Lab8

W.A.C Fernando (26545)

1.

```
abstract class BankAccount {
    private String accountNumber;
    private double balance;

    public String getAccountNumber() {
        return accountNumber;
    }

    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    // Abstract method to be implemented by subclasses
    public abstract double calculateInterest();
}


// SavingsAccount class that extends BankAccount and provides its
implementation for the "calculateInterest" method

class SavingsAccount extends BankAccount {
    private static final double SAVINGS_INTEREST_RATE = 0.12; // 12%

    @Override
    public double calculateInterest() {
        return getBalance() * SAVINGS_INTEREST_RATE;
    }
}


//CheckingAccount class that extends BankAccount and provides its
implementation for the "calculateInterest" method

class CheckingAccount extends BankAccount {
    private static final double CHECKING_INTEREST_RATE = 0.02; // 2%
    @Override
    public double calculateInterest() {
        return getBalance() * CHECKING_INTEREST_RATE;
    }
}




public class Main {
    public static void main(String[] args) {
        CheckingAccount checkingAccount = new CheckingAccount();
        checkingAccount.setBalance(1000000); // 1 million
        double checkingInterest = checkingAccount.calculateInterest();

        SavingsAccount savingsAccount = new SavingsAccount();
        savingsAccount.setBalance(20000000); // 20 million
        double savingsInterest = savingsAccount.calculateInterest();

        System.out.println("Interest for Checking Account: " + checkingInterest);
        System.out.println("Interest for Savings Account: " + savingsInterest);
    }
}



//output
Interest for Checking Account: 20000.0
Interest for Savings Account: 2400000.0
```

2.

```
//Shape interface with two abstract methods

public interface Shape {
    double calculateArea();
    double calculatePerimeter();
}


//Circle class that implements the "Shape" interface
public class Circle implements Shape {
    private double radius;

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

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

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

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


//Rectangle class that implements the "Shape" interface
public class Rectangle implements Shape {
    private double length;
    private double width;

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

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
```

```java
    }

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


//Triangle class that implements the "Shape" interface
public class Triangle implements Shape {
    private double sideA;
    private double sideB;
    private double sideC;

    public Triangle(double sideA, double sideB, double sideC) {
        this.sideA = sideA;
        this.sideB = sideB;
        this.sideC = sideC;
    }

    public double getSideA() {
        return sideA;
    }

    public void setSideA(double sideA) {
        this.sideA = sideA;
    }

    public double getSideB() {
        return sideB;
    }

    public void setSideB(double sideB) {
        this.sideB = sideB;
    }

    public double getSideC() {
        return sideC;
    }

    public void setSideC(double sideC) {
        this.sideC = sideC;
    }

    @Override
    public double calculateArea() {
        // Implement area calculation based on the sides of the triangle (Heron's
formula or other methods)
        // For simplicity, let's assume it's an equilateral triangle
        double s = (sideA + sideB + sideC) / 2;
        return Math.sqrt(s * (s - sideA) * (s - sideB) * (s - sideC));
    }

    @Override
    public double calculatePerimeter() {
        return sideA + sideB + sideC;
    }
}
```