Topic 1: Abstract Class with Abstract and Concrete Methods (Any four)

Problem Statement:

Create an abstract class Shape with abstract methods area() and perimeter(). Provide a concrete method displayInfo().

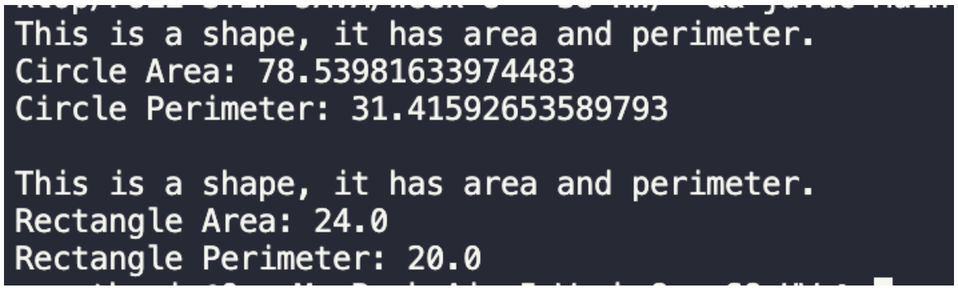
Create subclasses Circle and Rectangle that implement the abstract methods. Test the implementation by creating objects and displaying results.

Hints:

- Use abstract keyword for Shape class.
- Implement area() and perimeter() in subclasses.
- Call displayInfo() from subclass objects.

```
abstract class Shape {
    public abstract double area();
    public abstract double perimeter();

    public void displayInfo() {
        System.out.println("This is a shape. It has an area and perimeter.");
    }
}

class Circle extends Shape {
    private double radius;

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

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

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

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

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

    @Override
    public double area() {
        return length * width;
    }

    @Override
    public double perimeter() {
        return 2 * (length + width);
    }
}


public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        circle.displayInfo();
        System.out.println("Circle Area: " + circle.area());
        System.out.println("Circle Perimeter: " + circle.perimeter());
        System.out.println();


        Shape rectangle = new Rectangle(4, 6);
        rectangle.displayInfo();
        System.out.println("Rectangle Area: " + rectangle.area());
        System.out.println("Rectangle Perimeter: " + rectangle.perimeter());
    }
}
```

OUTPUT:-



```
This is a shape, it has area and perimeter.
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793

This is a shape, it has area and perimeter.
Rectangle Area: 24.0
Rectangle Perimeter: 20.0
```

Topic 2: Interface Implementation in Multiple Classes

Problem Statement:

Create an interface Playable with methods play() and pause().

Create two classes MusicPlayer and VideoPlayer that implement this interface.

Demonstrate polymorphism by storing objects in a Playable reference and invoking methods.

Hints:

● Use interface keyword.

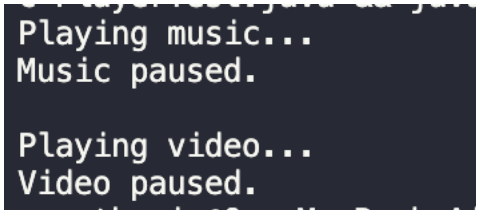
● Implement both methods in each class.

● Use Playable ref = new MusicPlayer(); to test polymorphism.

```
// Interface Playable
interface Playable {
    void play();
    void pause();
}

// Class MusicPlayer implements Playable
class MusicPlayer implements Playable {
    @Override
    public void play() {
        System.out.println("Playing music...");
    }

    @Override
    public void pause() {
        System.out.println("Music paused.");
    }
}
```

```
// Class VideoPlayer implements Playable
class VideoPlayer implements Playable {
    @Override
    public void play() {
        System.out.println("Playing video...");
    }

    @Override
    public void pause() {
        System.out.println("Video paused.");
    }
}

// Different main class name
public class PlayerTest {
    public static void main(String[] args) {
        // Polymorphism with MusicPlayer
        Playable ref = new MusicPlayer();
        ref.play();
        ref.pause();

        System.out.println();

        // Polymorphism with VideoPlayer
        ref = new VideoPlayer();
        ref.play();
        ref.pause();
    }
}
```

OUTPUT:-



```
Playing music...
Music paused.

Playing video...
Video paused.
```

Topic 3: Abstract Class + Interface Together

Problem Statement:

Create an abstract class Vehicle with abstract method start() and a concrete method stop().

