Self Practice - Day 06 - Inheritance, Polymorphism, Abstra

1. Question 1

Problem Statement:

Define a class called "BankAccount" with attributes `accountNumber`, `balance`, and `interestRate`, along with a method called `deposit()` to add funds to the account. `SavingsAccount` that extends `BankAccount` and includes an additional attribute called `minimumBalance` and a method called `withdraw()` to subtract funds from the a with a subclass called `FixedDepositAccount`, which inherits from `SavingsAccount` and includes a new attribute called `term`. Additionally, implement a method called the interest earned on the account.

Analysis:

1. Create a class `BankAccount` with attributes `accountNumber`, `balance`, and `interestRate`.
2. Create a method `deposit()` to add funds to the account.
3. Create a subclass `SavingsAccount` that extends `BankAccount` and includes an additional attribute `minimumBalance`.
4. Create a method `withdraw()` to subtract funds from the account.
5. Create a subclass `FixedDepositAccount` that inherits from `SavingsAccount` and includes a new attribute `term`.
6. Implement a method `getInterest()` to calculate and return the interest earned on the account.
7. Create objects of `BankAccount`, `SavingsAccount`, and `FixedDepositAccount` classes and demonstrate the working of the methods.

Code:

```
class BankAccount{
    int accountNumber;
    double balance;
    double interestRate;
    void deposit(double amount){
        balance += amount;
    }
}

class SavingsAccount extends BankAccount{
    double minimumBalance;
    void withdraw(double amount){
        if(balance - amount >= minimumBalance){
            balance -= amount;
        }
    }
}

class FixedDepositAccount extends SavingsAccount{
    int term;
    double getInterest(){
        return balance * interestRate * term;
    }
}

public class Question1{
    public static void main(String[] args){
        BankAccount b = new BankAccount();
        b.accountNumber = 123456;
        b.balance = 1000;
        b.interestRate = 0.05;
        b.deposit(500);
        System.out.println("Balance: " + b.balance);
```

```
        SavingsAccount s = new SavingsAccount();
        s.accountNumber = 123456;
        s.balance = 1000;
        s.interestRate = 0.05;
        s.minimumBalance = 500;
        s.withdraw(500);
        System.out.println("Balance: " + s.balance);

        FixedDepositAccount f = new FixedDepositAccount();
        f.accountNumber = 123456;
        f.balance = 1000;
        f.interestRate = 0.05;
        f.minimumBalance = 500;
        f.term = 5;
        System.out.println("Interest: " + f.getInterest());
    }
}
```

Output:

```
Balance: 1500.0
Balance: 500.0
Interest: 250.0
```

---

2. Question 2

Problem Statement:

You are tasked with implementing method overloading in the Employee class, which models an employee in an organization. The Employee class already has attribute

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 month
2. **calculateYearlySalary:** This method should calculate and return the employee's yearly salary based on their daily salary. The method should accept the daily salar 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 da salary and hours worked per day as parameters.

toString: This method should print the employee's ID, name, and salary

Analysis:

1. Create a class `Employee` with attributes `ID`, `name`, and `salary`.
2. Implement method overloading for the `calculateYearlySalary` method based on the monthly salary, daily salary, and hourly salary.
3. Implement a `toString` method to print the employee's details.
4. Create objects of the `Employee` class and demonstrate the working of the methods.

Code:

```
class Employee{
    int ID;
    String name;
    double salary;

    double calculateYearlySalary(double monthlySalary){
        return monthlySalary * 12;
    }
```

```
    double calculateYearlySalary(double dailySalary, int days){
        return dailySalary * days;
    }

    double calculateYearlySalary(double hourlySalary, int hours){
        return hourlySalary * hours * 365;
    }

    public String toString(){
        return "ID: " + ID + ", Name: " + name + ", Salary: " + salary;
    }
}

public class Question2{
    public static void main(String[] args){
        Employee e = new Employee();
        e.ID = 123;
        e.name = "John Doe";
        e.salary = 50000;
        System.out.println("Yearly Salary (Monthly): " + e.calculateYearlySalary(5000));
        System.out.println("Yearly Salary (Daily): " + e.calculateYearlySalary(2000, 25));
        System.out.println("Yearly Salary (Hourly): " + e.calculateYearlySalary(50, 8));
        System.out.println(e);
    }
}
```

Output:

```
Yearly Salary (Monthly): 1200000.0
Yearly Salary (Daily): 50000.0
Yearly Salary (Hourly): 147898.0
ID: 137, Name: Shabari, Salary: 100000.0
```

---

3. Question 3

Problem Statement:

Design a Java program to model bank accounts, including standard accounts, savings accounts, and current accounts, using inheritance. The program should allow us like deposit, withdrawal, and display account information. The savings accounts should have an additional feature of earning interest on the account balance, and the overdraft limit to prevent overdrawing.

1. Create a class called "Account" with the following attributes and methods: Attributes:
    - accountNumber (String)
    - accountName(String)
    - 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.

