CS2030 Programming Methodology II

Semester 1 2023/2024

6 & 7 September 2023

Problem Set #2

Interface

1. Given the following interfaces.

```
interface Shape {
    double getArea();
}

interface Printable {
    void print();
}
```

   (a) Suppose class `Circle` implements both interfaces above. Given the following program fragment,

   ```
   Circle c = new Circle(10);
   Shape s = c;
   Printable p = c;
   ```

   Are the following statements allowed? Why do you think Java does not allow some of the following statements?

   i. `s.print();`
   ii. `p.print();`
   iii. `s.getArea();`
   iv. `p.getArea();`

   (b) Now let's define another interface `PrintableShape` as

   ```
   interface PrintableShape extends Printable, Shape { }
   ```

   and let class `Circle` implement `PrintableShape` instead.

   Can an interface inherit from multiple parent interfaces? Would the following statements be allowed?

   ```
   Circle c = new Circle(10);
   PrintableShape ps = c;
   ```

   i. `ps.print();`
   ii. `ps.getArea();`

2. Given the following `Circle` and `Rectangle` classes that implement the `Shape` interface,

```
interface Shape {
    public double getArea();
}

class Circle implements Shape {
    private final double radius;

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

    public double getArea() {
        return Math.PI * this.radius * this.radius;
    }

    public String toString() {
        return "Circle with radius " + this.radius;
    }
}

class Rectangle implements Shape {
    private final double length;
    private final double width;

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

    public double getArea() {
        return this.length * this.width;
    }

    public String toString() {
        return "Rectangle " + this.length + " x " + this.width;
    }
}
```

we have seen how both a `Circle` and `Rectangle` object can be passed to the following `findVolume` method to have its volume computed.

```
double findVolume(Shape shape, double height) {
    return shape.getArea() * height;
}
```

Now your friend decided to create `Shape` as a class to represent both a circle and rectangle:

```
class Shape {
    String type;
    double a;
    double b;

    Shape(double radius) {
        this.type = "Circle";
        this.a = radius;
        this.b = 0;
    }

    Shape(double length, double width) {
        this.type = "Rectangle";
        this.a = length;
        this.b = width;
    }

    double getArea() {
        if (this.type.equals("Circle")) {
            return Math.PI * this.a * this.a;
        } else {
            return this.a * this.b;
        }
    }

    public String toString() {
        if (this.type.equals("Circle")) {
            return "Circle with radius " + this.a;
        } else {
            return "Rectangle " + this.a + " x " + this.b;
        }
    }
}
```

which when passed to `findVolume` would still return the same outcome. Justify why *programming to an interface* is a better implementation?
Hint: what if we need to include a `Square` into our implementation?

3. You are given the following method that returns the maximum integer within a non-empty list of integer elements.

```
int maximum(List<Integer> list) {
    int m = list.get(0);
    int i = 1;
    for (i = 1; i < list.size(); i++) {
        if (list.get(i) > m) {
            m = list.get(i);
        }
    }
    return m;
}
```

We would like to include an equivalent `minimum` method that returns the minimum integer element. How do we define the two methods while avoiding code duplication?