CS2030 Programming Methodology

Semester 1 2020/2021

26 August 2020

Problem Set #1

Basics of Object-Oriented Programming

1. Consider the following two classes:

```
class P {
    private int x;

    void changeSelf() {
        x = 1;
    }

    void changeAnother(P p) {
        p.x = 1;
    }
}

class Q {
    void changeAnother(P p) {
        p.x = 1;
    }
}
```

   (a) Which line(s) above violate the private access modifier of `x`?

   (b) What does this say about the concept of an "abstraction barrier"?

2. Consider the following definition of a `Vector2D` class:

```
class Vector2D {
    private double x;
    private double y;

    Vector2D(double x, double y) {
        this.x = x;
        this.y = y;
    }


    void add(Vector2D v) {
        this.x = this.x + v.x;
        this.y = this.y + v.y;
        // line A
    }
}
```

(a) Suppose that the following program fragment is executed in `JShell`, show the content of the stack and the heap when the execution reaches the line labelled `A` above.

```
Vector2D v1 = new Vector2D(1, 1);
Vector2D v2 = new Vector2D(2, 2);
v1.add(v2);
```

Label your variables and the values they hold clearly. You can use arrows to indicate object references.

(b) Suppose that the representation of `x` and `y` have been changed to a double array:

```
class Vector2D {
    private double[] coord2D;

    ...
}
```

i. What changes do you need for the other parts of class `Vector2D`?

ii. Would the program fragment in 2a above be valid?
Show the content of the stack and the heap when the execution reaches the line labelled `A` again.

3. Below is our familiar `Point` classes augmented with a `toString` method.

```
class Point {
    private final double x;
    private final double y;

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    double distanceTo(Point otherpoint) {
        double dispX = this.x - otherpoint.x;
        double dispY = this.y - otherpoint.y;
        return Math.sqrt(dispX * dispX + dispY * dispY);
    }

    @Override
    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }
}
```

The `toString` method provides a way to output an object in a more meaningful way, rather than just a reference value. To illustrate using JShell,

```
jshell> /open Point.java

jshell> Point p = new Point(0, 0)
p ==> (0.0, 0.0)
```

You are also given the `Circle` class.

```
class Circle {
    private final Point centre;
    private final double radius;

    Circle(Point centre, double radius) {
        this.centre = centre;
        this.radius = radius;
    }

    boolean contains(Point point) {
        return centre.distanceTo(point) <= radius;
    }

    @Override
    public String toString() {
        return "Circle centred at " + this.centre +
            " with radius " + this.radius;
    }
}
```

We can define an array of five points as follows:

```
jshell> Point[] points = new Point[]{new Point(0,0), new Point(0,-1),
   ...> new Point(1,0), new Point(0,1), new Point(-1,0)};
points ==> Point[5] { (0.0, 0.0), (0.0, -1.0), (1.0, 0.0), (0.0, 1.0), (-1.0, 0.0) }
```

(a) Within JShell, define a method `countCoverage` that takes in a `Circle` object, and an array of `Point` objects. This method will return the number of points contained within the circle.

(b) Write single line tests in JShell to test the correctness of the method. For example, a circle centred at the origin with radius 1.0 contains all five points; a circle centred at $(0.0, -1.0)$ with radius 1.0 contains two points.