RegNo: 23MCA1030 **Name**: Vinayak Kumar Singh

Java Programming Lab (PMCA502P)

Exercise 6

1. Calculator Interface: Define an interface named Calculator with methods add, subtract, multiply, and divide. Implement this interface in a class named BasicCalculator. Test the functionality of the BasicCalculator class by performing arithmetic operations.

Code:

```
// Calculator Interface
interface Calculator {
    double add(double a, double b);
    double subtract(double a, double b);
    double multiply(double a, double b);
    double divide(double a, double b);
}
// BasicCalculator class implementing the Calculator interface
class BasicCalculator implements Calculator {
    @Override
    public double add(double a, double b) {
        return a + b;
    }
    @Override
    public double subtract(double a, double b) {
        return a - b;
    }
    @Override
    public double multiply(double a, double b) {
        return a * b;
    }
```

```
    @Override
    public double divide(double a, double b) {
        if (b != 0) {
            return a / b;
        } else {
            throw new ArithmeticException("Cannot divide by zero.");
        }
    }
}
class Main {
    public static void main(String[] args) {
        BasicCalculator calculator = new BasicCalculator();
        double a = 10.0;
        double b = 3.0;
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        System.out.println("Addition: " + a + " + " + b + " = " + calculator.add(a, b));
        System.out.println("Subtraction: " + a + " - " + b + " = " +
calculator.subtract(a, b));
        System.out.println("Multiplication: " + a + " * " + b + " = " +
calculator.multiply(a, b));
        System.out.println("Division: " + a + " / " + b + " = " + calculator.divide(a, b));
    }
}
```

Output:

```
d:\Coding\Java\Ex6\1>cd "d:\Coding\Java\Ex6\1\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Addition: 10.0 + 3.0 = 13.0
Subtraction: 10.0 - 3.0 = 7.0
Multiplication: 10.0 * 3.0 = 30.0
Division: 10.0 / 3.0 = 3.3333333333333335
```

2.Shape Interface:

Here are partial versions of the code for the remaining exercises

// Shape interface

interface Shape {

double calculateArea();

double calculatePerimeter();

}

// You need to implement the Circle, Rectangle, and Triangle classes here

// Test class

public class Main {

public static void main(String[] args) {

// Create shapes and test functionality

}

}

Your task is to complete the code by implementing the Circle, Rectangle, and Triangle classes, which should implement the Shape interface and provide concrete implementations for each method. Once you've completed that, you can test the functionality in the Main class.

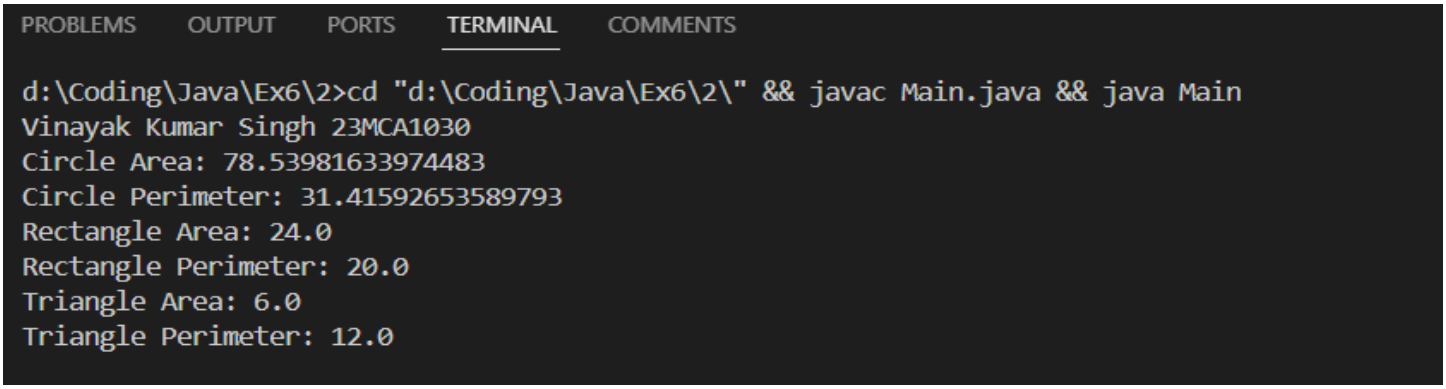
Code:

```
// Shape interface
interface Shape {
    double calculateArea();
    double calculatePerimeter();
}
// Circle class
class Circle implements Shape {
    private double radius;
    public Circle(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
class Rectangle implements Shape {
    private double length;
    private double width;
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    @Override
    public double calculateArea() {
        return length * width;
    }
    @Override
    public double calculatePerimeter() {
        return 2 * (length + width);
    }
}
// Triangle class
class Triangle implements Shape {
    private double side1;
    private double side2;
    private double side3;
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
```

```
    }
    @Override
    public double calculateArea() {
        double semiPerimeter = calculatePerimeter() / 2;
        return Math.sqrt(semiPerimeter * (semiPerimeter - side1) * (semiPerimeter - side2) * (semiPerimeter - side3));
    }
    @Override
    public double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}
// Test class
public class Main {
    public static void main(String[] args) {
        // Create shapes and test functionality
        Circle circle = new Circle(5.0);
        Rectangle rectangle = new Rectangle(4.0, 6.0);
        Triangle triangle = new Triangle(3.0, 4.0, 5.0);
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        System.out.println("Circle Area: " + circle.calculateArea());
        System.out.println("Circle Perimeter: " + circle.calculatePerimeter());
        System.out.println("Rectangle Area: " + rectangle.calculateArea());
        System.out.println("Rectangle Perimeter: " + rectangle.calculatePerimeter());
        System.out.println("Triangle Area: " + triangle.calculateArea());
        System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());
    }
}
```

