Lab Exercise 6
CS 2334

September 28, 2017

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 3D shapes. In addition, your implementation will facilitate the comparison of shape objects, even when they are of different shape classes.

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

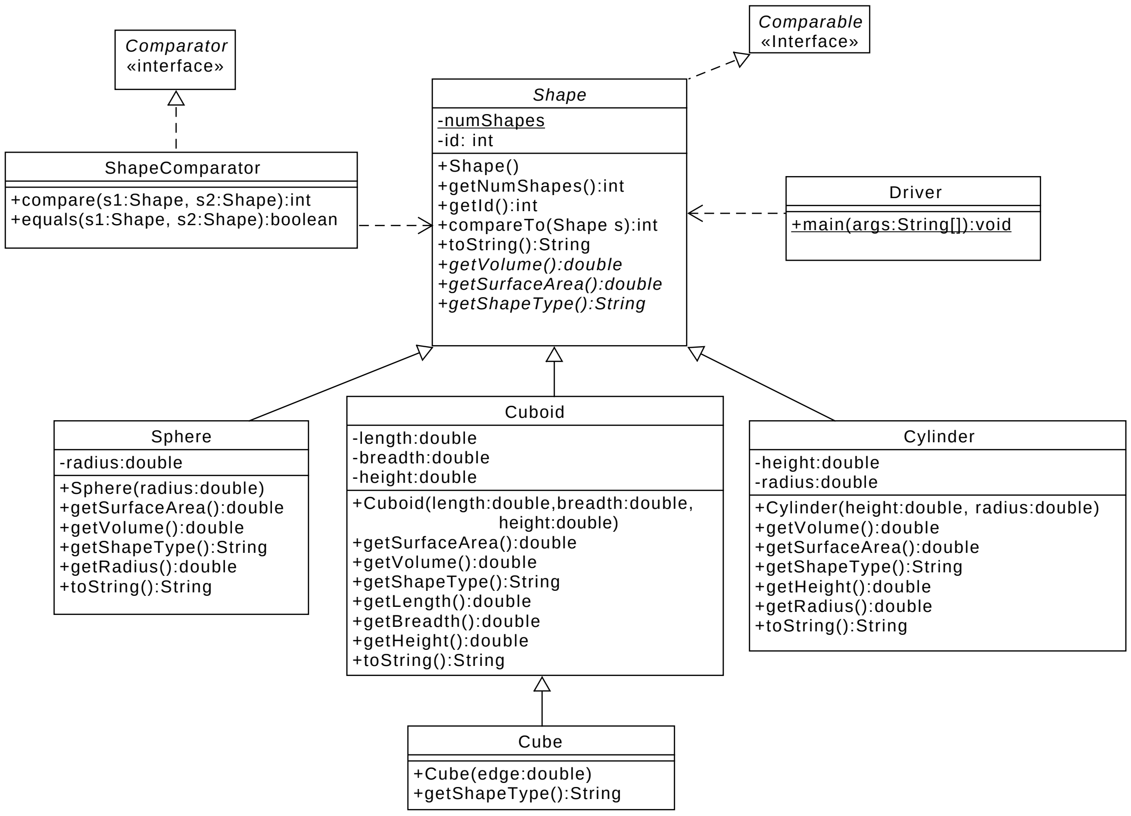
This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

Download the lab6 implementation:
`http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab6/lab6.zip`

Representing Different Shapes

Below is the UML representation of a set of classes that represent various 3D 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. Note that **Comparable** and **Comparator** are defined in the Java library. The concrete child classes must implement all methods from the abstract parent classes. In this lab, **Shape** is an abstract class.

The line from **Shape** to **Comparable** indicates that **Shape** must implement the **Comparable** interface. The **Shape** class compares shape objects based on their surface areas and then their volumes. In contrast, the **ShapeComparator** class must implement the **Comparator** interface. **ShapeComparator** compares shape objects based on their volume only. You must provide both the *compare()* and *equals()* methods.

All instances of a **Shape** are given a unique int *id*. These are to be assigned by the **Shape** constructor. The first instance of a shape that is created is assigned an *id* of 0 (zero); the next is assigned 1. *numShapes* is a static variable that will be incremented each time a new 3D Shape object is created and keeps track of the total number of 3D Shape objects that have been created.

In the **Cube** class, the lengths of the three sides i.e *length*, *breadth* and *height* are equal to *edge* length.

Lab 6: Specific Instructions

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

1. Import *lab6.zip* into your Eclipse workspace.
2. Implement the attributes and methods for each class
    - Complete the implementation of the *Shape* and *Cube* classes. Implement the *Sphere*, *Cuboid*, and *ShapeComparator* classes. Complete implementations of the *Cylinder* and *Driver* classes have been provided.
    - Use the same spelling and visibility 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. Create JUnit test for all the methods implemented in the above classes.
    - You need to convince yourself that everything is working properly.
    - Make sure that you cover all the classes and methods while creating your tests. Keep in mind that we have our own tests that we will use for grading.