Create an interface Fuel with method refuel().

Create class Car that extends Vehicle and implements Fuel. Test all methods.

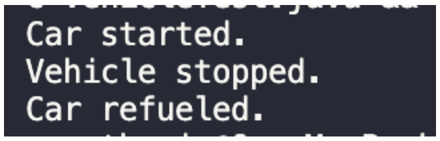
Hints:

- Use abstract class for Vehicle.
- Implement refuel() from Fuel interface in Car.
- Show method calls of start(), stop(), and refuel().

```
// Abstract class Vehicle
abstract class Vehicle {
   // Abstract method
   public abstract void start();

   // Concrete method
   public void stop() {
      System.out.println("Vehicle stopped.");
   }
}

// Interface Fuel
interface Fuel {
   void refuel();
}

// Class Car extends Vehicle and implements Fuel
class Car extends Vehicle implements Fuel {
   @Override
   public void start() {
      System.out.println("Car started.");
   }
```

```
    @Override
    public void refuel() {
        System.out.println("Car refueled.");
    }
}

// Main class to test
public class VehicleTest {
    public static void main(String[] args) {
        Car myCar = new Car();

        // Call methods
        myCar.start();    // Abstract method implemented
        myCar.stop();     // Concrete method from Vehicle
        myCar.refuel();   // Interface method implemented
    }
}
```

OUTPUT:-



```
Car started.
Vehicle stopped.
Car refueled.
```

Topic 4: Interface Inheritance (Extending Interface)

Problem Statement:

Create an interface Animal with method eat().

Create another interface Pet that extends Animal and adds method play().

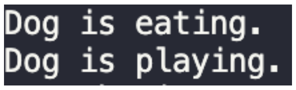
Create a class Dog that implements Pet. Demonstrate interface inheritance in action.

Hints:

- Use interface Pet extends Animal.
- Dog must implement both eat() and play().
- Create object of Dog and test.

```
// Base interface
interface Animal {
    void eat();
}

// Derived interface
interface Pet extends Animal {
    void play();
}

// Class Dog implements Pet (inherits Animal too)
class Dog implements Pet {
    @Override
    public void eat() {
        System.out.println("Dog is eating.");
    }

    @Override
    public void play() {
        System.out.println("Dog is playing.");
    }
}

// Main class to test
public class PetTest {
    public static void main(String[] args) {
        Dog myDog = new Dog();

        // Call methods
        myDog.eat();   // From Animal interface
        myDog.play();  // From Pet interface
    }
}
```

OUTPUT:-



```
Dog is eating.
Dog is playing.
```

Topic 5: Abstraction in Real-world Example

Problem Statement:

Create an abstract class BankAccount with abstract method calculateInterest() and concrete method deposit().

Create subclasses SavingsAccount and CurrentAccount that provide specific interest calculation logic.

Test the program by creating objects and calling methods.

Hints:

● Define abstract void calculateInterest(); in BankAccount.

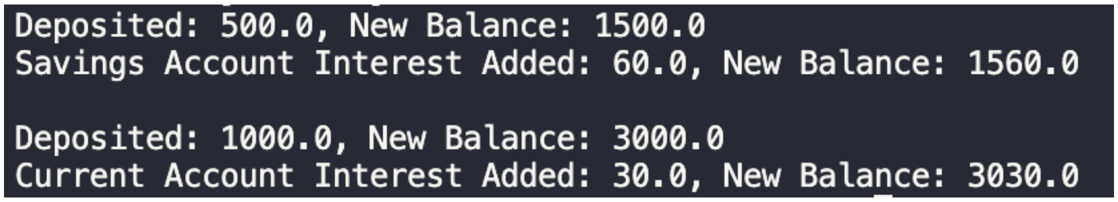
● Override calculateInterest() differently in SavingsAccount and CurrentAccount.

