Lab Exercise 6
CS 2334

February 21, 2019

Introduction

In this lab, you will experiment with using inheritance in Java through the use of abstract classes and interfaces. You will implement a set of classes that represent various shapes. In addition, your implementation will facilitate the comparison of shape objects, even when they are different shapes. You will also write unit tests to ensure that your code works appropriately.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. Create and extend abstract classes and methods
2. Use interfaces to define standard behavior across multiple classes
3. Appropriately select an abstract class or interface based on the requirements of the solution

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Check to see if the EclEmma plug-in is installed:
    (a) *Help* menu: select *About Eclipse Platform / Installation Details*
    (b) You will see a list of installed plugins. If *EclEmma* is on the list, then you are done.
2. If not already installed, then install the EclEmma plug-in:
    (a) *Help* menu: select *Install new software*
    (b) *Work with* box: enter `http://update.eclemma.org`
    (c) Select *EclEmma* from the list. Click *Next*
    (d) Read and accept the license agreement. Click *Finish*
3. Download lab 6 from canvas.
4. Import into your Eclipse Workspace:
    (a) *File* menu: *Import*
    (b) Select *General/Existing Projects into Workspace* and then click *Next*
    (c) *Select archive file*: browse to the lab6.zip file
    (d) Click *Finish*
5. Import JUnit into the project
    (a) Click on lab6
    (b) Click on Project (in the eclipse menu)/Properties
    (c) Select *Java Build Path*
    (d) Click *Libraries* tab
    (e) If this project is imported: you may need to select *JUnit4* and click *Remove*
    (f) Click *Add Library*
    (g) Select *JUnit*. Click *Next*
    (h) Select the most recent version (*JUnit 4*)
    (i) Click *Finish*

(j) Click *OK*

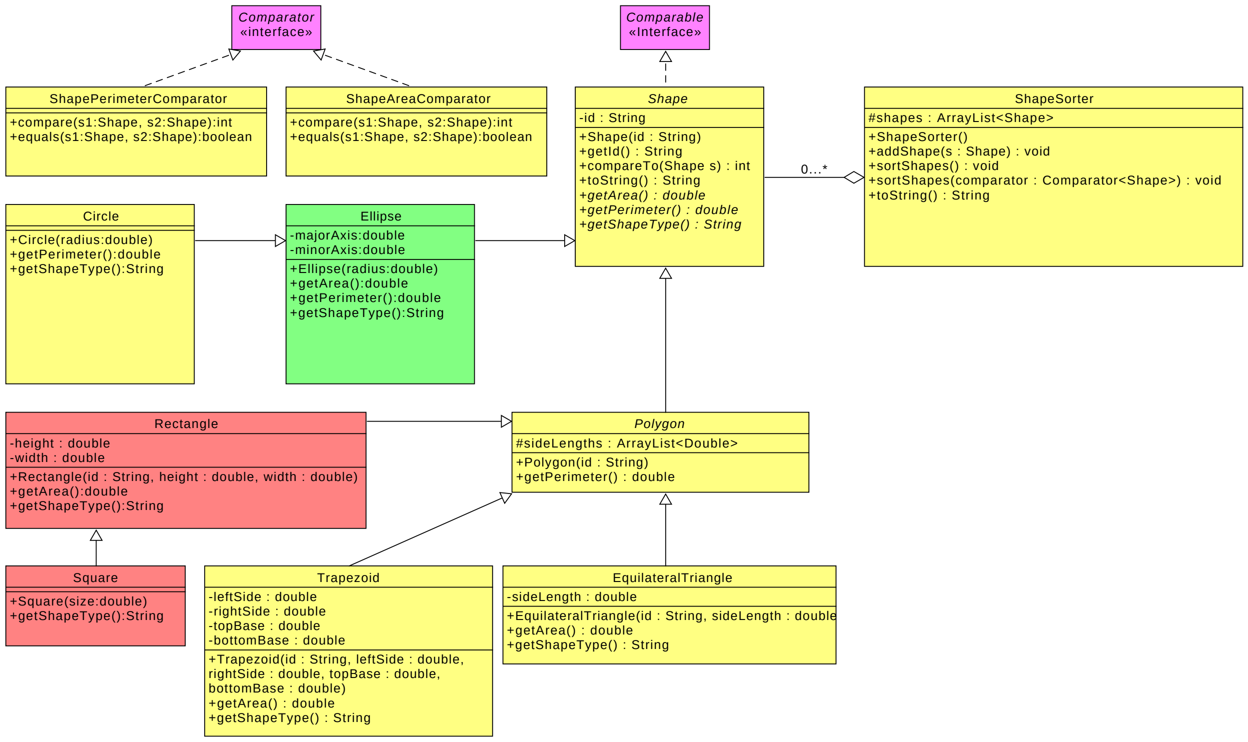
6. Carefully examine the code for the classes. Note the specification for the code detailed in this lab writeup and in the code commentary.

