Lab 5: (More!) Classes and Methods

Let's get started!

Today we'd like you to:

1. **Open Eclipse**. Remember to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf. If you need some help organizing your workspace, we'd be happy to help!
2. **Create a new Java project**. Call it something that will help keep your code organized, i.e., COMP 1510 Lab 5.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 5 project to quickly create a new Java class.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. **For full marks in this and all future labs, your lab instructor *may* require your code must be properly indented, fully commented, and free of Checkstyle complaints**. Remember to activate Checkstyle by right-clicking your project in the Package Explorer pane and selecting Checkstyle > Activate Checkstyle.

Going forward, a few words on modular vs non-modular projects. The baseline assumption is you will be using modular projects for any project that is using JavaFX, otherwise it is your choice. If the project is modular, have a module-info.java file and add the libraries in the Java Build Path>Libraries>Modulepath section. If the project is not modular, add the libraries under the classpath section.

What will you DO in this lab?

In this lab, you will:

1. Design and implement some more classes and their methods
2. Encapsulate your objects by using visibility modifiers and getters/setters
3. Write some more methods that accept parameters and return values
4. Implement constructors that initialize objects in logical and organized ways
5. Create a GUI with JavaFX and use an event handler to respond to a user action.

Table of Contents

Let's get started! **1**

What will you DO in this lab? **1**

1. Sphere **3**

2. Cube **3**

3. Cone **5**

4. Geometry Driver **5**

5. Enhancing our Name Class **6**

6. JavaFX: Voting with Buttons and Text Fields **7**

7. You're done! Show your lab instructor your work. **7**

1. Sphere

Geometry (and math in general) is fun. Software developers don't need to be geometers (that's what we call mathematicians who specialize in geometry) but we should be familiar with the basics. Here is the first of some fun geometry concepts for you to implement:

1. Create a new class called **Sphere** inside package ca.bcit.comp1510.lab5:
    a. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
        i. The name of the class and a (very) short description
        ii. An @author tag followed by your name
        iii. An @version tag followed by the version number.
    b. Do not include a **main method** inside the class definition. Remember the main method is what gets executed first when we start our program. In this case, the Sphere class is not a complete program. It will simply be used to represent a Sphere concept. We will create our main method in a separate Driver class.
2. A Sphere is defined mathematically as *the set of points in 3D space that are all at the same distance r (radius) from a given point*. This suggests that a Sphere should have instance variables that represent the following:
    a. X-coordinate
    b. Y-coordinate
    c. Z-coordinate [1]
    d. Radius.
3. **The Sphere class needs a constructor** which accepts a parameter for each of these four attributes, and assigns the value to the respective instance variable.
4. Create an **accessor** and a **mutator** for each instance variable.
5. Create a method that returns the **surface area** of the Sphere. The formula for the surface area A of a sphere of radius r $= 4\pi r^2$.
6. Create a method that returns the **volume** of the Sphere. The formula for the volume V of a sphere of radius r $= \frac{4}{3}\pi r^3$.
7. Create a **toString()** method which returns a String composed of the concatenation of the information in the Sphere. Customarily the toString() is the last method in the class.

2. Cube

For a cube centered at the origin, with edges parallel to the axes and with an edge length of 1, the Cartesian coordinates of the vertices are (±0.5, ±0.5, ±0.5), and the interior consists of all points (x, y, z) in the range [-0.5, 0.5]. Can you draw this on a piece of paper?

1. Create a new class called **Cube** inside package ca.bcit.comp1510.lab5:

[1] Visit https://en.wikipedia.org/wiki/Cartesian_coordinate_system#Three_dimensions to learn more about Cartesian (3D) coordinates.

   a. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
      i. The name of the class and a (very) short description
      ii. An @author tag followed by your name
      iii. An @version tag followed by the version number.
   b. Do not include a **main method** inside the class definition. We will create our main method in a separate Driver class.
2. A Cube should have instance variables that represent the following:
   a. X-coordinate
   b. Y-coordinate
   c. Z-coordinate
   d. Edge length.
3. **The Cube class needs a constructor** which accepts a parameter for each of these four attributes, and assigns the value to the respective instance variable.
4. Create an **accessor** and a **mutator** for each instance variable.
5. Create a **toString()** method which returns a String composed of the concatenation of the information in the Cube.
6. Create a method that returns the **surface area** of the Cube. The formula for the surface area A of a cube of edge length a $= 6a^2$.
7. Create a method that returns the **volume** of the Cube. The formula for the volume V of a Cube of edge length a $= a^3$.
8. Create a method that returns the **face diagonal** of the Cube. The formula for the face diagonal F of a Cube of edge length a $= \sqrt{2}a$.
9. Create a method that returns the **space diagonal** of the Cube. The formula for the space diagonal S of a Cube of edge length a $= \sqrt{3}a$.

