SWEN20003
Object Oriented Software Development
Workshop 3

Eleanor McMurtry

Semester 2, 2020

Workshop

This week, we are learning more about how to effectively structure classes in a Java program, and more practice with using two important Java types: **arrays** and **strings**.

- **Encapsulation** refers to grouping objects' data with the methods that operate on this data.
- **Information hiding** refers to hiding attributes and methods from the user of the class, e.g. using the `private` keyword.
- **Delegation** refers to the process of assigning different responsibilites to different classes.
- An **immutable** object is one whose attributes cannot be changed after it is created.

Questions

1. Using the principle of **information hiding**, assign privacy modifiers (either `public` or `private`) to attributes and methods in the below class.

```
public class Drone {
    double homeX;
    double homeY;
    double x;
    double y;
    double altitude = 0.0;

    Drone(double homeX, double homeY) {
        this.homeX = homeX;
        this.homeY = homeY;
        x = homeX;
        y = homeY;
    }

    void flyUp(double amount) {
        altitude += amount;
    }

    void flyDown(double amount) {
        altitude = Math.max(altitude - amount, 0);
    }

    double distanceToHome() {
        return distance(x, y, homeX, homeY);
    }

    static double distance(double x1, double y1, double x2, double y2) {
        return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 -y2));
    }
}
```

2. Consider the below class:

```
public class Rectangle {
    private double left;
    private double top;
    private double right;
    private double bottom;

    public Rectangle(double left, double top, double right, double bottom) {
        this.left = left;
        this.top = top;
        this.right = right;
        this.bottom = bottom;
    }

    public double getLeft() {
        return left;
    }

    public double getTop() {
        return top;
    }

    public double getWidth() {
        return right - left;
    }

    public double getHeight() {
        return bottom - top;
    }
}
```

Using the principle of **delegation**, create a `Point` class and replace the attributes of `Rectangle` with instances of `Point`.

3. (a) Design and implement classes to represent channels airing on TV.

   A channel has a *name*, and broadcasts up to 5 *shows* each day. (For simplicity, assume they are the same every day). A show has a *name*, a *duration* (in minutes) an *air time* (in hours and minutes).

   A channel with less than 5 shows can have a show added to its broadcast list. When doing so, it should check that two shows are not scheduled to run at the same time (otherwise adding the clashing show does nothing). A channel can also cancel a show, removing it from the broadcast list.

   (b) Add a `getShow` method to your channel class. Given a time (in hours and minutes), it should return the show that is scheduled to be running at that time (or `null` if there is no such show).

   (c) Create a class to represent a network of up to 3 channels. Networks have a *name*, and channels can be added to or removed from a network. A network has a `getShows` method that returns all shows running at a particular time on any channel in the network.

   (d) Add a `lookupShow` method to your network class that takes a show and returns which channel that show is scheduled to run on. If there are multiple channels, only return the first that you find.

4. Consider the below class (the raw code is attached to Canvas).

```
public class Person {
    public String name;
    public double x;
    public double y;
    public String householdName;

    private static Person[] people = new Person[100];
    private static int peopleCount = 0;
```

```
    public Person(String name, double x, double y, String householdName) {
        this.name = name;
        this.x = x;
        this.y = y;
        this.householdName = householdName;

        if (peopleCount < 100) {
            people[peopleCount++] = this;
        }
    }

    public double distanceToPerson(Person person) {
        return Math.sqrt((x - person.x) * (x - person.x)
                + (y - person.y) * (y - person.y));
    }

    private Person[] peopleCloserThan(double distance) {
        int numCloser = 0;
        // Count how many people are close
        for (int i = 0; i < peopleCount; ++i) {
            if (distanceToPerson(people[i]) < distance) {
                ++numCloser;
            }
        }

        // Create an appropriately-sized array, and then fill it
        Person[] result = new Person[numCloser];
        int count = 0;
        for (int i = 0; i < peopleCount; ++i) {
            if (distanceToPerson(people[i]) < distance) {
                result[count++] = people[i];
            }
        }

        return result;
    }

    public int numCloseOutsideHousehold(double distance) {
        Person[] people = peopleCloserThan(distance);
        int count = 0;
        for (int i = 0; i < people.length; ++i) {
            // If they are not from this person's household, increment counter
            if (!people[i].householdName.equals(householdName)) {
                ++count;
            }
        }
        return count;
    }
}
```

(a) If there are any public attributes or methods that should be private according to the principle of **information hiding**, make them private instead.

(b) Using the principle of **encapsulation**, define a `Point` class with an x- and y-coordinate, and a method `double distanceTo(Point other)` to calculate the distance to another point.

(c) Using the principle of **delegation**, replace the `x` and `y` attributes of `Person` with an instance of `Point`. Update the methods of `Person` accordingly.

(d) Using the principle of **delegation**, define a `Household` class with an appropriate `equals` method. Each household has a *name* and up to 5 *people* (set in the constructor). Replace the `householdName` attribute of `Person` with an instance of `Household`. Update the methods of `Person` accordingly. (You may assume household names are unique, and you may need to add a setter for the household.)