Java Lab 1: Diving Right In

Java lets us program with *objects*. We use *classes*, one per file, to describe how these objects work -- a blueprint, if you will. Here is the code for a simple class that represents a two-dimensional point:

```
/**
 * A two-dimensional point class.
 */
public class Point2d {

    /** X coordinate of the point */
    private double xCoord;

    /** Y coordinate of the point */
    private double yCoord;

    /** Constructor to initialize point to (x, y) value. */
    public Point2d(double x, double y) {
        xCoord = x;
        yCoord = y;
    }

    /** No-argument constructor:  defaults to a point at the origin. */
    public Point2d() {
        // Call two-argument constructor and specify the origin.
        this(0, 0);
    }

    /** Return the X coordinate of the point. */
    public double getX() {
        return xCoord;
    }

    /** Return the Y coordinate of the point. */
    public double getY() {
        return yCoord;
    }

    /** Set the X coordinate of the point. */
    public void setX(double val) {
        xCoord = val;
    }

    /** Set the Y coordinate of the point. */
    public void setY(double val) {
        yCoord = val;
    }
}
```

This code *must* be saved in a file called `Point2d.java`, due to Java's requirements on class names and file names. A copy of this file is provided for your consumption; you might save a copy of it to your working directory and build off of it for this lab.

Recall that we can *instantiate* our class anywhere else in our code by calling any of the constructors we've defined, like so:

```
    Point2d myPoint = new Point2d();            // creates a point at (0,0)
    Point2d myOtherPoint = new Point2d(5,3);    // creates a point at (5,3)
    Point2d aThirdPoint = new Point2d();
```

Be careful: `myPoint != aThirdPoint`, even though their values are the same! This is because the equality operator `==` (and its inverse, the inequality operator `!=`) compare the two object *references* for equality. In other words, `==` returns true if the two references point to the *same* object. In this code, `myPoint` and `aThirdPoint` each refers to a different instance of the `Point2d` class, so `myPoint == aThirdPoint` returns `false`, even though their values are the same!

To test for value-equality and not reference-equality, we define a method of the `Point2d` class called `equals`, which takes another `Object` as an argument, and performs the appropriate tests for equality. Remember that the object being compared must be the correct type, and its member-fields must have the same values as well.

Before you start coding away...

Coding style is very important in any software project that you work on. Believe it or not, the *vast majority* of a successful program's lifetime is spent in the debugging and maintenance phases. In these phases of a product's lifecycle, well-documented and readable code becomes a huge asset, saving massive amounts of time.

Unfortunately, the value of good coding style is often learned only through painful experience... But, CS11 is a good opportunity to learn and practice good coding style. Before you start on this assignment, review the CS11 Java Style Guidelines. There is even a helpful style-checker on the CS cluster, that will report any issues with your program.

Your task:

1. Create a new class `Point3d` to represent, you guessed it, points in three dimensional Euclidean space. It should be possible to:

   - create a new `Point3d` described by any three floating-point (type `double`) values,
   - create a new `Point3d` at (0.0, 0.0, 0.0) by default,
   - access and mutate all three values individually, and
   - compare two `Point3d`s for value-equality using an appropriate `equals` method.

   It should not be possible to directly access the internal data members of any `Point3d` object.

2. Furthermore, add a new method `distanceTo` which takes another `Point3d` as an argument, and computes a double-precision floating-point approximation of the straight-line distance between the two points, and returns that value.

3. Create a second class called `Lab1` that exists primarily to house the static method `main`. Remember that `main` must be `public`, have a `void` return type, and accept an array of `String`s as an argument. Inside this class, add some functionality:

   - Input three ordered triples from the user, each representing the coordinates of one point in 3-space. Generate three `Point3d` objects from this data. (For now, you can assume that the user will not enter invalid data.)

     If you don't know how to get input from the user, you can use the function in this file. Put it as a static method in your `Lab1` class. Note that this method uses classes in the `java.io` package, which is not visible to your code by default. To make it visible, add this to the very top of your file:

     ```
     import java.io.*;
     ```

     This makes all classes in the `java.io` package visible to your `Lab1` code. (You don't have to do this with the classes in the `java.lang` package, since those are made available to your classes by default.)

   - Write a second static method `computeArea` which takes three `Point3d`'s and computes the area within the triangle bounded by them. (You may wish to use Heron's formula.) Return this area as a `double`.

   - Use the data and code you gathered and wrote above to determine the area and print that out for the user's consumption.

     Before you call `computeArea`, however, test for value equality between all of the three `Point3d`'s. If any pair of points is "equal", report this to the user and do not compute an area.

4. Compile both of your source files together:

   ```
   javac Point3d.java Lab1.java
   ```

   and then run your `Lab1` program, testing it with several sample triangles.

5. When you are finished with lab 1, you can submit your files on the csman website.

[end lab 1]