Lab Exercise 6
CS 2334

February 20, 2018

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

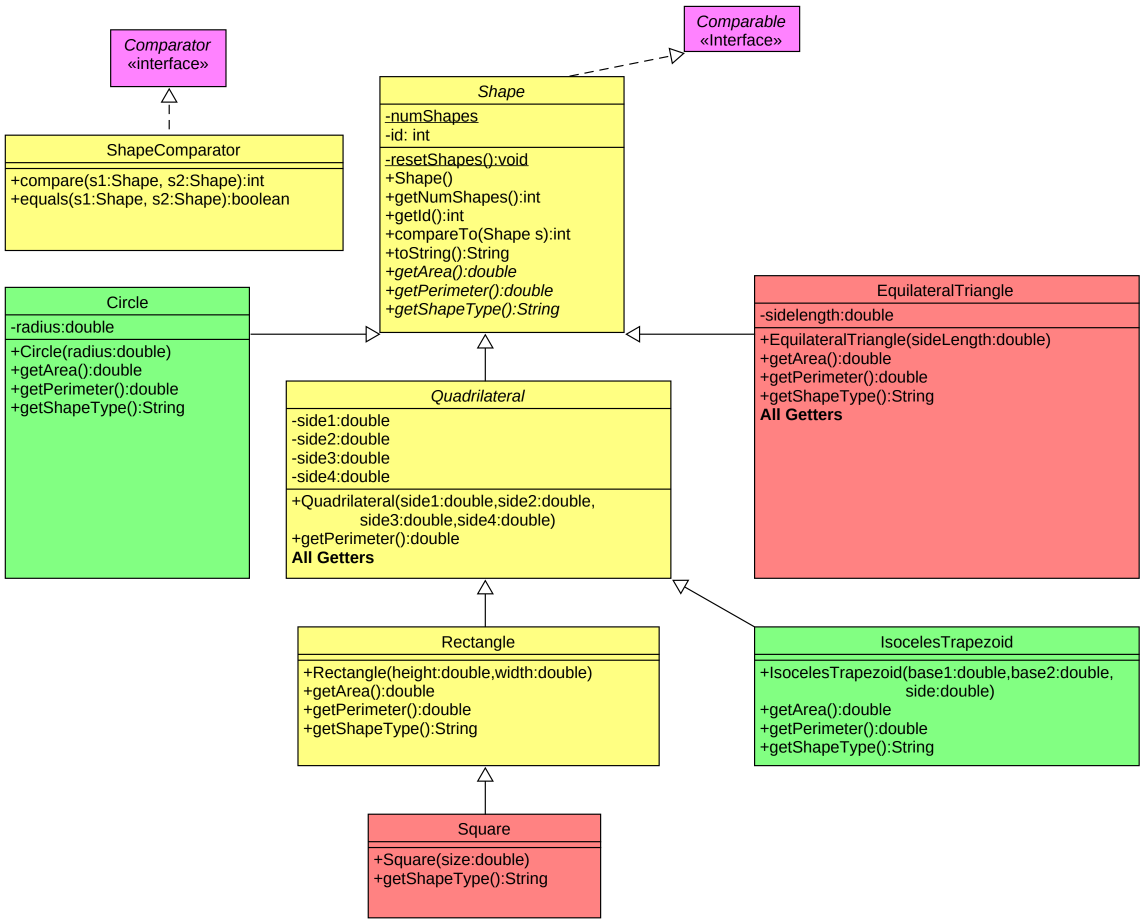
This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab6 implementation into your eclipse workspace.
   (a) Download the lab6 implementation from canvas.
   (b) In Eclipse, select *File/Import*
   (c) Select *General/Existing projects into workspace.* Click *Next*
   (d) Select *Select archive file.* Browse to the lab6.zip file. Click *Finish*

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 JUnit 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, **Shapes** and **Quadrilateral** 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 **ShapeComparator** class must implement the **Comparator** interface. **ShapeComparator** compares shapes based on their perimeter. These are interfaces provided by standard Java. You should not attempt to create these interfaces.

All instances of a **Shape** are given a unique int *id*. These are to be assigned by the **Shape** constructor. The instance of a shape is assigned an *id* of 0 (zero); the next is assigned 1.

Lab 6: Specific Instructions

Start from the class files that are provided in lab6.zip.

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.

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* class and JUnit to thoroughly test all of your code
    - 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.

Final Steps

1. Generate Javadoc using Eclipse.
    - Select *Project/Generate Javadoc...*
    - Make sure that your project is selected, as are all of the classes the classes.
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*
2. Open the *lab6/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.
3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, February 23nd.
- Use the same submission process as you used in lab 4. You must submit your implementation to the *Lab 6* area on the Web-Cat 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: 45 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 a product of the fraction of **our tests** that your code passes and the fraction of **your code** that is covered by *your tests*. In other words, your submission must perform well on both metrics in order to receive a reasonable grade.

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 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 (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

If you do not submit compiled Javadoc for your project, 5 points will be deducted from this part of your score.

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).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late. For a submission to be considered *on time*, it must arrive at the server by the designated minute (and zero seconds). For a deadline of 9:00, a submission that arrives at 9:00:01 is considered late (in this context, it is one minute late).

After 15 submissions to Web-Cat, you will be penalized one point for every additional submission.

For labs, the server will continue to accept submissions for three days after the deadline. In these cases, you will still have the benefit of the automatic feedback. However, beyond ten minutes late, you will receive a score of zero.

The grader will make their best effort to select the submission that yields the highest score.