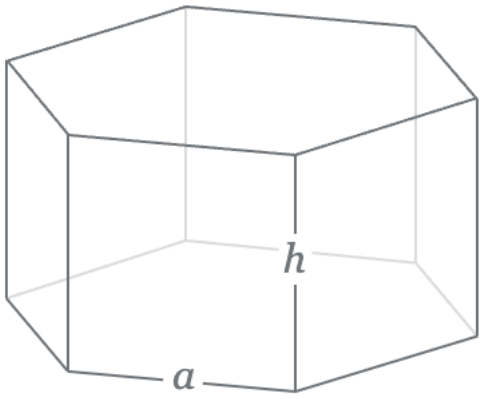
Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. You may submit your files to the skeleton code assignment until the project due date but should try to do this much earlier. The skeleton code assignment is ungraded, but it checks that your classes and methods are named correctly and that methods and parameters are correctly typed. The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. In order to avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code. If you are unable to submit via Web-CAT, you should e-mail your files in a zip file to your TA before the deadline.

Files to submit to Web-CAT (all three files must be submitted together):

- HexagonalPrism.java
- HexagonalPrismList.java
- HexagonalPrismListApp.java

The **Hexagonal Prism** is a prism with hexagonal base. This polyhedron has 8 faces, 18 edges, and 12 vertices. The formulas are provided to assist you in computing return values for the respective methods in the HexagonalPrism class described in this project.
(Sources: "Google" Hexagonal Prism; https://en.wikipedia.org/wiki/Hexagonal_prism)



Formulas for surface area (***A***), lateral surface area ($\boldsymbol{A_L}$), base area ($\boldsymbol{A_B}$), and volume (***V***) are shown below where ***a*** is the *base edge length* and ***h*** is *height*, both of which will be read in.

$$A = 6ah + 3\sqrt{3}\,a^2$$

$$A_L = 6ah$$

$$A_B = 3\sqrt{3}\,\frac{a^2}{2}$$

$$V = \frac{3\sqrt{3}}{2}\,a^2h$$

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines HexagonalPrism objects, the second class defines HexagonalPrismList objects, and the third, HexagonalPrismListApp, reads in a file name entered by the user then reads the list name and HexagonalPrism data from the file, creates HexagonalPrism objects and stores them in an ArrayList of HexagonalPrism objects, creates an HexagonalPrismList object with the list name and ArrayList,

prints the HexagonalPrismList object, and then prints summary information about the HexagonalPrismList object.

- **HexagonalPrism.java (assuming that you successfully created this class in the previous project, just copy the file to your new project folder and go on to HexagonalPrismList.java on page 4. Otherwise, you will need to create HexagonalPrism.java as part of this project.)**

  Requirements: Create a HexagonalPrism class that stores the label, edge, and height. The HexagonalPrism class also includes methods to set and get each of these fields, as well as methods to calculate the surface area, lateral surface area, base area, and volume of a HexagonalPrism object, and a method to provide a String value of a HexagonalPrism object (i.e., a class instance).

  Design: The HexagonalPrism class has fields, a constructor, and methods as outlined below.

  (1) Fields (instance variables): label of type String, edge of type double, and height of type double. Initialize the String to "" and the doubles to zero in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the HexagonalPrism class, and these should be the only instance variables in the class.

  (2) Constructor: Your HexagonalPrism class must contain a public constructor that accepts three parameters (see types of above) representing the label, edge, and height. Instead of assigning the parameters directly to the fields, the respective set method for each field (described below) should be called. For example, instead of the statement label = labelIn; use the statement setLabel(labelIn); Below are examples of how the constructor could be used to create HexagonalPrism objects. Note that although String and numeric literals are used for the actual parameters (or arguments) in these examples, variables of the required type could have been used instead of the literals.

```
HexagonalPrism ex1 = new HexagonalPrism ("Ex 1", 3.0, 5.0);

HexagonalPrism ex2 = new HexagonalPrism (" Ex 2   ", 21.3, 10.4);

HexagonalPrism ex3 = new HexagonalPrism ("Ex 3", 10.0, 204.5);
```

  (3) Methods: Usually a class provides methods to access and modify each of its instance variables (known as get and set methods) along with any other required methods. The methods for HexagonalPrism, which should each be public, are described below. See formulas in Code and Test below.
  - `getLabel`: Accepts no parameters and returns a String representing the label field.
  - `setLabel`: Takes a String parameter and returns a boolean. If the string parameter is not null, then the label field is set to the "trimmed" String and the method returns true. Otherwise, the method returns false, and the label field is not set.