Notes

1. Examine Cylinder.toString(). For a Cylinder item with ID = 1, shapeType = "Cylinder", SurfaceArea = 31.42, Volume = 12.57, height = 4.00 and radius = 1.00, toString() returns a String in the following format:

   ```
   Cylinder:	ID = 1	Surface Area = 31.42	Volume = 12.57	height = 4.00	radius = 1.00
   ```

   Note that each of Surface Area, Volume, height and radius are represented using exactly two digits following the decimal point. The whitespace preceding each variable name is implemented with a single *tab* character ("\t"). There is also one space on either side each equal sign.

2. Sphere.toString(): for a Sphere object with ID = 0, shapeType = "Sphere", SurfaceArea = 201.06, Volume = 268.08 and radius = 4.00, toString() must return a String in the following format:

   ```
   Sphere:	ID = 0	Surface Area = 201.06	Volume = 268.08	radius = 4.00
   ```

   The format follows the same pattern as the **Cylinder** class.

3. Cuboid.toString(): for a Cuboid object with ID = 2, shapeType = "Cuboid", a SurfaceArea = 94.00, Volume = 60.00, length = 3.00, breadth = 4.00 and height = 5.00, toString() must return a String in the following format:

   ```
   Cuboid:	ID = 2	Surface Area = 94.00	Volume = 60.00	length = 3.00	breadth = 4.00 ↩
       height = 5.00
   ```

   Note that the left arrow is not part of the String (it only indicates that the full String does not fit on the width of this page).

4. The *Shape* class, through **Comparable** interface, compares the shape objects based on their surface areas (in ascending order); if their surface areas are identical, the comparison is based on their volume (again, in ascending order). An example list of objects after sorting is as follows:

   ```
   Cuboid:	ID = 2	Surface Area = 6.00	Volume = 1.00	length = 1.00	breadth = 1.00 ↩
       height = 1.00
   Cylinder:	ID = 1	Surface Area = 12.57	Volume = 3.14	height = 1.00	radius = ↩
       1.00
   Sphere:	ID = 0	Surface Area = 12.57	Volume = 4.19	radius = 1.00
   Cube:	ID = 3	Surface Area = 54.00	Volume = 27.00	length = 3.00	breadth = 3.00 ↩
       height = 3.00
   Cube:	ID = 4	Surface Area = 96.00	Volume = 64.00	length = 4.00	breadth = 4.00 ↩
       height = 4.00
   ```

5. The *ShapeComparator* class, through the **Comparator** interface, compares the shape objects based on their Volume (in descending order). An example list of **Shapes** after sorting with the **ShapeComparator** is as follows:

```
Cube:   ID = 4  Surface Area = 96.00    Volume = 64.00  length = 4.00   breadth = 4.00 ↩
    height = 4.00
Cube:   ID = 3  Surface Area = 54.00    Volume = 27.00  length = 3.00   breadth = 3.00 ↩
    height = 3.00
Sphere: ID = 0  Surface Area = 12.57    Volume = 4.19   radius = 1.00
Cylinder:   ID = 1  Surface Area = 12.57    Volume = 3.14   height = 1.00   radius = 1.00
Cuboid: ID = 2  Surface Area = 6.00     Volume = 1.00   length = 1.00   breadth = 1.00 ↩
    height = 1.00
```

6. The formulas for the 3D Shape Classes are as follows:

Class	Surface Area	Volume	Shape Type
Sphere	$4\pi r^2$	$\frac{4}{3}\pi r^3$	"Sphere"
Cuboid	$2(l \times b + b \times h + h \times l)$	$l \times b \times h$	"Cuboid"
Cylinder	$2\pi r(r + h)$	$h\pi r^2$	"Cylinder"
Cube	$6l^2$	l^3	"Cube"

Final Steps

1. Generate Javadoc using Eclipse.
   - Select *Project/Generate Javadoc...*
   - Make sure that your **lab6** project is selected, as are all of your java files
   - Select your *doc* directory
   - 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

Before submission, finish testing your program by executing your unit tests. If your program passes all tests and your classes are covered completely by your test classes, then you are ready to attempt a submission. Here are the details:

- All required components (source code and compiled documentation) are due at 7pm on Saturday, September 30. **Submission must be done through the Web-Cat server.**
- Use the same submission process as you used in lab 1. You must submit your implementation to the *Lab 6: Abstract Classes and Interfaces* area on the Web-Cat server.

Reading

- API for the *Collections Framework*
- **Comparable** interface.
- **Comparator** interface.

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.