CSCE 314 Programming Languages – Fall 2015
Hyunyoung Lee
Assignment 6
Assigned on Friday, October 30, 2015

Electronic submission to eCampus due at 9:00 a.m., Wednesday, 11/11/2015
Signed *coversheet* due at the beginning of class on Wednesday, 11/11/2015
If you do not turn in a signed coversheet your work will not be graded.

"On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment."

Typed or printed name of student	Section (501 or 502)
Signature of student	UIN

Note 1: This homework set is *individual* homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually.

Note 2: Turn in one `yourLastName-yourFirstName.tar` or `yourLastName-yourFirstName.zip` file on eCampus, nothing else. What to submit is detailed below.

Note 3: All Java code that you submit must compile without errors using `javac` of Java 8. If your code does not compile, you will likely receive zero points for this assignment.

Note 4: Remember to put the head comment in your files, including your name and *acknowledgements of any help received* in doing this assignment. Again, remember the honor code.

You will earn total 120 (plus 30 if you do the extra credit problem) points.

---

1. Inheritance and dynamic dispatching. A `Shape` class represents a geometric figure in some coordinate position. `Shape` allows for finding out its position and area with the methods `Point position()` and `double area()`. `Point` is a class that can represent a two-dimensional coordinate.

The `Shape` class must be an abstract class, from which you will derive five subclasses: `Triangle`, `Rectangle`, `Square`, `Circle`, and `LineSegment` to represent different kinds of shapes. A triangle is defined by three points. A rectangle is defined by two points, the upper-left corner and the lower-right corner. A square is defined by its upper-left corner, and the length of a side. A circle is defined by the center point and the radius. A line segment is defined by two end points. The specification of each shape is of the following syntax (`e` means the empty string).

```
<shapes> ::= <shape> ; <shapes> | e
<shape>  ::= <triangle> | <rectangle> | <square> | <circle> | <linesegment>
<triangle>    ::= t <point> <point> <point>
<rectangle>   ::= r <point> <point>
<square>      ::= s <point><number>
<circle>      ::= c <point> <number>
<linesegment> ::= l <point> <point>
<point>  ::= <number> <number>
```

`<number>` is anything that Java can interpret as a number. For example, the following string specifies a rectangle, a circle, and a line segment, each followed by a semicolon as a delimiter.

```
r 0 0 1 1; c 3 3.1 0.1; l 0.0 1.1 2 5;
```

The first `<point>` in each `<shape>` is considered to be the *position* of the shape.

Tasks

1. Implement class `Point`.
2. Implement class `Shape`. It will be convenient to store one point in the base class `Shape`, as each shape will need at least one point that will serve as the position of the shape.
3. Implement five classes: `Triangle`, `Rectangle`, `Square`, `Circle`, and `LineSegment` for representing different kinds of shapes. Each of the five classes should inherit from the class `Shape` and define `area()` appropriately.
4. Implement a class `AreaCalculator` with one static method `calculate(Shape[] shapes)` that will calculate the total area of an array of shapes.
5. Implement a class `Main`, of which the `main()` method will accept command line arguments and do the following:
   (a) If the first argument is "R" and the second argument is a number (say n), `main()` constructs and stores in an array n different shapes selecting the shape and the point(s) (and the number in case of square or circle) at random as follows: Each point of each shape should be selected from the square region whose corners are $(0, 0)$ and $(100, 100)$. The radii of any circles and the side lengths of any squares should be selected from the interval [0, 100].
   (b) If the first argument is "S" then the specification of shapes are given in the second argument as a string, the format of which is as shown above. Process the input string according to the specification, constructing the shapes, and storing them in an array.

   Then, `main()` prints out what kind of shapes it constructed (and stored in an array) and the total area of the shapes.

2. Defining equality. Two different shapes are equal if and only if they are of the same kind, their position is the same and the geometric figures they represent are equal. Textbook Section 3.8 discusses implementing equality.

Tasks

1. Implement the `equals` method for `Shape` and all of its derived classes.
2. Override `hashCode` for these classes (as you should whenever you override `Object`'s `equals` method) when necessary.

3. Comparison. Two shapes can be compared based on area, so that shape `a` is less than or equal to shape `b` if and only if `a`'s area is less than or equal to `b`'s area.

Tasks

1. Make this ordering the *natural ordering* of `Shape` and all its derived classes. (Sections 4.1 and 21.3 discuss natural orderings and making classes comparable).
2. Extend the implementation of your `Main.main` so that it also prints out the shapes in an increasing order according to your natural ordering. To be able to print out shapes, add the `toString` method to each of the shape classes.
3. Explain why the above ordering is *not* a good choice for a natural ordering of the shape class. Write your answer in a text file `natural-ordering-answer.txt` and include the file in your a6 zip folder (or tar ball).

4. Extra credit (30 Points) Add a method `draw` to the shape classes that draws the shape on a window, and extend the `main()` to display the shapes (in addition to the output it was already producing).