Hands-on No. : 7

Topic : OOPs- Inheritance, polymorphism, Abstraction, Interface

Date : 15-05-2024

Solve the following problems

Question No.	Question Detail	Level
1	Suppose you are working on a project for a car dealership that wants to keep track of their inventory. They have various types of vehicles in their inventory, including cars, trucks, and motorcycles. To manage this inventory efficiently, you are tasked with creating a program using Java that utilizes single inheritance. Design a Java program to help the car dealership manage their inventory. Your program should include the following classes: 1. **Vehicle** class: • Attributes: • **brand** (String): to store the brand name of the vehicle. • **year** (int): to store the manufacturing year of the vehicle. • Methods: • **displayInfo()**: a method to display the brand name and manufacturing year of the vehicle. 2. **Car** class (subclass of **Vehicle**): • Additional Attributes: • **model** (String): to store the model of the car. • Additional Method: • **displayCarInfo()**: a method to display the brand name, model, and manufacturing year of the car.	Easy

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

	Your program should allow the user to create an instance of a car, set its attributes (brand, year, model), and then display information about the car using both **displayInfo()** and **displayCarInfo()** methods.	
2	You're tasked with developing a program for a company to manage information about its employees and calculate their salaries. The company has different types of employees: full-time employees, part-time employees, and contract employees. Each type of employee has specific attributes and salary calcu Design a Java program to assist the company in managing employee information and calculating their salaries. Your program should include the following classes: 1. **Employee** class: • Attributes: • **employeeID** (int): to store the employee ID. • **name** (String): to store the name of the employee. • Methods: • **displayInfo()**: a method to display the employee ID and name. • **calculateSalary()**: an abstract method to calculate the salary of the employee. 2. **FullTimeEmployee** class (subclass of **Employee**): • Additional Attribute: • **salary** (double): to store the monthly salary of the full-time employee. • Methods: • **calculateSalary()**: implements the method to calculate the salary based on the formula: salary = monthly salary. 3. **PartTimeEmployee** class (subclass of **Employee**): • Additional Attributes:	Medium

	• **hourlyRate** (double): to store the hourly rate of the part-time employee. • **hoursWorked** (int): to store the number of hours worked by the part-time employee. • Methods: • **calculateSalary()**: implements the method to calculate the salary based on the formula: salary = hourly rate * hours worked. 4. **ContractEmployee** class (subclass of **Employee**): • Additional Attribute: • **contractDuration** (int): to store the duration of the contract in months. • Methods: • **calculateSalary()**: implements the method to calculate the salary based on the formula: salary = contract duration * fixed monthly payment. Your program should allow the user to create instances of full-time, part-time, and contract employees, set their attributes, and then calculate and display their salaries using the appropriate methods (**displayInfo()** and **calculateSalary()**). Provide the Java implementation of the above scenario, ensuring proper hierarchical inheritance between the **Employee**, **FullTimeEmployee**, **PartTimeEmployee**, and **ContractEmployee** classes.	
3	You are building a restaurant ordering system. Create an abstract class called MenuItem with properties for Name (string) and Price (decimal). Implement an abstract method named Cook() that simulates cooking the menu item. Derive two classes from MenuItem called Burger and Pizza. Implement the Cook() method in both derived classes to display the name and cooking instructions for the menu item.	Easy

	Write a program that creates instances of Burger and Pizza and calls their Cook() methods.	
4	You are building a restaurant ordering system. Create an abstract class called MenuItem with properties for Name (string) and Price (decimal). Implement an abstract method named Cook() that simulates cooking the menu item. Derive two classes from MenuItem called Burger and Pizza. Implement the Cook() method in both derived classes to display the name and cooking instructions for the menu item. Write a program that creates instances of Burger and Pizza and calls their Cook() methods.	Easy
5	You are building an e-commerce platform. Create an abstract class called Product with properties for Name (string) and Price (decimal). Implement an abstract method named AddToCart() that adds the product to the user's shopping cart. Additionally, implement a non-abstract method named GetDiscountedPrice() that calculates and returns the discounted price for the product based on certain conditions. Derive two classes from Product called Book and Electronics. Implement the AddToCart() method in both derived classes to add the respective products to the shopping cart. Implement the GetDiscountedPrice() method to apply specific discount calculations for books and electronics. Write a program that creates instances of Book and Electronics and calls their AddToCart() and GetDiscountedPrice() methods.	Easy
6	You are building a shape calculator application. Create an interface called IShape with the following methods: CalculateArea(): This method should calculate and return the area of the shape. CalculatePerimeter(): This method should calculate and return the perimeter of the shape. Write two classes, Rectangle and Circle, that implement the IShape interface. In each class, implement the methods to calculate and return the	Easy

	area and perimeter of the shape. Write a program that creates instances of both Rectangle and Circle, sets their dimensions, and calls their respective methods to calculate the area and perimeter.	
7	Write a JAVA program that uses method overloading to perform different operations on a string. Implement a method called ProcessString that can perform different operations like converting the string to uppercase, reversing the string, and calculating the length of the string. The method should be overloaded to accept different numbers and types of parameters.	Easy
8	Write a JAVA program that demonstrates method overloading by creating multiple methods with the same name but different numbers of parameters. Implement a method called CalculateArea that can calculate the area of a rectangle, a circle, and a triangle. The method should be overloaded to accept different numbers and types of parameters.	Medium
9	Create a JAVA program that uses method overloading to perform different operations on an array of integers. Implement a method called ProcessArray that can perform different operations like finding the sum of the array elements, finding the maximum value in the array, and calculating the average of the array elements. The method should be overloaded to accept different numbers and types of parameters.	Easy
10	Create a base class called Shape with a virtual method named CalculateArea() that returns a double value. Derive two classes from Shape called Rectangle and Circle. Override the CalculateArea() method in both derived classes to calculate and return the area of a rectangle and a circle, respectively. Finally, write a program that creates instances of Rectangle and Circle and calls their CalculateArea() methods.	Easy

11	You are tasked with designing a Java program to model different types of vehicles with various features. Each vehicle can have common functionality such as starting and stopping, as well as specific features like electric or gas-powered engines. Design a program using interfaces to represent these features and implement classes for specific types of vehicles. Define an interface named **Vehicle** with two methods: • **start()**: a method to start the vehicle. • **stop()**: a method to stop the vehicle. 2. Define two additional interfaces: • **ElectricVehicle** with a method **charge()** for vehicles with electric engines. • **GasVehicle** with a method **refuel()** for vehicles with gas engines. 3. Implement classes for specific types of vehicles: • Implement a class named **ElectricCar** for electric cars. This class should implement the **Vehicle** and **ElectricVehicle** interfaces and provide implementations for all required methods. • Implement a class named **GasMotorcycle** for gas-powered motorcycles. This class should implement the **Vehicle** and **GasVehicle** interfaces and provide implementations for all required methods. 4. In the **Main** class, create instances of **ElectricCar** and **GasMotorcycle**, and test their functionality by calling their methods to start, stop, charge, and refuel. 5. Ensure that each class provides meaningful output messages when their methods are called to demonstrate the functionality of the vehicles.	Medium