Practice No. : **7**

Topic : **OOPs- Inheritance, Polymorphism, Abstraction, Interface**

Date : **15-05-2024**

Solve the following problems

Question No.	Question Detail	Level

1	Define a class called "BankAccount" with attributes "accountNumber," "balance," and "interestRate," along with a method called "deposit()" to add funds to the account.Create a subclass called "SavingsAccount" that extends "BankAccount" and includes an additional attribute called "minimumBalance" and a method called "withdraw()" to subtract funds from the account.Further extend the hierarchy with a subclass called "FixedDepositAccount," which inherits from "SavingsAccount" and includes a new attribute called "term." Additionally, implement a method called "getInterest()" to calculate and return the interest earned on the account.
	Program:
	// BankAccount.java
	public class BankAccount {
	    protected String accountNumber;
	    protected double balance;
	    protected double interestRate;
	    public BankAccount(String accountNumber, double balance, double interestRate) {
	        this.accountNumber = accountNumber;
	        this.balance = balance;
	        this.interestRate = interestRate;
	    }
		Medium

```
    public void deposit(double amount) {
        balance += amount;
        System.out.println(amount + " deposited successfully.");
    }
}
// SavingsAccount.java
public class SavingsAccount extends BankAccount {
    private double minimumBalance;

    public SavingsAccount(String accountNumber, double balance,
double interestRate, double minimumBalance) {
        super(accountNumber, balance, interestRate);
        this.minimumBalance = minimumBalance;
    }

    public void withdraw(double amount) {
        if (balance - amount >= minimumBalance) {
            balance -= amount;
            System.out.println(amount + " withdrawn successfully.");
        } else {
            System.out.println("Insufficient funds.");
        }
    }
}
// FixedDepositAccount.java
public class FixedDepositAccount extends SavingsAccount {
    private int term;
    public FixedDepositAccount(String accountNumber, double
balance, double interestRate, double minimumBalance, int term) {
        super(accountNumber, balance, interestRate,
minimumBalance);
        this.term = term;
    }
    public double getInterest() {
        // Calculate interest based on the balance, interest rate, and
term
```

	double interest = balance * interestRate * term / 100;
	return interest;
	}
	}	

2	You are tasked with implementing method overloading in the **Employee** class, which models an employee in an organization. The **Employee** class already has attributes such as **ID**, **name**, and **salary**. You need to implement the following methods in the **Employee** class using method overloading:
	1. **calculateYearlySalary**: This method should calculate and return the employee's yearly salary based on their monthly salary. The method should accept the monthly salary as a parameter.
	2. **calculateYearlySalary:** This method should calculate and return the employee's yearly salary based on their daily salary. The method should accept the daily salary and number of days in a year as a parameter.
	3. **calculateYearlySalary:** This method should calculate and return the employee's yearly salary based on their hourly salary and the number of hours worked per day. The method should accept the hourly salary and hours worked per day as parameters.
	toString: This method should print the employee's ID, name, and salary.
	Program:
	public class Employee {
	    private int id;
	    private String name;
	    private double salary;
	
	    // Constructor
	    public Employee(int id, String name, double salary) {
	        this.id = id;
	        this.name = name;
	        this.salary = salary;
	    }
		Medium

	// Method overloading to calculate yearly salary based on monthly salary
	public double calculateYearlySalary(double monthlySalary) {
	return monthlySalary * 12;
	}
	
	// Method overloading to calculate yearly salary based on daily salary and number of days
	public double calculateYearlySalary(double dailySalary, int daysInYear) {
	return dailySalary * daysInYear;
	}
	// Method overloading to calculate yearly salary based on hourly salary and hours worked per day
	public double calculateYearlySalary(double hourlySalary, int hoursPerDay) {
	// Assuming 5 working days per week and 52 weeks in a year
	return hourlySalary * hoursPerDay * 5 * 52;
	}
	// toString method to print employee information
	@Override
	public String toString() {
	return "Employee ID: " + id + ", Name: " + name + ", Salary: " + salary;
	}
	}	

3	Design a Java program to model bank accounts, including standard accounts, savings accounts, and current accounts, using inheritance. The program should allow users to perform basic banking operations like deposit, withdrawal, and display account information. The savings accounts should have an additional feature of earning interest on the account balance, and the current accounts should have an overdraft limit to prevent overdrawing. 1. Create a class called "Account" with the following attributes and methods: **Attributes:** • accountNumber (String) • accountName(String)	Medium

- balance (double)

Methods:

- Account constructor: Accepts an account number and initial balance as parameters and initializes the account.
- deposit(double amount): Adds the given amount to the account balance.
- withdraw(double amount): Subtracts the given amount from the account balance, ensuring the balance does not go negative.
- getAccountNumber(): Returns the account number.
- getAccountName(): Returns the account holder name.
- getBalance(): Returns the current account balance.
- displayInfo(): Displays the account number and current balance.

