Lab #6 – Drawable/Scaleable Shapes II, using inheritance

Total Points: 50

**Introduction**

In the last lab you created several shapes that implemented the Drawable and Scaleable interfaces. You also implemented the DrawableObjectList that was a collection of Drawable shapes. In that lab you likely noticed a good bit of repeated code. In this lab you will use inheritance to implement a wider variety of shapes with less repetition of code. Specifically, you will implement and test classes named Line, Text, Rectangle, Square, Ellipse and Circle that represent the corresponding objects.

**Getting Started**

1. Create your GitHub repository and import the code into Eclipse as described on the "How to…" webpage, available from the course homepage.

2. As in some previous labs, remove any compilation errors in the starter code by following the instructions for "Adding the JUnit4 library to a project in Eclipse" the course "How to…" webpage.

**Design**

Provided with the lab is the class Point that implements the Drawable interface, familiar from the previous lab. A Point represents a singled colored pixel at a given $(x,y)$ coordinate. The Point class provides implementations of all the methods in the Drawable interface. It also contains two methods move() and translate() that change the position of the point in different ways (see the documentation in the Point class for more information). Each of the new classes that you create will be a subclass of Point (or a subclass of one of its subclasses).

The Point class and all of its subclasses form what is called an *inheritance hierarchy*. The inheritance hierarchy that you will be implementing can be represented visually as shown below. The Drawable and Scaleable interfaces are also shown. A solid arrow from one class to another indicates a subclass/superclass relationship (e.g. Line is a subclass of Point). Dashed lines indicate that a class implements an interface (e.g. Ellipse implements the Scaleable interface).



**The Assignment**

Your assignment for this lab is to implement and test classes named Line, Text, Rectangle, Square, Ellipse and Circle that represent the corresponding objects. Once they are complete you must create another class that displays a picture or animation using at least one of each type of object.

The classes that you are creating must be implemented using the inheritance hierarchy described above in the Design section. Each class should inherit as much functionality as possible from its superclass. Each class should also override any inherited methods that need to behave differently in the subclass. Every class must have a Javadoc comment at the top describing the purpose of the class. Every method and constructor in every class must have a Javadoc comment describing its functionality. Each class must also have an associated JUnit test class that tests its functionality (although as with the previous lab, draw() methods need not be tested). Note that once a piece of functionality is tested in a superclass it does not need to be tested again in any subclasses. Thus, for subclasses only new and overridden methods need to be tested.

**Submitting your solution**

As usual, push your code to GitHub regularly for backup purposes and push your final version to submit the assignment. In addition, as usual, submit your lab report via Moodle. The lab report for this lab will consist only of the self-assessment report.