- `getEdge`: Accepts no parameters and returns a double representing the edge field.
- `setEdge`: Accepts a double parameter and returns a boolean as follows. If the parameter is greater than zero, sets the edge field to the double passed in and returns true. Otherwise, the method returns false, and the edge field is not set.
- `getHeight`: Accepts no parameters and returns a double representing the height field.
- `setHeight`: Accepts a double parameter and returns a boolean as follows. If the parameter is greater than zero, sets the height field to the double passed in and returns true. Otherwise, the method returns false, and the height field is not set.
- `surfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.
- `lateralSurfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.
- `baseArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using the using the formula above.
- `toString`: Returns a String containing the information about the HexagonalPrism object formatted as shown below, including decimal formatting (`"#,##0.0##"`) for the double values. Newline escape sequences should be used to achieve the proper layout. In addition to the field values (or corresponding "get" methods), the following methods should be used to compute appropriate values in the toString method: lateralSurfaceArea(), baseArea(), surfaceArea(), and volume(). Each line should have no leading and no trailing spaces (e.g., there should be no spaces before a newline (\n) character). The results of printing the toString value of ex1, ex2, and ex3 respectively are shown below.

```
HexagonalPrism "Ex 1" has 8 faces, 18 edges, and 12 vertices.
   edge = 3.0 units
   height = 5.0 units
   lateral surface area = 90.0 square units
   base area = 23.383 square units
   surface area = 136.765 square units
   volume = 116.913 cubic units

HexagonalPrism "Ex 2" has 8 faces, 18 edges, and 12 vertices.
   edge = 21.3 units
   height = 10.4 units
   lateral surface area = 1,329.12 square units
   base area = 1,178.721 square units
   surface area = 3,686.562 square units
   volume = 12,258.7 cubic units
```

```
HexagonalPrism "Ex 3" has 8 faces, 18 edges, and 12 vertices.
   edge = 10.0 units
   height = 204.5 units
   lateral surface area = 12,270.0 square units
   base area = 259.808 square units
   surface area = 12,789.615 square units
   volume = 53,130.659 cubic units
```

Code and Test: As you implement your HexagonalPrism class, you should compile it and then test it using interactions. For example, as soon you have implemented and successfully compiled the constructor, you should create instances of HexagonalPrism in interactions (e.g., copy/paste the examples above on page 2). Remember that when you have an instance on the workbench, you can unfold it to see its values. You can also open a viewer canvas window and drag the instance from the Workbench tab to the canvas window. After you have implemented and compiled one or more methods, create an HexagonalPrism object in interactions and invoke each of your methods on the object to make sure the methods are working as intended. You may find it useful to create a separate class with a main method that creates an instance of HexagonalPrism then prints it out. This would be similar to the HexagonalPrismApp class you will create below, except that in the HexagonalPrismApp class you will read in the values and then create and print the object.

- **HexagonalPrismList.java**

Requirements: Create an HexagonalPrismList class that stores the name of the list and an ArrayList of HexagonalPrism objects. It also includes methods that return the name of the list, number of HexagonalPrism objects in the HexagonalPrismList, total surface area, total volume, average surface, and average volume for all HexagonalPrism objects in the HexagonalPrismList. The toString method returns a String containing the name of the list followed by each HexagonalPrism in the ArrayList, and a summaryInfo method returns summary information about the list (see below).

Design: The HexagonalPrismList class has two fields, a constructor, and methods as outlined below.

(1) Fields (or instance variables): (1) a String representing the name of the list and (2) an ArrayList of HexagonalPrism objects. These are the only fields (or instance variables) that this class should have, and both should be private.