2. Create a class called **"SavingsAccount"** that inherits from the "Account" class. The **"SavingsAccount"** should have an additional attribute and method:

   Additional Attribute:

   - interestRate (double)

   Additional Method:

   - addInterest(): Adds interest to the account balance based on the interest rate.

3. Create a class called **"CurrentAccount"** that inherits from the "Account" class. The **"CurrentAccount"** should have an additional attribute and method:

   Additional Attribute:

   - overdraftLimit (double)

   Additional Method:

   - getOverdraftLimit(): Returns the overdraft limit for the current account.

4. In the "SavingsAccount" class, override the "displayInfo()" method from the base class to include the interest rate in the account information display.
5. In the "CurrentAccount" class, override the "displayInfo()" method from the base class to include the overdraft limit in the account information display.

6. In a separate class, create a main method to demonstrate the functionality of the "Account", "SavingsAccount", and "CurrentAccount" classes:
    - Create a standard account, a savings account, and a current account with initial balances and specific interest rates, and overdraft limits.
    - Perform deposit and withdrawal operations on all accounts.
    - Display the account information after each transaction, including the interest rate for the savings account and the overdraft limit for the current account.
    - For the savings account, add interest and display the updated balance.
    - Ensure that the program correctly handles withdrawals that exceed the account balance and overdraft limit.

Program:

```
// Account.java
public class Account {
    protected String accountNumber;
    protected String accountName;
    protected double balance;

    public Account(String accountNumber, String accountName, double balance) {
        this.accountNumber = accountNumber;
        this.accountName = accountName;
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (balance - amount >= 0) {
            balance -= amount;
        } else {
            System.out.println("Insufficient funds.");
        }
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public String getAccountName() {
        return accountName;
    }

    public double getBalance() {
        return balance;
    }

    public void displayInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Account Holder: " + accountName);
        System.out.println("Balance: " + balance);
    }
}
// SavingsAccount.java
public class SavingsAccount extends Account {
    private double interestRate;

    public SavingsAccount(String accountNumber, String
accountName, double balance, double interestRate) {
        super(accountNumber, accountName, balance);
        this.interestRate = interestRate;
    }

    public void addInterest() {
        double interest = balance * interestRate / 100;
        deposit(interest);
    }
    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Interest Rate: " + interestRate + "%");
    }
}
// CurrentAccount.java
public class CurrentAccount extends Account {
    private double overdraftLimit;

    public CurrentAccount(String accountNumber, String
accountName, double balance, double overdraftLimit) {
        super(accountNumber, accountName, balance);
        this.overdraftLimit = overdraftLimit;
    }

    public double getOverdraftLimit() {
        return overdraftLimit;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Overdraft Limit: " + overdraftLimit);
    }
}
// Main.java
public class Main {
    public static void main(String[] args) {
        // Create accounts
        Account standardAccount = new Account("A123", "John Doe",
1000);
        SavingsAccount savingsAccount = new
SavingsAccount("S456", "Alice Smith", 2000, 2.5);
        CurrentAccount currentAccount = new CurrentAccount("C789",
"Bob Johnson", 1500, 500);
```

	// Perform transactions
	standardAccount.deposit(500);
	standardAccount.withdraw(200);
	savingsAccount.deposit(1000);
	savingsAccount.withdraw(300);
	currentAccount.deposit(700);
	currentAccount.withdraw(200);
	currentAccount.withdraw(2000); // Attempt to withdraw more than balance
	currentAccount.withdraw(1000); // Attempt to withdraw within overdraft limit
	
	// Display account information
	System.out.println("Standard Account Information:");
	standardAccount.displayInfo();
	System.out.println();
	
	System.out.println("Savings Account Information:");
	savingsAccount.displayInfo();
	System.out.println();
	
	System.out.println("Current Account Information:");
	currentAccount.displayInfo();
	System.out.println();
	
	// Add interest to savings account and display updated balance
	savingsAccount.addInterest();
	System.out.println("Updated Savings Account Balance after adding interest:");
	savingsAccount.displayInfo();
	}
	}	

4	Imagine you are developing a system for managing different types of shapes, such as circles, rectangles, and triangles. Describe how you would utilize abstract classes and methods to define a common interface for geometric shapes while allowing each shape to implement its own logic for calculating area and perimeter.	Medium