Output:



3. Bank Account Interface:

Design an interface named BankAccount with methods deposit, withdraw, and getBalance. Implement this interface in classes SavingsAccount and CurrentAccount. Test these classes by depositing, withdrawing, and checking balances.

Code:

```
// BankAccount interface
interface BankAccount {
    void deposit(double amount);
    void withdraw(double amount);
    double getBalance();
}
// SavingsAccount class
class SavingsAccount implements BankAccount {
    private double balance;
    public SavingsAccount(double initialBalance) {
        this.balance = initialBalance;
    }
    @Override
    public void deposit(double amount) {
        balance += amount;
    }
    @Override
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance");
        }
    }
    @Override
    public double getBalance() {
        return balance;
    }
}
```

```
// CurrentAccount class
class CurrentAccount implements BankAccount {
    private double balance;
    private double overdraftLimit;
    public CurrentAccount(double initialBalance, double overdraftLimit) {
        this.balance = initialBalance;
        this.overdraftLimit = overdraftLimit;
    }
    @Override
    public void deposit(double amount) {
        balance += amount;
    }
    @Override
    public void withdraw(double amount) {
        if (balance + overdraftLimit >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance and overdraft limit");
        }
    }
    @Override
    public double getBalance() {
        return balance;
    }
    public double getOverdraftLimit() {
        return overdraftLimit;
    }
}
// Test class
class Main {
    public static void main(String[] args) {
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Test SavingsAccount
        SavingsAccount savingsAccount = new SavingsAccount(1000.0);
        System.out.println("Savings Account Balance: " +
savingsAccount.getBalance());
```

```
        savingsAccount.deposit(500.0);
        System.out.println("Savings Account Balance after Deposit: " +
savingsAccount.getBalance());
        savingsAccount.withdraw(1200.0);
        System.out.println("Savings Account Balance after Withdraw: " +
savingsAccount.getBalance());
        System.out.println();
        // Test CurrentAccount
        CurrentAccount currentAccount = new CurrentAccount(2000.0, 500.0);
        System.out.println("Current Account Balance: " +
currentAccount.getBalance());
        System.out.println("Current Account Overdraft Limit: " +
currentAccount.getOverdraftLimit());
        currentAccount.deposit(1000.0);
        System.out.println("Current Account Balance after Deposit: " +
currentAccount.getBalance());
        currentAccount.withdraw(3000.0);
        System.out.println("Current Account Balance after Withdraw: " +
currentAccount.getBalance());
        currentAccount.withdraw(1000.0);
        System.out.println("Current Account Balance after Withdraw: " +
currentAccount.getBalance());
    }
}
```

Output:

```
PROBLEMS    OUTPUT    PORTS    TERMINAL    COMMENTS

d:\Coding\Java\Ex6\3>cd "d:\Coding\Java\Ex6\3\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Savings Account Balance: 1000.0
Savings Account Balance after Deposit: 1500.0
Savings Account Balance after Withdraw: 300.0

Current Account Balance: 2000.0
Current Account Overdraft Limit: 500.0
Current Account Balance after Deposit: 3000.0
Current Account Balance after Withdraw: 0.0
Insufficient balance and overdraft limit
Current Account Balance after Withdraw: 0.0
```

4. Employee Interface:

Define an interface named Employee with methods calculateSalary and displayDetails. Implement this interface in classes Manager, Clerk, and Technician. Calculate and display the salary details for each type of employee.

