. Attributes include hoursWorked and ratePerHour.
	• In the main program, create objects of each subclass and display their name and calculated salary using calculateSalary().

It is going to be hard but, hard does not mean impossible.

3	Create a Payment interface that defines a contract for processing different payment modes.
	Requirements:
	• Create an interface Payment with method:
	  ○ void pay(double amount)
	• Implement the interface in the following classes:
	  ○ CreditCardPayment
	  ○ UPIPayment
	  ○ WalletPayment
	• Each implementation should display a specific message indicating the payment method and amount.
	• In the main program, use the interface reference to invoke pay() on different payment types.
	Sample Input and Output:
	=== Welcome to the Payment System ===
	Enter amount to pay: ₹1000
	
	Select Payment Method:
	1. Credit Card
	2. UPI
	3. Wallet
	Enter your choice (1-3): 1
	
	Paid ₹1000.0 via Credit Card.
	=== Welcome to the Payment System ===
	Enter amount to pay: ₹750
	
	Select Payment Method:
	1. Credit Card
	2. UPI
	3. Wallet
	Enter your choice (1-3): 2
	
	Paid ₹750.0 via UPI.
	=== Welcome to the Payment System ===
	Enter amount to pay: ₹500

	Select Payment Method: 1. Credit Card 2. UPI 3. Wallet Enter your choice (1-3): 3 Paid ₹500.0 via Wallet.
4	Create a base class Notification with a method void send(String message). Override it in: • EmailNotification • SMSNotification • PushNotification In the main class, use a Notification reference to send the same message using all three types. Sample Input and Output Sending Email: System maintenance at midnight. Sending SMS: System maintenance at midnight. Sending Push Notification: System maintenance at midnight.
5	Create an interface Taxable with method double calculateTax(). Implement it in: • SalaryIncome – tax = 10% of salary • BusinessIncome – tax = 20% of profit • FreelanceIncome – tax = 15% of income Let the user enter income details and display the tax owed for each type using the interface reference. OR Create a Java program to **calculate tax** for different types of income using **interfaces and polymorphism**. Requirements: 1. **Create an interface** named Taxable with a method: double calculateTax(); 2. **Implement this interface** in three classes: ○ SalaryIncome: Calculates **10% tax** on the salary. ○ BusinessIncome: Calculates **20% tax** on the profit. ○ FreelanceIncome: Calculates **15% tax** on the freelance income. 3. In the **main class (TaxCalculator)**: ○ Accept input from the user for **salary, business profit, and freelance income**.

	○ Use the Taxable interface reference to **call calculateTax()** for each type. ○ Display the tax amount for each income type. Sample Input and Output Enter salary income: 50000 Salary Tax: ₹5000.0 Enter business profit: 100000 Business Tax: ₹20000.0 Enter freelance income: 40000 Freelance Tax: ₹6000.0

6	Design a Java program that models the behavior of animals using the concepts of:
	• Abstract Classes
	• Interfaces with default methods and constants
	• Polymorphism
	Requirements:
	1. **Create an abstract class** Animal with the following:
	○ A **protected String variable** name to store the animal's name.
	○ A **constructor** that initializes the name.
	○ An **abstract method**: void eat();
	2. Create an interface Sound with:
	• An abstract method: void makeSound();
	• A constant variable:String CATEGORY = "Domestic Animal";
	• A Default method :
	default void info() {
	System.out.println("All animals make some sound.");
	}
	3. **Create two concrete classes**:
	• Dog
	• Cat
	Both classes should:
	• Extend the abstract class Animal
	• Implement the Sound interface
	• Provide concrete implementation of eat() and makeSound():
	○ Dog should display:
	▪ Eating: "Buddy eats bones."

- Sound: "Buddy says: Woof Woof!"
- Cat should display:
  - Eating: "Whiskers eats fish."
  - Sound: "Whiskers says: Meow Meow!"

- Use the default method info() from the Sound interface.
- Display the value of CATEGORY from the Sound interface.

4.**In the main method** (in AnimalTest class):

- Create objects of Dog and Cat using **polymorphic references**:
  - Use Animal reference for eat()
  - Use Sound reference for makeSound() and info()
- Display the behavior of each animal.

Sample Input and Output:

Buddy eats bones.
Buddy says: Woof Woof!
All animals make some sound.
Category: Domestic Animal

Whiskers eats fish.
Whiskers says: Meow Meow!
All animals make some sound.
Category: Domestic Animal