7. Add a TODO.java file at the top level of your project, following the instructions for todo lists posted on canvas. You will need to fill a todo list out for each lab.

8. (Optional, but recommended) add the TodoChecker.java file to your default package. Follow the instructions on canvas to properly configure your project to run the program.

Representing Different Shapes

Below is the UML representation of a set of classes that represent various shapes. Your task will be to implement this set of classes and an associated set of test procedures.



The classes in italics represent abstract classes or interfaces. The concrete child classes must implement all methods from the abstract parent classes In this lab, **Shape** and **Triangle** are the abstract classes.

The coloring of the classes indicates what work you need to do to complete the lab assignment. This is done for your convenience; you can also follow TODOS and the other lab instructions to understand what you need to do. The colors indicate as follows:

1. Green: A class/interface that is completely implemented. You do not need to edit these.

2. Yellow: A class/interface that is partially implemented. You will need to look for TODOs and read the writeup to determine what else is needed to complete the class.
3. Red: A class/interface that is completely unimplemented. You will need to create this class and ensure that it is fully functional.
4. Purple: This is a class/interface provided in standard Java. You can look online for documentation for these classes. You should never attempt to create these classes. Doing so will cause many errors.

The line from **Shape** to **Comparable** indicates that **Shape** must implement the **Comparable** interface. Similarly, the **ShapeAreaComparator** class must implement the **Comparator** interface. **ShapeAreaComparator** compares shapes based on their area, and **ShapePerimeterComparator** compares shapes based on their perimeter. Shape itself provides a way to compare shapes by implementing the comparable interface. These are interfaces provided by standard Java. You should not attempt to create these interfaces.

Lab 6: Specific Instructions

Start from the class files that are provided in lab6.zip. Documentation on what the code should do is laid out in the given javadoc.

1. Create a new Java class (or modify a provided one) for each object class described in the UML diagram
   - Be sure that the class name is exactly as shown
   - You must use the default package, meaning that the package field must be left blank. You may need to drag and drop files into the folder labeled "src" if your code is not already in the default package.
   - Follow the example code given to you to better understand what you should be doing. Make sure that each class implementing the abstract methods getArea() and getPerimeter() calculate the correct values. getShapeType() should return a String that is the same as the class name. Don't overthink the code for the shapes. Use super constructors wisely, use basic geometry, and don't add anything that's not in the UML.
2. Implement the attributes and methods for each class

    - Use the same spelling for instance variables and method names as shown in the UML
    - Do not add functionality to the classes beyond what has been specified
    - Don't forget to document as you go!

3. Edit the *ShapeTest* and *ShapeSorterTest* classes and JUnit to thoroughly test all of your code
    - You will need to import org.junit.Assert and org.junit.Test to get junit running. You will notice that the Assert.assertEquals() format is the same as what you have been working with.
    - You need to convince yourself that everything is working properly
    - Make sure that you cover all the classes and methods while creating your test. Keep in mind that we have our own tests that we will use for grading.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Tuesday, February 26th.
- You must submit your implementation to the *Lab 6* area on the Web-Cat server. Make sure that you export your project from eclipse properly and upload it to the server.

References

- The API of the interfaces used for comparisons can be found at:
  `https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`
  `https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html`
- Information for the area/perimeter of an equilateral triangle: `https://en.wikipedia.org/wiki/Equilateral_triangle`

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 60 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Design/Readability: 20 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names
- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

Github: 10 points

You will need to use github when developing your project. The grader will check that you have made several commits (¿= 5) while developing your lab. Commits should have good messages. You will link your github repo on canvas.

Lab Plan: 10 points

Your todo.txt file will be checked by the webcat unit test to verify its format. A grader will also asses the quality of your lab plan. Your plan should have at least 5 objectives that should be relatively granular (e.g. an objective of ”finish lab” is much too large and not a good tasks in a plan). The grader will ensure that your predicted and actual times are reasonable (e.g. not 20000 minutes).