Code:

```
// Employee interface
interface Employee {
    double calculateSalary();
    void displayDetails();
}
// Manager class
class Manager implements Employee {
    private String name;
    private double baseSalary;
    private double bonus;
    public Manager(String name, double baseSalary, double bonus) {
        this.name = name;
        this.baseSalary = baseSalary;
        this.bonus = bonus;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + bonus;
    }
    @Override
    public void displayDetails() {
        System.out.println("Employee Type: Manager");
        System.out.println("Name: " + name);
        System.out.println("Salary: " + calculateSalary());
    }
}
// Clerk class
class Clerk implements Employee {
    private String name;
    private double baseSalary;
```

```
    private double transportAllowance;
    public Clerk(String name, double baseSalary, double transportAllowance) {
        this.name = name;
        this.baseSalary = baseSalary;
        this.transportAllowance = transportAllowance;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + transportAllowance;
    }
    @Override
    public void displayDetails() {
        System.out.println("Employee Type: Clerk");
        System.out.println("Name: " + name);
        System.out.println("Salary: " + calculateSalary());
    }
}
// Technician class
class Technician implements Employee {
    private String name;
    private double baseSalary;
    private double overtimeHours;
    private double overtimeRate;
    public Technician(String name, double baseSalary, double overtimeHours,
double overtimeRate) {
        this.name = name;
        this.baseSalary = baseSalary;
        this.overtimeHours = overtimeHours;
        this.overtimeRate = overtimeRate;
    }
    @Override
    public double calculateSalary() {
        return baseSalary + (overtimeHours * overtimeRate);
    }
    @Override
    public void displayDetails() {
```

```
        System.out.println("Employee Type: Technician");
        System.out.println("Name: " + name);
        System.out.println("Salary: " + calculateSalary());
    }
}
// Test class
class Main {
    public static void main(String[] args) {
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Create employees and display details
        Manager manager = new Manager("Deepak Dewangan", 5000.0, 1000.0);
        Clerk clerk = new Clerk("Aniket Shriwas", 3000.0, 500.0);
        Technician technician = new Technician("Aman Nirmalkar", 4000.0, 20.0,
50.0);
        manager.displayDetails();
        System.out.println();
        clerk.displayDetails();
        System.out.println();
        technician.displayDetails();
    }
}
```

Output:

```
PROBLEMS    OUTPUT    PORTS    TERMINAL    COMMENTS

D:\Coding\Java>cd "d:\Coding\Java\Ex6\4\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Employee Type: Manager
Name: Deepak Dewangan
Salary: 6000.0

Employee Type: Clerk
Name: Aniket Shriwas
Salary: 3500.0

Employee Type: Technician
Name: Aman Nirmalkar
Salary: 5000.0
```

5. Animal Interface:

Create an interface named Animal with methods eat, sleep, and makeSound. Implement this interface in classes Dog, Cat, and Bird. Test the implementation by calling methods for different types of animals.

Code:

```
// Animal interface
interface Animal {
    void eat();
    void sleep();
    void makeSound();
}
// Dog class
class Dog implements Animal {
    @Override
    public void eat() {
        System.out.println("The dog is eating.");
    }
    @Override
    public void sleep() {
        System.out.println("The dog is sleeping.");
    }
    @Override
    public void makeSound() {
        System.out.println("The dog says: Woof! Woof!");
    }
}
// Cat class
class Cat implements Animal {
    @Override
    public void eat() {
        System.out.println("The cat is eating.");
    }
    @Override
    public void sleep() {
        System.out.println("The cat is sleeping.");
```

```
    }
    @Override
    public void makeSound() {
        System.out.println("The cat says: Meow! Meow!");
    }
}
// Bird class
class Bird implements Animal {
    @Override
    public void eat() {
        System.out.println("The bird is eating.");
    }
    @Override
    public void sleep() {
        System.out.println("The bird is sleeping.");
    }
    @Override
    public void makeSound() {
        System.out.println("The bird says: Chirp! Chirp!");
    }
}
// Test class
public class Main {
    public static void main(String[] args) {
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Create animals and test functionality
        Dog dog = new Dog();
        Cat cat = new Cat();
        Bird bird = new Bird();
        dog.eat();
        dog.sleep();
        dog.makeSound();
        System.out.println();
        cat.eat();
        cat.sleep();
        cat.makeSound();
```

```
        System.out.println();
        bird.eat();
        bird.sleep();
        bird.makeSound();
    }
}
```