Program:

```
abstract class Shape {
    abstract double calculateArea();
    abstract double calculatePerimeter();
}
class Circle extends Shape {
    private double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

class Rectangle extends Shape {
    private double length;
    private double width;

    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    double calculateArea() {
        return length * width;
    }
    @Override
    double calculatePerimeter() {
        return 2 * (length + width);
    }
}

class Triangle extends Shape {
    private double side1;
    private double side2;
    private double side3;

    Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    @Override
    double calculateArea() {
        // Implement logic for calculating area of a triangle
        // using Heron's formula or other methods
        return 0.0;
    }

    @Override
    double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}
public class Main {
    public static void main(String[] args) {
        // Create instances of different shapes
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);
        Shape triangle = new Triangle(3, 4, 5);

        // Calculate and display area and perimeter of each shape
```

	System.out.println("Circle:");
	System.out.println("Area: " + circle.calculateArea());
	System.out.println("Perimeter: " + circle.calculatePerimeter());
	
	System.out.println("\nRectangle:");
	System.out.println("Area: " + rectangle.calculateArea());
	System.out.println("Perimeter: " + rectangle.calculatePerimeter());
	
	System.out.println("\nTriangle:");
	// Implement logic for calculating area of triangle
	// System.out.println("Area: " + triangle.calculateArea());
	System.out.println("Perimeter: " + triangle.calculatePerimeter());
	}
	}	

5	You are developing a library management system where users can search for books using different criteria such as title, author, or ISBN (International Standard Book Number). To enhance user experience, you decide to implement overloaded methods for searching books based on different criteria.
	
	Design and implement a Java class named "Library" that includes overloaded methods for searching books. Each method should take different parameters representing search criteria, such as title, author, or ISBN. Provide appropriate return types and handle cases where no matching books are found. Additionally, create a main method to demonstrate the usage of these overloaded search methods with sample input data.
	
	Program:
	import java.util.ArrayList;
	import java.util.List;
	
	class Book {
	private String title;
	private String author;
	private String isbn;
		Medium

```
    public Book(String title, String author, String isbn) {
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public String getIsbn() {
        return isbn;
    }
}

public class Library {
    private List<Book> books;

    public Library() {
        books = new ArrayList<>();
    }

    public void addBook(Book book) {
        books.add(book);
    }

    public List<Book> searchBooks(String title) {
        List<Book> result = new ArrayList<>();
        for (Book book : books) {
            if (book.getTitle().equalsIgnoreCase(title)) {
                result.add(book);
            }
        }
        return result;
    }

    public List<Book> searchBooks(String author) {
        List<Book> result = new ArrayList<>();
        for (Book book : books) {
            if (book.getAuthor().equalsIgnoreCase(author)) {
                result.add(book);
            }
        }
        return result;
    }

    public Book searchBook(String isbn) {
        for (Book book : books) {
            if (book.getIsbn().equals(isbn)) {
                return book;
            }
        }
        return null; // Book not found
    }

    public static void main(String[] args) {
        // Create a library
        Library library = new Library();

        // Add some books to the library
        library.addBook(new Book("Java Programming", "John Smith",
"1234567890"));
        library.addBook(new Book("Python Programming", "Alice
Johnson", "0987654321"));
        library.addBook(new Book("Data Structures", "Bob Brown",
"9876543210"));
        library.addBook(new Book("Algorithms", "Alice Johnson",
"5432109876"));
```

```
        // Search for books by title
        System.out.println("Books with title 'Java Programming':");
        List<Book> javaBooks = library.searchBooks("Java
Programming");
        if (javaBooks.isEmpty()) {
            System.out.println("No books found.");
        } else {
            for (Book book : javaBooks) {
                System.out.println(book.getTitle() + " by " +
book.getAuthor());
            }
        }

        // Search for books by author
        System.out.println("\nBooks by author 'Alice Johnson':");
        List<Book> aliceBooks = library.searchBooks("Alice Johnson");
        if (aliceBooks.isEmpty()) {
            System.out.println("No books found.");
        } else {
            for (Book book : aliceBooks) {
                System.out.println(book.getTitle() + " by " +
book.getAuthor());
            }
        }
        // Search for a book by ISBN
        System.out.println("\nBook with ISBN '9876543210':");
        Book book = library.searchBook("9876543210");
        if (book == null) {
            System.out.println("Book not found.");
        } else {
            System.out.println(book.getTitle() + " by " +
book.getAuthor());
        }
    }
}
```

6	You are developing a software application for a university that manages student information. As part of the system, you have a base	Medium

Sometimes later becomes never. DO IT NOW!

class named "Person" that represents common attributes and behaviors shared among different individuals, such as students, faculty, and staff. Each specific type of person (e.g., student, faculty) will have its own implementation of the "displayInfo()" method to provide details specific to that type.

