1. Goal

The goal of this assignment is to get back into the swing of things Java, especially its object-oriented features including Abstract Data Types (ADTs), Interfaces, and Abstract Classes. (Later we will add Inheritance, Generics, and the JCF!)

2. Problem Statement [client’s statement of their need, often referred to as the requirements]

The lads have been tasked with implementing code to manipulate Polar and Cartesian points. Patrick said that Sponge Bob didn’t get the point. Sponge Bob said it was ok because he could ask you to.

3. Analysis [What is the client’s problem?]

The client needs two concrete classes PolarPoint and CartesianPoint. A polar point maintains an angle \( \theta \) and a radius \( \rho \), while a Cartesian point maintains \( x \) and \( y \) coordinates. Both inherit from a common abstract class [I’ll let you pick a good name for this abstract class]. Furthermore, the program needs to include two separate interfaces for translating and rotating points. [Each interface should be in its own file.] [Given the interfaces presented in Chapter 1, make sure to use appropriate names for your interfaces].

For this first assignment there is no user interface. [A user interface is the way the program interacts with its user. Common examples include a GUI (graphical user interface) and a CLI (command line interface, which uses menus and command-line options).] For this assignment, instead of a user interface, hard-code a test driver. [Name your test driver Tester.java for the benefit of the testing script, oh, and your grade.]

Digging deeper into understanding “What”, translating a Cartesian point by the amount \((by_x, by_y)\), adds \(by_x\) to the point’s \(x\) value and \(by_y\) to its \(y\) value. Rotating a Cartesian point by \(\theta\) updates \(x\) to be \(x \times \cos(\theta) - y \times \sin(\theta)\) and \(y\) to be \(y = x \times \sin(\theta) + y \times \cos(\theta)\). In a similar faction, rotating a polar point by \(\theta\) degrees adds \(\theta\) to the point’s angle. Translating a polar point by the amount \((by_x, by_y)\) can be accomplished by converting the point to a Cartesian point, doing the translation, and then converting it back.

3. Analysis [What is the client’s problem?]

Q: [hey client,) What kinds of points?
A: Patrick said polar points, the Sponge Cartesian. You get to implement both!
[Notice that the client does not describe these two as ‘concrete classes’, which is the term that a software engineer would use. The client also does not explicitly mention that these two share a common core, which an engineer captures using an abstract class.]
Q: What [state] information about each kind of point is important?
A: A polar point has an angle $\theta$ and a radius $\rho$, while a Cartesian point has $x$ and $y$ coordinates.

Q: What do you want to be able to do with the points?
A: Translate and rotate them. [Notice that the client does not use the term ‘interface’, so you, as the software engineer, need to recognize that these are actions that a point should implement.] [It is still the case that each interface gets its own appropriately-named file.]

Q: What does it mean to translate a point?
A: Translating a Cartesian point by the amount ($by_x$, $by_y$), adds $by_x$ to the point’s $x$ value and $by_y$ to its $y$ value. Translating a polar point by the amount ($by_x$, $by_y$) can be accomplished by converting the point to a Cartesian point, doing the translation, and then converting it back.

Q: What does it mean to rotate a point?
A: Rotating a Cartesian point by $\theta$ degrees updates $x$ to be $x \times \cos(\theta) - y \times \sin(\theta)$ and $y$ to be $y = x \times \sin(\theta) + y \times \cos(\theta)$. Rotating a polar point adds $\theta$ to the point’s angle.

Q: What commands does the program support?
A: None. [For Assignment 1, just hard-code a unit test driver. Name it Tester.java for the benefit of the testing script.]

4. Design [How]

[Given the understanding from the analysis, design considers how an engineer plans to solve the problem.] In this case the design comes down to the use of abstract classes, interfaces, and concrete classes.

What to hand in

1. By 6:00am Friday please push your plateau schedule and unit test plan of at least eight tests (four for each kind of point).
2. A printout of your UML class diagram (I used ArgoUML, via the command argo).
3. A well-formatted 2-up printout of your source code. Here are some useful incantations:
   ```
   man <command> (for example, man a2ps)
a2ps -T 4 -q -Avirtual -2 -o mycode.ps <files>
gv --geometry 1020x770+05+05 --orientation=landscape mycode.ps
   ps2pdf mycode.ps
   xpdf mycode.pdf
   (once happy) lpr mycode.ps
   ```
4. A GitHub repo that includes (you must use these names as the grading script assumes them!)
   * README.md with the sections definitions, plateau schedule, and unit test plan,
   * mycode.pdf (ensure that you are happy with the formatting by using xpdf), and
   * your Java source code with main in Tester.java.

In README.md under the heading “definitions” include a one sentence definition of how the following are used in your code: abstract class, concrete class, inheritance, and interface.

Assignment Requirements [ part of being a course rather than part of software development ]

- Each source code file must start with
  ```
  /**
   * This is my code! It’s goal is to ...
   * CS 312 - Assignment 1
   * @author Your Name
   * @version 1.0 9/10/2019 (a version number followed by the date)
   */
  ```
- Use clear documentation and careful formatting. Be consistent in indentation and alignment of braces. Each open brace “{” must be on its own line.
Notes

• Your code can contain no more than one static method.
• Consider using the testing tool JUnit (We will starting using JUnit in our fourth week).
• Also, consider using test-driven development: write a test, write code to pass it, repeat.
• Avoid the use of the instanceof keyword. Such often indicates poor OO thinking.
• ArgoUML can create UML diagrams and skeleton Java files based on the UML.
• Error checking is a plus.
• Reread the general notes regarding style, braces, and header comments.
• Work on declarative naming. For example, rather than ConvertToPolar(...) a declarative name is asPolar(...)
• While not really the OO way, declare attributes private for now.
• Don’t comment the obvious :)
• Use a tab stop (indent) of either 2 or 4 spaces ... no more!
• The GitHub classroom invite is https://classroom.github.com/a/4VGSd7CZ
• All concrete classes must include reasonable toString methods

Initial Grading Rubric

definitions missing from README creation of unnecessary objects
plateau schedule missing from README poor naming
fails to compile / execute extra instance variables
method headers (see general notes) not 2up
layout of code (presentation matters! ) no pdf in repo
use of get/set including derived files (e.g., *.class) in repo
more than one static method commenting the obvious

Advice from those who came before you

(1) I had a difficult time getting started because I was having trouble even thinking of where to start. I read and reread the relevant chapter in the book a few times and then everything began to click. After that I was able to make steady progress on all of the different classes and interfaces.

(2) After looking at your solution to Assignment 1, I now realize that it would have been more efficient to make the interfaces of translation and rotation more generic so that I didn’t have excess methods that weren’t needed in PolarPoint and CartesianPoint.

(3) I needed to see how to use “this” instead of creating a get() method.
[Important: your code must not include any get() or set() methods.]

(4) I had a hard time conceptualizing a method for translate and rotate that could work for both.

(5) I think my [mistake was] that I tend to just jump into coding and try to figure out how to accomplish each step individually without considering the program/assignment as a whole. I can attempt to combat this failing of mine by thinking before coding about both the steps individually but also how they relate to the program as a whole.