Analysis:

1. Create a class `Account` with attributes `accountNumber`, `accountName`, and `balance`.
2. Implement methods to deposit, withdraw, get account number, account name, balance, and display account information.
3. Create a subclass `SavingsAccount` that inherits from `Account` and includes an additional attribute `interestRate` and a method `addInterest()`.
4. Create a subclass `CurrentAccount` that inherits from `Account` and includes an additional attribute `overdraftLimit` and a method `getOverdraftLimit()`.
5. Override the `displayInfo()` method in `SavingsAccount` and `CurrentAccount` to include the interest rate and overdraft limit, respectively.
6. Create objects of `Account`, `SavingsAccount`, and `CurrentAccount` classes and demonstrate the working of the methods.

Code:

```
class Account{
    String accountNumber;
    String accountName;
    double balance;

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

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

    void withdraw(double amount){
        if(balance - amount >= 0){
            balance -= amount;
        }
    }

    String getAccountNumber(){
        return accountNumber;
    }

    String getAccountName(){
        return accountName;
    }

    double getBalance(){
        return balance;
    }
```

```
    void displayInfo(){
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: " + balance);
    }
}

class SavingsAccount extends Account{
    double interestRate;

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

    void addInterest(){
        balance += balance * interestRate;
    }

    void displayInfo(){
        super.displayInfo();
        System.out.println("Interest Rate: " + interestRate);
    }
}

class CurrentAccount extends Account{
    double overdraftLimit;

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

    double getOverdraftLimit(){
        return overdraftLimit;
    }

    void displayInfo(){
        super.displayInfo();
        System.out.println("Overdraft Limit: " + overdraftLimit);
    }
}

public class Question3{
    public static void main(String[] args){
        Account a = new Account("123456", 1000);
        a.deposit(500);
        a.withdraw(200);
        a.displayInfo();
        SavingsAccount s = new SavingsAccount("123457", 2000, 0.05);
        s.deposit(500);
        s.withdraw(300);
        s.addInterest();
        s.displayInfo();
        CurrentAccount c = new CurrentAccount("123458", 3000, 500);
        c.deposit(1000);
        c.withdraw(4000);
        c.displayInfo();
    }
}
```

Output:

```
Account Number: 123456
Balance: 1300.0
```

```
Account Number: 123457
Balance: 2310.0
Interest Rate: 0.05
Account Number: 123458
Balance: 0.0
Interest Rate: 500.0
Overdraft Limit: 0.0
```

---

4. Question 4

Problem Statement:

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 interface for geometric shapes while allowing each shape to implement its own logic for calculating area and perimeter.

Analysis:

1. Create an abstract class `Shape` with abstract methods `calculateArea()` and `calculatePerimeter()`.
2. Create concrete classes `Circle`, `Rectangle`, and `Triangle` that extend the `Shape` class and implement the abstract methods.
3. Define the logic for calculating the area and perimeter for each shape in the respective classes.
4. Create objects of `Circle`, `Rectangle`, and `Triangle` classes and demonstrate the working of the methods.

Code:

```
abstract class Shape{
    abstract double calculateArea();
    abstract double calculatePerimeter();
}

class Circle extends Shape{
    double radius;

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

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

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

class Rectangle extends Shape{
    double length;
    double width;

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

    double calculateArea(){
        return length * width;
    }

    double calculatePerimeter(){
        return 2 * (length + width);
```

```
    }
}

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

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

    double calculateArea(){
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    double calculatePerimeter(){
        return side1 + side2 + side3;
    }
}

public class Question4{
    public static void main(String[] args){
        Circle c = new Circle(5);
        System.out.println("Circle Area: " + c.calculateArea());
        System.out.println("Circle Perimeter: " + c.calculatePerimeter());

        Rectangle r = new Rectangle(4, 6);
        System.out.println("Rectangle Area: " + r.calculateArea());
        System.out.println("Rectangle Perimeter: " + r.calculatePerimeter());

        Triangle t = new Triangle(3, 4, 5);
        System.out.println("Triangle Area: " + t.calculateArea());
        System.out.println("Triangle Perimeter: " + t.calculatePerimeter());
    }
}
```

Output:

```
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Rectangle Area: 24.0
Rectangle Perimeter: 20.0
Triangle Area: 6.0
Triangle Perimeter: 12.0
```

---

5. Question 5

Problem Statement:

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 N 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 representi 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 thes sample input data.

Analysis:

1. Create a class `Library` with overloaded methods for searching books based on different criteria such as title, author, or ISBN.
2. Implement methods to handle cases where no matching books are found.
3. Create a main method to demonstrate the usage of the overloaded search methods with sample input data.