Design and implement a Java class hierarchy for managing student information, starting with a base class named "Person" that includes a method named "displayInfo()". Then, create subclasses for representing different types of students, such as "UndergraduateStudent" and "GraduateStudent", and override the "displayInfo()" method in each subclass to display relevant information for that type of student. Provide sample data and demonstrate how the overridden methods are invoked using polymorphism.

Program:

```
// Person.java
public class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
// Student.java
public class Student extends Person {
    protected int studentId;

    public Student(String name, int age, int studentId) {
        super(name, age);
        this.studentId = studentId;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Student ID: " + studentId);
    }
}
// UndergraduateStudent.java
public class UndergraduateStudent extends Student {
    private String major;

    public UndergraduateStudent(String name, int age, int studentId,
String major) {
        super(name, age, studentId);
        this.major = major;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Major: " + major);
    }
}
// GraduateStudent.java
public class GraduateStudent extends Student {
    private String researchTopic;

    public GraduateStudent(String name, int age, int studentId, String
researchTopic) {
        super(name, age, studentId);
        this.researchTopic = researchTopic;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
```

	System.out.println("Research Topic: " + researchTopic);
	}
	}
	public class Main {
	public static void main(String[] args) {
	Person person = new Person("John Doe", 30);
	Student student1 = new Student("Alice Smith", 20, 123456);
	UndergraduateStudent undergrad = new UndergraduateStudent("Bob Johnson", 21, 654321, "Computer Science");
	GraduateStudent grad = new GraduateStudent("Emily Brown", 25, 987654, "Machine Learning");
	
	System.out.println("Displaying information for a Person:");
	person.displayInfo();
	System.out.println("\nDisplaying information for a Student:");
	student1.displayInfo();
	System.out.println("\nDisplaying information for an Undergraduate Student:");
	undergrad.displayInfo();
	System.out.println("\nDisplaying information for a Graduate Student:");
	grad.displayInfo();
	}
	}	

7	You have been hired as a software developer to create a system for managing different types of vehicles. The system should utilize the concepts of polymorphism and interfaces in Java. Your task is to design and implement the following: 1. Create an interface called **Vehicle** that declares the following methods: • **void start ()**: This method should start the vehicle. • **void stop ()**: This method should stop the vehicle. 2. Create three classes that implement the **Vehicle** interface:	Medium

- **Car**: This class represents a car and should provide an implementation for the **start()** and **stop()** methods specific to a car.
- **Motorcycle**: This class represents a motorcycle and should provide an implementation for the **start ()** and **stop()** methods specific to a motorcycle.
- **Truck**: This class represents a truck and should provide an implementation for the **start()** and **stop()** methods specific to a truck.

3. In the **main** method of your program, create an array of **Vehicle** objects. Include instances of cars, motorcycles, and trucks in the array.

4. Iterate over the array and call the **start()** method for each vehicle to start it.

5. Iterate over the array again and call the **stop()** method for each vehicle to stop it.

Your program should demonstrate the concept of polymorphism by treating each vehicle object as an instance of the **Vehicle** interface, allowing the same method calls (**start()** and **stop()**) to be made on different types of vehicles.

Ensure that your program is logically structured, follows naming conventions, and includes appropriate comments to explain the purpose of each class and method.

Note: Focus on implementing the polymorphism concept and interface methods. You do not need to implement additional functionality such as vehicle details or specific actions in the **start()** and **stop()** methods.

Program:

```
// Vehicle.java
public interface Vehicle {
   void start();
   void stop();
}
// Car.java
public class Car implements Vehicle {
   @Override
```

```
    public void start() {
        System.out.println("Car started.");
    }

    @Override
    public void stop() {
        System.out.println("Car stopped.");
    }
}
// Motorcycle.java
public class Motorcycle implements Vehicle {
    @Override
    public void start() {
        System.out.println("Motorcycle started.");
    }

    @Override
    public void stop() {
        System.out.println("Motorcycle stopped.");
    }
}
// Truck.java
public class Truck implements Vehicle {
    @Override
    public void start() {
        System.out.println("Truck started.");
    }

    @Override
    public void stop() {
        System.out.println("Truck stopped.");
    }
}
// Main.java
public class Main {
    public static void main(String[] args) {
        // Create an array of Vehicle objects
        Vehicle[] vehicles = new Vehicle[3];
        vehicles[0] = new Car();
        vehicles[1] = new Motorcycle();
        vehicles[2] = new Truck();

        // Iterate over the array and call start() method for each vehicle
        System.out.println("Starting vehicles:");
        for (Vehicle vehicle : vehicles) {
            vehicle.start();
        }

        // Iterate over the array again and call stop() method for each vehicle
        System.out.println("\nStopping vehicles:");
        for (Vehicle vehicle : vehicles) {
            vehicle.stop();
        }
    }
}
```