Output:

```
PROBLEMS  OUTPUT  PORTS  TERMINAL  COMMENTS

d:\Coding\Java\Ex6\4>cd "d:\Coding\Java\Ex6\5\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
The dog is eating.
The dog is sleeping.
The dog says: Woof! Woof!

The cat is eating.
The cat is sleeping.
The cat says: Meow! Meow!

The bird is eating.
The bird is sleeping.
The bird says: Chirp! Chirp!
```

6. Define an interface named Vehicle with methods start and stop. Include a constant variable MAX_SPEED representing the maximum speed of the vehicle. Implement this interface in classes Car and Bicycle. Test the functionality by starting and stopping vehicles.

Expected Output:

Car started. Maximum speed: 60 km/h

Car stopped.

Bicycle started. Maximum speed: 60 km/h

Bicycle stopped.

Code:

```
// Vehicle interface
interface Vehicle {
    int MAX_SPEED = 60; // km/h
    void start();
    void stop();
}
// Car class
class Car implements Vehicle {
    @Override
    public void start() {
        System.out.println("Car started. Maximum speed: " + MAX_SPEED + "
km/h");
    }
    @Override
    public void stop() {
        System.out.println("Car stopped.");
    }
}
// Bicycle class
class Bicycle implements Vehicle {
    @Override
    public void start() {
```

```
        System.out.println("Bicycle started. Maximum speed: " + MAX_SPEED + "
km/h");
    }
    @Override
    public void stop() {
        System.out.println("Bicycle stopped.");
    }
}
// Test class
public class Main {
    public static void main(String[] args) {
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Create vehicles and test functionality
        Car car = new Car();
        Bicycle bicycle = new Bicycle();
        car.start();
        car.stop();
        System.out.println();
        bicycle.start();
        bicycle.stop();
    }
}
```

Output:

```
PROBLEMS   OUTPUT   PORTS   TERMINAL   COMMENTS

d:\Coding\Java\Ex6\5>cd "d:\Coding\Java\Ex6\6\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Car started. Maximum speed: 60 km/h
Car stopped.

Bicycle started. Maximum speed: 60 km/h
Bicycle stopped.
```

7. ElectronicDevice Interface Extension:

Design a base interface named ElectronicDevice with methods turnOn and turnOff. Extend this interface to create another interface named SmartDevice with additional methods like connectToInternet and runApp. Implement both interfaces in classes like Smartphone, SmartTV, and SmartWatch. Test the functionality by turning on, turning off, and performing smart actions on devices.

Code:

```
// ElectronicDevice interface
interface ElectronicDevice {
    void turnOn();
    void turnOff();
}
// SmartDevice interface extends ElectronicDevice
interface SmartDevice extends ElectronicDevice {
    void connectToInternet();
    void runApp();
}
// Smartphone class implements SmartDevice
class Smartphone implements SmartDevice {
    @Override
    public void turnOn() {
        System.out.println("Smartphone turned on.");
    }
    @Override
    public void turnOff() {
        System.out.println("Smartphone turned off.");
    }
    @Override
    public void connectToInternet() {
        System.out.println("Smartphone connected to the internet.");
    }
    @Override
    public void runApp() {
```

```
        System.out.println("Running app on the smartphone.");
    }
}
// SmartTV class implements SmartDevice
class SmartTV implements SmartDevice {
    @Override
    public void turnOn() {
        System.out.println("Smart TV turned on.");
    }
    @Override
    public void turnOff() {
        System.out.println("Smart TV turned off.");
    }
    @Override
    public void connectToInternet() {
        System.out.println("Smart TV connected to the internet.");
    }
    @Override
    public void runApp() {
        System.out.println("Running app on the smart TV.");
    }
}
// SmartWatch class implements ElectronicDevice (not SmartDevice)
class SmartWatch implements ElectronicDevice {
    @Override
    public void turnOn() {
        System.out.println("Smart watch turned on.");
    }
    @Override
    public void turnOff() {
        System.out.println("Smart watch turned off.");
    }
}
// Test class
class Main {
    public static void main(String[] args) {
```

```
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Test Smartphone
        Smartphone smartphone = new Smartphone();
        smartphone.turnOn();
        smartphone.connectToInternet();
        smartphone.runApp();
        smartphone.turnOff();
        System.out.println();
        // Test SmartTV
        SmartTV smartTV = new SmartTV();
        smartTV.turnOn();
        smartTV.connectToInternet();
        smartTV.runApp();
        smartTV.turnOff();
        System.out.println();
        // Test SmartWatch
        SmartWatch smartWatch = new SmartWatch();
        smartWatch.turnOn();
        smartWatch.turnOff();
    }
}
```