(2) Constructor: Your HexagonalPrismList class must contain a constructor that accepts a parameter of type String representing the name of the list and a parameter of type ArrayList<HexagonalPrism> representing the list of HexagonalPrism objects. These parameters should be used to assign the fields described above (i.e., the instance variables).

(3) Methods: The methods for HexagonalPrismList are described below.

- `getName`: Returns a String representing the name of the list.

- `numberOfHexagonalPrisms`: Returns an int representing the number of HexagonalPrism objects in the HexagonalPrismList. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `totalSurfaceArea`: Returns a double representing the total surface area for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `totalVolume`: Returns a double representing the total volume for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `averageSurfaceArea`: Returns a double representing the average surface area for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `toString`: Returns a String (does not begin with \n) containing the name of the list followed by each HexagonalPrism in the ArrayList. In the process of creating the return result, this toString() method should include a while loop that calls the toString() method for each HexagonalPrism object in the list (adding a \n before and after each). Be sure to include appropriate newline escape sequences. For an example, see lines 3 through 28 in the output below from HexagonalPrismListApp for the *HexagonalPrism_data_1.txt* input file. [Note that the toString result should **not** include the summary items in lines 30 through 36 of the example. These lines represent the return value of the summaryInfo method below.]
- `summaryInfo`: Returns a String (does not begin with \n) containing the name of the list (which can change depending on the value read from the file) followed by various summary items: number of HexagonalPrism objects, total surface area, total volume, average surface area, and average volume. Use `"#,##0.0##"` as the pattern to format the double values. For an example, see lines 30 through 36 in the output below from HexagonalPrismListApp for the *HexagonalPrism_data_1.txt* input file. The second example below shows the output from HexagonalPrismListApp for the *HexagonalPrism_data_0.txt* input file which contains a list name but no HexagonalPrism data.

Code and Test: Remember to import java.util.ArrayList. Each of the four methods above that finds a total or average requires that you use a loop (i.e., a while loop) to retrieve each object in the ArrayList. As you implement your HexagonalPrismList class, you can compile it and then test it using interactions. However, it may be easier to create a class with a simple main method that creates an HexagonalPrismList object and calls its methods.

- **HexagonalPrismListApp.java**

Requirements: Create an HexagonalPrismListApp class with a main method that (1) reads in the name of the data file entered by the user and (2) reads list name and HexagonalPrism data from the file, (3) creates HexagonalPrism objects, storing them in a local ArrayList of HexagonalPrism objects; and finally, (4) creates an HexagonalPrismList object with the name of the list and the ArrayList of HexagonalPrism objects, and then prints the HexagonalPrismList object followed

summary information about the HexagonalPrismList object. **All input and output for this project must be done in the main method**.

- **Design**: The main method should prompt the user to enter a file name, and then it should read in the data file. The first record (or line) in the file contains the name of the list. This is followed by the data for the HexagonalPrism objects. Within a while loop, each set of HexagonalPrism data (i.e., label, edge, and height) is read in, and then a HexagonalPrism object should be created and added to the local ArrayList of HexagonalPrism objects. After the file has been read in and the ArrayList has been populated, the main method should create a HexagonalPrismList object with the name of the list and the ArrayList of HexagonalPrism objects as parameters in the constructor. It should then print the HexagonalPrismList object, and then print the summary information about the HexagonalPrismList (i.e., print the value returned by the summaryInfo method for the HexagonalPrismList). The output from two runs of the main method in HexagonalPrismListApp is shown below. The first is produced after reading in the *HexagonalPrism_data_1.txt* file, and the second is produced after reading in the *HexagonalPrism_data_0.txt* file. Your program output should be formatted exactly as shown on the next page.

