CSci 1933 Lab 4

October 13, 2015

1. Introduction

The purpose of this lab is to have you create classes that illustrate basic Object-oriented programming techniques, including inheritance and polymorphism, as well as introduce you to the ArrayList class.

2. Your task

In today's lab, you will implement two classes.
To complete this lab, do the following:

2.1. In IntelliJ, create a new Java project, called Lab5..

2.2. Import the following files into our Lab5 project.

- Shape.java The abstract class that you will extend. Place this in the src/ directory.
- ShapeTest.java The JUnit test for your Square and Circle classes that you will implement. Place this in the src/ directory.
- ShapeSorter.java The class containing the main() method for this project. Place this in the src/ directory.

2.3. Add JUnit to your project. To do this, do the following:

- Select “Project structure” from the “File” menu.
- Go to the "Libraries" group, click the little green plus (look up), and choose "From Maven...".
- Search for "junit" -- you're looking for something like "junit:junit:4.8.1" (latest stable option), and click OK.

2.4. In your src/ directory, create two new classes called Circle and Square.

- Circle will need an instance variable of type double to store its radius.
- Square will need an instance variable of type double to store the length of its sides.
- Both will need constructors that take in a double as an argument, and assign it to their instance variables mentioned above.

2.5. Circle and Square should both extend the Shape class provided. When one class extends another class, it is considered a subclass. We want Circle and Square to be subclasses of Shape. To do this, add the keywords extends followed by Shape

to the class declaration. For example, if we had a class called Triangle, it would look like this:

```
public class Triangle extends Shape {
```

Because Shape is an abstract class, any subclasses that aren't abstract must implement its abstract methods. Shape has two abstract methods that you must implement:

- ```
  public double getArea() {
      ...
  }
  ```
  getArea() should return the area for the given shape. You can use Math.PI for the value of pi, and Math.pow() to square numbers. Math.pow() takes two arguments and returns the value of the first argument raised to the power of the second argument. So to square a number x, you would use Math.pow(x,2). Just in case you forgot, the area of a circle can be determined by pi * r^2.

  > **Note** Remember them that we are using **static** methods built into a Math class that is included in the Java Class Library.

- ```
  public String getShapeName() {
      ...
  }
  ```
  getShapeName() should return the name of the shape: "Circle" and "Square".

Run the unit tests in ShapeTest.java to ensure that you've implemented these methods correctly (testShapeCompareTo() will fail until you finish step 2.7).

2.6. Lets now examine the main() method inside of ShapeSorter. At the start of the method a List is declared and instantiated. The < ... > after List and LinkedList, defines the list to hold objects of the type Shape. In Java this is called Generics.

```
List<Shape> shapes = new LinkedList<Shape>();
```

We want to use the ArrayList implementation of List instead of LinkedList. Replace the java.util.LinkedList import at the top of the file with java.util.ArrayList. Then change the instantiation to call ArrayList's constructor instead of LinkedList's.

Because shapes is declared as a List, we don't need to change any other code for this. Common methods like adding and removing elements, iterating through elements, and moving elements are provided by the List interface. This allows us to code to the interface, and not worry about the implementation.

Now run the main() method and you should see the list of shapes printed out. Notice that they are not sorted by their area, despite having called Collections.sort() on the list.

Hint Instead of manually adding the imports, you can use IntelliJ's inherent functionality for doing this. Once you declare a variable and have not imported the required package, you'll see the variable name in red text along with a light bulb. You can either click on the bulb or hit Option (Mac)/Alt(Windows) + Enter to get a list of recommendations. Not only for imports, you can save a lot of manual work you use the IntelliJ suggestions that pop up when there's a red underline in IntelliJ.

2.7. Finally we need to correct the compareTo method in Shape, so that our list of shapes can be sorted. The compareTo method comes from the Comparable interface, which Shape implements. It currently looks like this:

```
/**
 * This method is used for sorting shapes by their area.
 *
 * @return less than 0 if s has a larger area than this
 * shape, greater than 0 if s has a smaller area than
 * this shape, and 0 if s has the same size area as
 * this shape.
 */
public int compareTo(Shape s) {
    return 0;
}
```

Note Have a look at the online Java documentation for the Comparable interface and compareTo(). You can google "Comparable Java" to get there.

To complete this lab, all of the JUnit tests should pass, and the main() method should display a list of shapes sorted by area.

---END---

---Solutions will be posted on moodle page 30 minutes before the lab completion time---