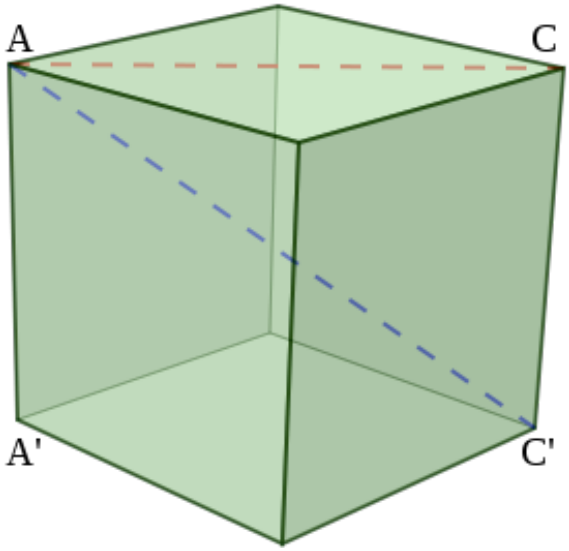

Figure 1 Cube Diagonals. AC' (shown in blue) is a space diagonal while AC (shown in red) is a face diagonal. [2]

[2] By R. A. Nonenmacher [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY-SA 4.0-3.0-2.5-2.0-1.0 (https://creativecommons.org/licenses/by-sa/4.0-3.0-2.5-2.0-1.0)], via Wikimedia Commons

3. Cone

Cones come in a few varieties, and we will consider the right circular cone. A cone with a circular base is a circular cone. A circular cone whose axis is perpendicular to the base is a right circular cone.

1. Create a new class called **Cone** inside package ca.bcit.comp1510.lab5:
    a. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
        i. The name of the class and a (very) short description
        ii. An @author tag followed by your name
        iii. An @version tag followed by the version number.
2. Do not include a **main method** inside the class definition. As before, the class is not a complete program.
3. Let's not worry about a Cone's location in 3D space (yet!). If we ignore the right circular cone's location, we can represent it with two variables: radius and height. Create **instance variables** for its radius and height.
4. A Cone has a public **constructor** which accepts one parameter for each of the instance variables. The body of the constructor should assign each parameter to its respective instance variable.
5. Create an **accessor** and a **mutator** for each instance variable.
6. Create a method that returns the **volume** of the Cone. The formula for the volume V of a cone of radius r and height h $= \frac{1}{3}\pi r^2 h$.
7. Create a method that returns the **slant height** of the Cone. The formula for the slant height SH of a Cone of radius r and height h $= \sqrt{r^2 + h^2}$.
8. Create a method that returns the **surface area** of the Cone. The formula for the surface area A of a Cone of radius r and height h $= \pi r^2 + \pi r(\sqrt{r^2 + h^2})$
9. Create a **toString**() method which returns a String composed of the concatenation of the information in the Cone.

4. Geometry Driver

We've created three simple shape classes. Now let's practice using the Scanner to interact with the user and create some Shapes:

1. Create a new class called **GeometryDriver** inside package ca.bcit.comp1510.lab5:
    a. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
        i. The name of the class and a (very) short description
        ii. An @author tag followed by your name
        iii. An @version tag followed by the version number.
    b. Include a **main method** inside the class definition. Remember the main method is what gets executed first when we start our program.

2. Inside the main method, **instantiate a Scanner object** and use it to interact with the user. Remember to always label your output.
3. Prompt the user to enter the radius and coordinates for a sphere. Use these values to **create a Sphere** object. Print out its surface area and volume.
4. Prompt the user to enter the edge length and centre coordinates for a cube. Use these values to **create a Cube** object. Print out its surface area, volume, and two diagonals.
5. Finally prompt the user to enter the radius and height for a right circular cone. Use these values to **create a Cone** object. Print out its volume, slant height, and surface area.
6. Modify your GeometryDrive to use a **DecimalFormat** object to format the output. Ensure all values are printed to a maximum of three decimal places.
7. **Challenge**: how can you use a **NumberFormat** object to format your output instead?

5. Enhancing our Name Class

We created a simple Name class in lab 4. It stores three parts, and has accessors, mutators, and a toString method. Let's enhance it with some more functionality:

1. Let's copy your Name class from lab 4 to lab 5. Find it in your lab 4 project. Right click it in the Package Manager, and choose copy. Right click the package ca.bcit.comp1510.lab5 and select paste.
2. Did you notice that when you copied the class to the new project, the package declaration in the file was automatically updated? Neat!
3. Add the following methods to your Name class:
    a. A method that accepts no parameters and returns the length of the name, i.e., the sum of the lengths of the three parts of the name.
    b. A method that accepts no parameters and returns a String consisting of the three initials IN UPPERCASE.
    c. A method that accepts an integer n and returns the nth character in the full three part name.
    d. A method that accepts no parameters and returns a String consisting of the last name followed by a comma followed by the first name followed by the middle name. Remember to put spaces between the names.
    e. A method that accepts a String and returns true if the String is equal to the first name (use the equals method in the String class!). This is not quite the proper way to define an equals method, as we will learn later.
    f. A method that accepts a Name object and returns true if the three parts of the name object are the same as the three parts of "this" Name object.
4. Create a class called **NameDriver** which contains a main method. Inside the main method, instantiate a Name object using your name and test your methods to make sure they work. Does it work for all names?

6. JavaFX: Voting with Buttons and Text Fields

It's time for some fun.

Last week in lecture we saw the concepts of controls, events, and event handlers.

You learned how to create a simple PushCounter (refer to Listing 4.7 in the text), and you examined text fields in the FahrenheitConverter (refer to Listing 4.9 in the text).

You read about three ways to specify an event handler for a button or for responding when the user presses return in a text field:

1. method reference, i.e., this::processButtonPress (covered in the lecture)
2. separate class that implements EventHandler<ActionEvent>
3. lambda expression.

Using the PushCounter and FahrenheitConverter code (which is in the COMP 1510 Examples zip file available on the Learning Hub) as reference, modify the PushCounter program so:

1. There are two separate counters, each of which is incremented by its own button. You can label one counter "YES", and the other counter "NO".
2. Add a button for each counter that DECREMENTS the count.
3. Add a text field. When the user types a number in the text field and presses enter, both counters should be set to that value. After setting the counters, reset the text field so that it is blank.

7. You're done! Show your lab instructor your work.