Code:

```
class Book{
    String title;
    String author;
    String ISBN;

    Book(String title, String author, String ISBN){
        this.title = title;
        this.author = author;
        this.ISBN = ISBN;
    }
}

class Library{
    Book[] books;

    Library(Book[] books){
        this.books = books;
    }

    Book searchBook(String title){
        for(Book book : books){
            if(book.title.equals(title)){
                return book;
            }
        }
        return null;
    }

    Book searchBook(String author, int a){
        for(Book book : books){
            if(book.author.equals(author)){
                return book;
            }
        }
        return null;
    }

    Book searchBookByISBN(String ISBN){
        for(Book book : books){
            if(book.ISBN.equals(ISBN)){
                return book;
            }
        }
        return null;
    }
}

public class Question5{
    public static void main(String[] args){
        Book[] books = {
            new Book("Java Programming", "John Doe", "123456"),
            new Book("Python Programming", "Jane Smith", "234567"),
            new Book("Data Structures", "Alice Johnson", "345678")
        };
        Library library = new Library(books);
```

```
        Book book1 = library.searchBook("Java Programming");
        if(book1 != null){
            System.out.println("Book found - Title: " + book1.title + ", Author: " + book1.author + ", ISBN: " + book1.ISBN);
        }else{
            System.out.println("Book not found.");
        }
        Book book2 = library.searchBook("Jane Smith", 1);
        if(book2 != null){
            System.out.println("Book found - Title: " + book2.title + ", Author: " + book2.author + ", ISBN: " + book2.ISBN);
        }else{
            System.out.println("Book not found.");
        }
        Book book3 = library.searchBookByISBN("345678");
        if(book3 != null){
            System.out.println("Book found - Title: " + book3.title + ", Author: " + book3.author + ", ISBN: " + book3.ISBN);
        }else{
            System.out.println("Book not found.");
        }
    }
}
```

Output:

```
Book found - Title: Java Programming, Author: John Doe, ISBN: 123456
Book found - Title: Python Programming, Author: Jane Smith, ISBN: 234567
Book found - Title: Data Structures, Author: Alice Johnson, ISBN: 345678
```

---

6. Question 6

Problem Statement:

You are developing a software application for a university that manages student information. As part of the system, you have a base class named "Person" that represe 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 "dis specific to that type. Design and implement a Java class hierarchy for managing student information, starting with a base class named "Person" that include a method subclasses for representing different types of students, such as "UndergraduateStudent" and "GraduateStudent", and override the "displayInfo()" method in each subc that type of student. Provide sample data and demonstrate how the overridden methods are invoked using polymorphism.

Analysis:

1. Create a base class `Person` with a method `displayInfo()` to provide common attributes and behaviors shared among different individuals.
2. Create subclasses `UndergraduateStudent` and `GraduateStudent` to represent different types of students.
3. Override the `displayInfo()` method in each subclass to display relevant information for that type of student.
4. Create objects of `UndergraduateStudent` and `GraduateStudent` classes and demonstrate the use of polymorphism by invoking the overridden methods.

Code:

```
class Person{
    String name;
    int age;

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

    void displayInfo(){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
```

```
    }
}

class UndergraduateStudent extends Person{
    String major;

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

    void displayInfo(){
        super.displayInfo();
        System.out.println("Major: " + major);
    }
}

class GraduateStudent extends Person{
    String researchArea;

    GraduateStudent(String name, int age, String researchArea){
        super(name, age);
        this.researchArea = researchArea;
    }

    void displayInfo(){
        super.displayInfo();
        System.out.println("Research Area: " + researchArea);
    }
}

public class Question6{
    public static void main(String[] args){
        Person p1 = new Person("Alice", 20);
        p1.displayInfo();

        UndergraduateStudent u = new UndergraduateStudent("Bob", 22, "Computer Science");
        u.displayInfo();

        GraduateStudent g = new GraduateStudent("Charlie", 25, "Machine Learning");
        g.displayInfo();
    }
}
```

Output:

```
Name: Alice
Age: 20
Name: Bob
Age: 22
Major: Computer Science
Name: Charlie
Age: 25
Research Area: Machine Learning
```

---

7. Question 7

Problem Statement:

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 i 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:
    - 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 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 s methods.

Analysis:

1. Create an interface `Vehicle` with methods `start()` and `stop()`.
2. Create classes `Car`, `Motorcycle`, and `Truck` that implement the `Vehicle` interface.
3. Implement the `start()` and `stop()` methods specific to each type of vehicle.
4. Create an array of `Vehicle` objects with instances of `Car`, `Motorcycle`, and `Truck`.
5. Iterate over the array and call the `start()` method for each vehicle.
6. Iterate over the array again and call the `stop()` method for each vehicle.

Code:

```
interface Vehicle{
    void start();
    void stop();
}

class Car implements Vehicle{
    public void start(){
        System.out.println("Car started");
    }

    public void stop(){
        System.out.println("Car stopped");
    }
}

class Motorcycle implements Vehicle{
    public void start(){
        System.out.println("Motorcycle started");
    }

    public void stop(){
        System.out.println("Motorcycle stopped");
    }
}

class Truck implements Vehicle{
    public void start(){
        System.out.println("Truck started");
    }
```

```
    public void stop(){
        System.out.println("Truck stopped");
    }
}

public class Question7{
    public static void main(String[] args){
        Vehicle[] vehicles = {new Car(), new Motorcycle(), new Truck()};
        for(Vehicle v : vehicles){
            v.start();
        }
        for(Vehicle v : vehicles){
            v.stop();
        }
    }
}
```

Output:

```
Car Started
Motorcycle Started
Truck Started
Car Stopped
Motorcycle Stopped
Truck Stopped
```