CS2030 Programming Methodology
Semester 2 2019/2020

23 January 2020
Problem Set #1 Suggested Guidance
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`?
   The abstraction barrier sits between the client and the implementer. Here class `P` *is the implementer, and* `Q` *is the client that makes use of the* `p`*, an object of* `P`*.*

   (b) What does this say about the concept of an "abstraction barrier"?
   The barrier is not broken when one one object of type `P` *accesses the instance variables of another type* `P` *object, since* `P` *is the sole implementer.*

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

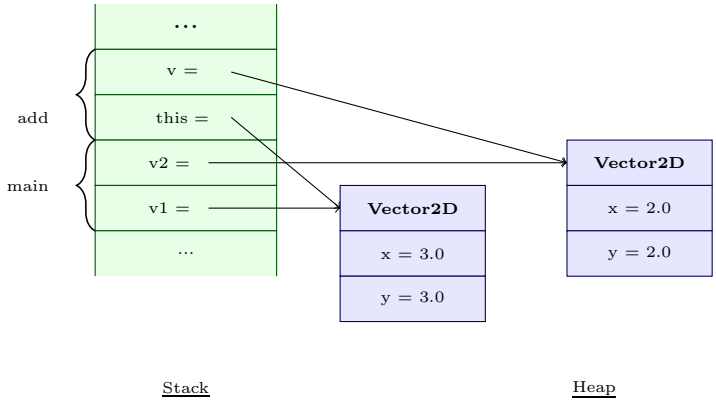
```
class Vector2D {
    private double x;
    private double y;

    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 in a `main` method, 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. Draw boxes around the stack frames of the methods `main` and `add`, and label them.



Unlike languages like C, Java has automatic memory management. The garbage collector "cleans up" or reclaims memory taken up by unreferenced objects in the heap.

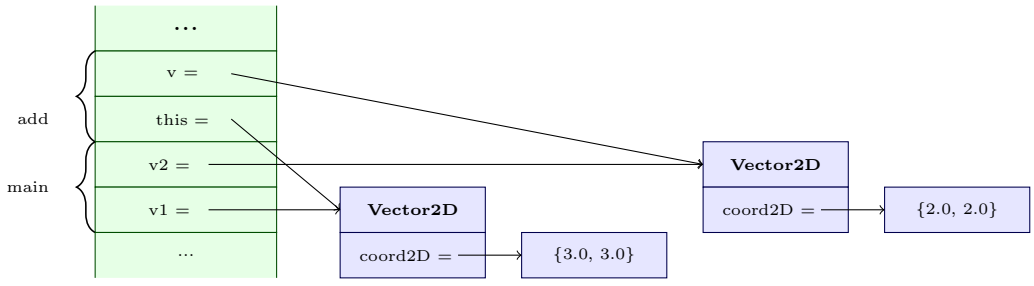
(b) Suppose that the representation of `x` and `y` have been changed to a double array:

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

    ...
}
```

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

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

    Vector2D(double x, double y) {
        this.coord2D = new double[]{x, y};
    }

    void add(Vector2D v) {
        coord2D = new double[] {
            this.coord2D[0] + v.coord2D[0],
            this.coord2D[1] + v.coord2D[1]};
    }
}
```

ii. Would the program fragment in 2a above be valid?
Yes, the program fragment is still valid. The lower-level implementation of how the x amd y coordinates are stored and operated on in `Vector2D` *is encapsulated from other clients.*
Show the content of the stack and the heap when the execution reaches the line labelled `A` again.



3. Below is the `Point` and `Circle` classes augmented with a `toString` method.

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

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

    double distance(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 + ")";
    }
}

class Circle {
    private Point centre;
    private double radius;

    Circle(Point centre) {
        this.centre = centre;
        this.radius = 1.0;
    }

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

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

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

Using JShell, we 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) Define a method `findCoverage` that takes in `points` as an array of `Point`. For each of the points, construct a unit circle (i.e. circle of radius 1.0) and find the coverage among all five points. A point is "covered" by the unit circle if it contains the point.

```
Circle centred at (0.0, 0.0) with radius 1.0 contains 5 points.
Circle centred at (0.0, -1.0) with radius 1.0 contains 2 points.
Circle centred at (1.0, 0.0) with radius 1.0 contains 2 points.
Circle centred at (0.0, 1.0) with radius 1.0 contains 2 points.
Circle centred at (-1.0, 0.0) with radius 1.0 contains 2 points.
```

```
int countCoverage(Circle circle, Point[] points) {
    int count = 0;
    for (Point point: points) {
        if (circle.contains(point)) {
            count = count + 1;
        }
    }
   return count;
}

void findCoverage(Point[] points) {
    for (Point point : points) {
        Circle c = new Circle(point, 1.0);
        System.out.println(c + " contains " +
                countCoverage(c, points) + " points.");
    }
}
```

(b) Now, define a `Main` class that uses the `Point` and `Circle` classes to solve the above problem. Specifically, the program reads in the first input as the number of points, reads in the set of points, and then outputs the coverage

A sample run of the program is given below. User input is underlined.

```
$ java Main
5
0 0
0 -1
1 0
0 1
-1 0
Circle centred at (0.0, 0.0) contains 5 points.
Circle centred at (0.0, -1.0) contains 2 points.
Circle centred at (1.0, 0.0) contains 2 points.
Circle centred at (0.0, 1.0) contains 2 points.
Circle centred at (-1.0, 0.0) contains 2 points.
```

```
import java.util.Scanner;

class Main {
    static Point[] readPoint() {
        Scanner sc = new Scanner(System.in);
        Point[] points = new Point[sc.nextInt()];
        for (int i = 0; i < points.length; i++) {
            points[i] = new Point(sc.nextDouble(), sc.nextDouble());
        }
        return points;
    }

    static int countCoverage(Circle circle, Point[] points) {
        int count = 0;
        for (Point point: points) {
            count += circle.contains(point) ? 1 : 0;
        }
        return count;
    }

    static void findCoverage(Point[] points) {
        for (Point point : points) {
            Circle c = new Circle(point, 1.0);
            System.out.println(c + " contains " +
                    countCoverage(c, points) + " points.");
        }
    }

    public static void main(String[] args) {
        Point[] points = readPoint();
        findCoverage(points);
    }
}
```

Note the use of `static` in front of all methods of the class Main. Remember that this static modifier is used to define members (methods and fields) that belong to the class, not to instantiated objects of the class.