Lab 1, CSC/CPE 203 - Java collections and control

Orientation

This lab allows you to practice Java syntax related to collections and control (Part 1) and basic class structure and methods (Part 2). Those already familiar with Java should finish the lab and then offer to help others.
This lab is due at the end of lab period in week 1.

Objectives

- To be able to iterate through different Java collections with various control loops (part 1)
- To be able to write your own class with specified methods (part 2).
- To be able to use unit testing, including writing your own tests for your code (part 1 and 2)
- To be able to write a very simple UML diagram of a class using yEd

Resources

- Your peers
- Your instructor
- The Java Standard Library - Bookmark this web page, you will be using the Java Standard Library documentation regularly this quarter.

Retrieve the files provided for this lab from Polylearn under the lab assignment folder.

Setup

For this lab you are "required" once again to compile and run Java from the command-line.

To compile multiple Java files, you can run the following command (you can also specify specific files, as you prefer).

```
javac *.java
```

NEW For this lab, we will be using the JUnit library to help us test the correctness of our code. Thus, you will need to tell the compiler where the JUnit library is. There are two ways to do this, with the second being the better.

The first includes the library in the "class path" for the given call to the compile. This must be repeated each time you compile the files and when you run the test cases (see below).

```
javac -cp /home/akeen/public/junit-4.12.jar:/home/akeen/public/hamcrest-core-1.3.jar:. *.java
```

```
java -cp /home/akeen/public/junit-4.12.jar:/home/akeen/public/hamcrest-core-1.3.jar:. org.junit.runner.JUnitCore TestCases
```

Instead, it is much more convenient to configure the system to add the library to your class path each time you log in. Add the following line to your `.mybashrc` file in your home directory.

```
export CLASSPATH=/home/akeen/public/junit-4.12.jar:/home/akeen/public/hamcrest-core-1.3.jar:${CLASSPATH}:.
```

After doing so (and either restarting your shell/terminal or logging out and back in or typing "source .mybashrc"), you need only execute the following to compile and run.

```
javac *.java
```

```
java org.junit.runner.JUnitCore TestCases
```

Specifications

Part 1: Java Control Constructs and Collection Introduction

In the provided `part1` subdirectory, edit each of the given files to complete the implementations and tests. Comments in each file will direct the edits that you are expected to make. (Start by looking at the file TestCases.java).

It is recommended that you use the order of the unit tests in `TestCases.java` as a guide for the order in which to complete the methods. This order corresponds to the increasing complexity of the methods.

The output for a failed test is incredibly verbose (a stack trace is printed). You can see the specific error at the numbered line above each stack trace. *As you fix the code one file at a time, re-run the code to make sure that the number of errors is decreasing!*

Part 2: Classes and Objects

For this part you will implement a simple class (named `Point`) representing a point in two-dimensional space. Do this by creating a new file named: Point.java and put all code specified below. This class must support the following operations (including a single constructor).

- `public Point(double x, double y)`

  Constructor.

- `public double getX()`

  Returns the x-coordinate of this point.

- `public double getY()`

  Returns the y-coordinate of this point.

- `public double getRadius()`

  Returns the distance from the origin to the point.

- `public double getAngle()`

  Returns the angle (in radians) from the x-axis in the in range of -*pi* to *pi*.

- `public Point rotate90()`

  Returns a newly created Point representing a 90-degree (counterclockwise) rotation of `this` point about the origin. Consider drawing a picture for a point not on an axis as a guide for how you might implement this operation (hint: there is a solution that does not require any sophisticated computations).

  Note that you are not expected to define equality for `Point` objects at this time; as such, you will need to test the components (x and y) for the expected values.

Update the given `TestCases.java` file, in the `part2` directory, to verify that your implementation is working. **You will be asked to show a test case for every method you wrote.** You will see some existing test cases that are used to verify some design/style aspects of your implementation, which you do not need to change or modify.

Part 3: UML

See the attached pdf description of the UML portion of this lab.

Demo

Demonstrate **both** parts of your working program to your instructor at once. Be prepared to show your source code. You will need to demo your working code the last lab day in week1. (Partial credit available up to instructor discretion).

Submission

Please also submit your code for record keeping using Polylearn.