CS2030 Programming Methodology II
Semester 1 2024/2025

Week of 2 – 6 September 2024
Problem Set #2
Abstraction, Encapsulation and Interface

1. Study the following classes `P` and `Q`:

```
 1:   class P {
 2:       private final int x;
 3:
 4:       P(int x) {
 5:           this.x = x;
 6:       }
 7:
 8:       P foo() {
 9:           return new P(this.x + 1);
10:       }
11:
12:       P bar(P p) {
13:           p.x = p.x + 1;
14:           return p;
15:       }
16:   }
17:
18:   class Q {
19:       P baz(P p) {
20:           return new P(p.x + 1);
21:       }
22:   }
```

   (a) Which line(s) above violate the `final` modifier of property `x` in class `P`?

   (b) Which line(s) above violate the `private` modifier of property `x` in class `P`? Relate your observation to the concept of an "abstraction barrier".

2. We modify the `Point` class to represent the coordinates with a `Pair` object.

```
class Point {
    private final Pair<Double, Double> coord;

    Point(double x, double y) {
        this.coord = new Pair<Double, Double>(x, y);
    }

    private double getX() {
        return this.coord.t();
    }
```

```
    private double getY() {
        return this.coord.u();
    }

    double distanceTo(Point otherPoint) {
        double dx = this.getX() - otherPoint.getX();
        double dy = this.getY() - otherPoint.getY();
        return Math.sqrt(dx * dx + dy * dy);
    }

    public String toString() {
        return "(" + this.getX() + ", " + this.getY() + ")";
    }
}
```

(a) Do `getX()` and `getY()` in `Point` violate *Tell-Don't-Ask*?

(b) Do `t()` and `u()` in the `Pair` class violate *Tell-Don't-Ask*?

3. During the lecture, we developed the following `Circle` to support both circles at a fixed location, as well as empty circles.

```
import java.util.Optional;

class Circle {
    private final Optional<Point> centre;
    private final double radius;

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

    Circle(double radius) {
        this.centre = Optional.empty();
        this.radius = radius;
    }

    public String toString() {
        return "Circle " +
            this.centre.map(c -> "at " + c).orElse("") +
            " with radius " + this.radius;
    }
}
```

Include the instance method `boolean isOverlap(circle)` that returns `true` if there is an overlap, or `false` otherwise.