● Use constructor to set balance and test deposit/interest methods.

```
}



// Abstract class BankAccount
abstract class BankAccount {
    protected double balance;

    // Constructor
    BankAccount(double balance) {
        this.balance = balance;
    }

    // Abstract method
    public abstract void calculateInterest();

    // Concrete method
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount + ", New Balance: " + balance);
    }
}

// SavingsAccount subclass
class SavingsAccount extends BankAccount {
    private double interestRate = 0.04; // 4% interest

    SavingsAccount(double balance) {
        super(balance);
    }
    @Override
    public void calculateInterest() {
        double interest = balance * interestRate;
        balance += interest;
        System.out.println("Savings Account Interest Added: " + interest + ", New Balance:
" + balance);
    }
}

// CurrentAccount subclass
class CurrentAccount extends BankAccount {
    private double interestRate = 0.01; // 1% interest (or can be zero)

    CurrentAccount(double balance) {
        super(balance);
    }

    @Override
    public void calculateInterest() {
        double interest = balance * interestRate;
        balance += interest;
        System.out.println("Current Account Interest Added: " + interest + ", New Balance:
" + balance);
    }
}

// Main class to test
```

```
public class BankTest {
    public static void main(String[] args) {
        // Test SavingsAccount
        BankAccount savings = new SavingsAccount(1000);
        savings.deposit(500);
        savings.calculateInterest();

        System.out.println();

        // Test CurrentAccount
        BankAccount current = new CurrentAccount(2000);
        current.deposit(1000);
        current.calculateInterest();
    }
}
```

OUTPUT:-



```
Deposited: 500.0, New Balance: 1500.0
Savings Account Interest Added: 60.0, New Balance: 1560.0

Deposited: 1000.0, New Balance: 3000.0
Current Account Interest Added: 30.0, New Balance: 3030.0
```

Topic 6: Multiple Interfaces with Same Method Name

Problem Statement:

Create two interfaces Printer and Scanner, each having a method connect().

Create a class AllInOneMachine that implements both interfaces and provides its own implementation for connect().

Demonstrate how a single class can resolve method name conflicts and handle multiple interfaces.

Hints:

● Use interface Printer and interface Scanner.

● Both will have a method void connect().

● In AllInOneMachine, implement both connect() methods (since they have same signature, one method will serve both).

● Create objects and test with references:

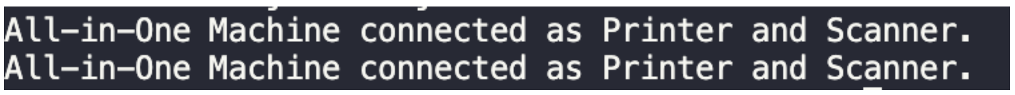
○ Printer p = new AllInOneMachine();

○ Scanner s = new AllInOneMachine();

```
// Interface Printer
interface Printer {
    void connect();
}

// Interface Scanner
interface Scanner {
    void connect();
}

// Class implementing both interfaces
class AllInOneMachine implements Printer, Scanner {
    @Override
    public void connect() {
        System.out.println("All-in-One Machine connected as Printer and Scanner.");
    }
}

// Main class to test
public class MachineTest {
    public static void main(String[] args) {
        // Using Printer reference
        Printer p = new AllInOneMachine();
        p.connect();  // Calls AllInOneMachine's connect()

        // Using Scanner reference
        Scanner s = new AllInOneMachine();
        s.connect();  // Calls same connect()
    }
```

}

OUTPUT:-



```
All-in-One Machine connected as Printer and Scanner.
All-in-One Machine connected as Printer and Scanner.
```