Output:

```
PROBLEMS  OUTPUT  PORTS  TERMINAL  COMMENTS

d:\Coding\Java\Ex6\6>cd "d:\Coding\Java\Ex6\7\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
Smartphone turned on.
Smartphone connected to the internet.
Running app on the smartphone.
Smartphone turned off.

Smart TV turned on.
Smart TV connected to the internet.
Running app on the smart TV.
Smart TV turned off.

Smart watch turned on.
Smart watch turned off.
```

8. Define a base interface named Employee with methods work and takeBreak. Extend this interface to create two more interfaces, Manager and Clerk, each with additional methods such as manageTeam and organizeTasks. Implement these interfaces in classes like TeamManager, SalesClerk, and OfficeClerk. Test the functionality by simulating employee actions.

Code

```
// Employee interface
interface Employee {
    void work();
    void takeBreak();
}
// Manager interface extends Employee
interface Manager extends Employee {
    void manageTeam();
}
// Clerk interface extends Employee
interface Clerk extends Employee {
    void organizeTasks();
}
// TeamManager class implements Manager
class TeamManager implements Manager {
    @Override
    public void work() {
        System.out.println("The team manager is working.");
    }
    @Override
    public void takeBreak() {
        System.out.println("The team manager is taking a break.");
    }
    @Override
    public void manageTeam() {
        System.out.println("The team manager is managing the team.");
    }
}
// SalesClerk class implements Clerk
```

```
class SalesClerk implements Clerk {
    @Override
    public void work() {
        System.out.println("The sales clerk is working.");
    }
    @Override
    public void takeBreak() {
        System.out.println("The sales clerk is taking a break.");
    }
    @Override
    public void organizeTasks() {
        System.out.println("The sales clerk is organizing tasks.");
    }
}
// OfficeClerk class implements Clerk
class OfficeClerk implements Clerk {
    @Override
    public void work() {
        System.out.println("The office clerk is working.");
    }
    @Override
    public void takeBreak() {
        System.out.println("The office clerk is taking a break.");
    }
    @Override
    public void organizeTasks() {
        System.out.println("The office clerk is organizing tasks.");
    }
}
// Test class
class Main {
    public static void main(String[] args) {
        System.out.println("Vinayak Kumar Singh 23MCA1030");
        // Test TeamManager
        TeamManager teamManager = new TeamManager();
        teamManager.work();
```

```
        teamManager.takeBreak();
        teamManager.manageTeam();
        System.out.println();
        // Test SalesClerk
        SalesClerk salesClerk = new SalesClerk();
        salesClerk.work();
        salesClerk.takeBreak();
        salesClerk.organizeTasks();
        System.out.println();
        // Test OfficeClerk
        OfficeClerk officeClerk = new OfficeClerk();
        officeClerk.work();
        officeClerk.takeBreak();
        officeClerk.organizeTasks();
    }
}
```

Output

```
PROBLEMS    OUTPUT    PORTS    TERMINAL    COMMENTS

D:\Coding\Java>cd "d:\Coding\Java\Ex6\8\" && javac Main.java && java Main
Vinayak Kumar Singh 23MCA1030
The team manager is working.
The team manager is taking a break.
The team manager is managing the team.

The sales clerk is working.
The sales clerk is taking a break.
The sales clerk is organizing tasks.

The office clerk is working.
The office clerk is taking a break.
The office clerk is organizing tasks.
```