Lab: Polymorphism

Problems for exercises and homework for the "Java OOP Basics" course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/481/Polymorphism-Lab .

1. Overload Method

Create a class **MathOperation**, which should have method **add().** Method **add()** have to be invoked with **two, three** or **four Integers.**

You should be able to use the class like this:

Main.java

```
public static void main(String[] args) throws IOException {
    MathOperation math = new MathOperation();
    System.out.println(math.add(2, 2));
    System.out.println(math.add(3, 3, 3));
    System.out.println(math.add(4, 4, 4, 4));
}
```

Examples

Input	Output
	4 9 16

Solution

Class **MathOperation** should look like this:

```
public class MathOperation {
    public int add(int a, int b) {
        return a + b;
    }

    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public int add(int a, int b, int c, int d) {
        return a + b + c + d;
    }
}
```

2. Method Overriding

Read **n** lines from console. If line consist only **one Double** number it is square, if numbers are **two** it is rectangle. Numbers are sides of Rectangle. You need to have two classes:

- **Rectangle**
- **Square**

You should be able to use the class like this:

Main.java

```
public static void main(String[] args) throws IOException {
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    int n = Integer.parseInt(reader.readLine());
    List<Rectangle> listOfRectangles = new ArrayList<>();

    for (int i = 0; i < n; i++) {
        String[] reminder = reader.readLine().split( " ");
        if (reminder.length == 1) {
            listOfRectangles.add(new Square(Double.parseDouble(reminder[0])));
        } else {
            listOfRectangles.add(new Rectangle(Double.parseDouble(reminder[0]),
                                               Double.parseDouble(reminder[1])));
        }
    }

    for (Rectangle rectangle : listOfRectangles) {
        System.out.println(rectangle.area());
    }
}
```

Examples

Input	Output
5 5 25 25 4 5 3 7 2	25.0 625.0 20.0 21.0 4.0

Solution

Square class should look like this:

```
public class Square extends Rectangle {
    public Square(Double side) {
        super(side);
    }

    @Override
    public Double area() {
        return this.sideA * this.sideA;
    }
}
```

3. Shapes

Create class hierarchy, starting with abstract class **Shape**:

- **Fields:**
  - **perimeter**
  - **area**
- **Encapsulation for this fields**
- **Abstract methods:**
  - `calculatePerimeter()`
  - `calculateArea()`

Extend Shape class with two children:

- **Rectangle**
- **Circle**

Each of them need to have:

- **Fields:**
  - **height and width for Rectangle**
  - **radius for Circle**
- **Encapsulation for this fields**
- **Public constructor**
- **Concrete methods for calculations (perimeter and area)**