Example 1

Line #	Program output
1	`Enter file name: HexagonalPrismList_data_1.txt`
2	
3	`----- HexagonalPrism Test List -----`
4	
5	`HexagonalPrism "Ex 1" has 8 faces, 18 edges, and 12 vertices.`
6	`   edge = 3.0 units`
7	`   height = 5.0 units`
8	`   lateral surface area = 90.0 square units`
9	`   base area = 23.383 square units`
10	`   surface area = 136.765 square units`
11	`   volume = 116.913 cubic units`
12	
13	`HexagonalPrism "Ex 2" has 8 faces, 18 edges, and 12 vertices.`
14	`   edge = 21.3 units`
15	`   height = 10.4 units`
16	`   lateral surface area = 1,329.12 square units`
17	`   base area = 1,178.721 square units`
18	`   surface area = 3,686.562 square units`
19	`   volume = 12,258.7 cubic units`
20	
21	`HexagonalPrism "Ex 3" has 8 faces, 18 edges, and 12 vertices.`
22	`   edge = 10.0 units`
23	`   height = 204.5 units`
24	`   lateral surface area = 12,270.0 square units`
25	`   base area = 259.808 square units`
26	`   surface area = 12,789.615 square units`
27	`   volume = 53,130.659 cubic units`
28	
29	
30	`----- Summary for HexagonalPrism Test List -----`
31	`Number of HexagonalPrisms: 3`
32	`Total Surface Area: 16,612.943 square units`
33	`Total Volume: 65,506.272 cubic units`
34	`Average Surface Area: 5,537.648 square units`
35	`Average Volume: 21,835.424 cubic units`
36	

Example 2

Line #	Program output
1	`Enter file name: HexagonalPrismList_data_0.txt`
2	
3	`----- HexagonalPrism Empty Test List -----`
4	
5	
6	`----- Summary for HexagonalPrism Empty Test List -----`
7	`Number of HexagonalPrisms: 0`
8	`Total Surface Area: 0.0 square units`
9	`Total Volume: 0.0 cubic units`
10	`Average Surface Area: 0.0 square units`
11	`Average Volume: 0.0 cubic units`
12	

Code: Remember to import java.util.ArrayList, java.util.Scanner, and java.io.File, and java.io.FileNotFoundException prior to the class declaration. Your main method declaration should indicate that main `throws FileNotFoundException`. After your program reads in the file name from the keyboard, it should read in the data file using a Scanner object that was created on a file using the file name entered by the user.

```
... = new Scanner(new File(fileName));
```

You can assume that the first line in the data file is the name of the list, and then each set of three lines contains the data from which an HexagonalPrism object can be created. After the name of the list has been read and assigned to a local variable, a while loop should be used to read in the HexagonalPrism data. The boolean expression for the while loop should be `(_____.hasNext())` where the blank is the name of the Scanner you created on the file. Each iteration through the loop reads three lines. As each of the lines is read from the file, the respective local variables for the HexagonalPrism data items (label, edge, height) should be assigned, after which the HexagonalPrism object should be created and added to a local ArrayList of HexagonalPrism objects. The next iteration of the loop should then read the next set of three lines then create the next HexagonalPrism object and add it to the local ArrayList of HexagonalPrism objects, and so on. After the file has been processed (i.e., when the loop terminates after the hasNext method returns false), name of the list and the ArrayList of HexagonalPrism objects should be used to create an HexagonalPrismList object. Then the list should be printed by printing a leading \n and the HexagonalPrismList object. Finally, the summary information is printed by printing a leading \n and the value returned by the summaryInfo method invoked on the HexagonalPrismList object.

Test: You should test your program minimally (1) by reading in the *HexagonalPrism_data_1.txt* input file, which should produce the first output above, and (2) by reading in the *HexagonalPrism_data_0.txt* input file, which should produce the second output above. Although your program may not use all the methods in the HexagonalPrismList and HexagonalPrism classes, you should ensure that all your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.

General Notes

1. All input from the keyboard and all output to the screen should done in the main method. Only one Scanner object on System.in should be created and this should be done in the main method. All printing (i.e., using the System.out.print and/or System.out.println methods) should be in the main method. Hence, none of your methods in the HexagonalPrism and HexagonalPrismList classes should do any input/output (I/O).

2. Be sure to download the test data files (*HexagonalPrism_data_1.txt* and *HexagonalPrism_data_0.txt*) and store them in same folder as your source files. It may be useful examine the contents of the data files. Find the data files in the jGRASP Browse tab and then open each data file in jGRASP to see the items that your program will be reading from the file. Be sure to close the data files without changing them.