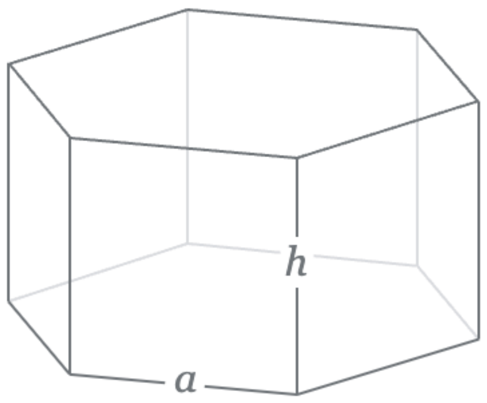
Deliverables

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 (***A_L***), base area (***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, HexagonalPrismListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates a HexagonalPrismList object), (2) print report, (3) print summary, (4) add a HexagonalPrism object to the HexagonalPrismList object, (5) delete a HexagonalPrism object from the HexagonalPrismList object, (6) find a HexagonalPrism object in the HexagonalPrismList object, (7) Edit a HexagonalPrism in the HexagonalPrismList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (HexagonalPrism.java, HexagonalPrismList.java, HexagonalPrism_data_1.txt, and HexagonalPrism_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

- **HexagonalPrism.java (<u>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</u>.)**

  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 <u>these should be the only instance variables in the class</u>.

  (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.
  - o `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** – extended from the previous project by **adding the last six methods below. (<u>Assuming that you successfully created this class in the previous project, just copy HexagonalPrismList.java to your new Project folder and then add the indicated methods. Otherwise, you will need to create all of HexagonalPrismList.java as part of this project</u>.)**

  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.

The following six methods are new in Project 6:

- `getList`: Returns the ArrayList of HexagonalPrism objects (the second field above).
- `readFile`: Takes a String parameter representing the file name, reads in the file, storing the list name and creating an ArrayList of HexagonalPrism objects, uses the list name and the ArrayList to create a HexagonalPrismList object, and then returns the HexagonalPrismList object. See note #1 under Important Considerations for the

HexagonalPrismListMenuApp class (last page) to see how this method should be called. The `readFile` method header should include the **throws** `FileNotFoundException` clause.

- `addHexagonalPrism`: Returns nothing but takes three parameters (label, edge, and height), creates a new HexagonalPrism object, and adds it to the HexagonalPrismList object.
- `findHexagonalPrism`: Takes a label of a HexagonalPrism as the String parameter and returns the corresponding HexagonalPrism object if found in the HexagonalPrismList object; otherwise returns null. This method should ignore case when attempting to match the label.
- `deleteHexagonalPrism`: Takes a String as a parameter that represents the label of the HexagonalPrism and returns the HexagonalPrism if it is found in the HexagonalPrismList object and deleted; otherwise returns null. This method should use the String equalsIgnoreCase method when attempting to match a label in the HexagonalPrism object to delete.
- `editHexagonalPrism`: Takes three parameters (label, edge, and height), uses the label to find the corresponding the HexagonalPrism object. If found, sets the edge and height to the values passed in as parameters, and returns true. If not found, simply returns false. Note that this method should not set the label.

Code and Test: Remember to import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.FileNotFoundException. These classes will be needed in the readFile method which will require a throws clause for FileNotFoundException. Some of the methods above require that you use a loop to go through the objects in the ArrayList. You may want to implement the class below in parallel with this one to facilitate testing. That is, after implementing one to the methods above, you can implement the corresponding "case" in the switch for the menu described below in the HexagonalPrismListMenuApp class.

- **HexagonalPrismListMenuApp.java** (replaces HexagonalPrismListApp class from the previous project)

  Requirements: Create a HexagonalPrismListMenuApp class with a main method that presents the user with a menu with eight options, each of which is implemented to do the following: (1) read the input file and create a HexagonalPrismList object, (2) print the HexagonalPrismList object, (3) print the summary for the HexagonalPrismList object, (4) add a HexagonalPrism - object to the HexagonalPrismList object, (5) delete a HexagonalPrism object from the HexagonalPrismList object, (6) find a HexagonalPrism object in the HexagonalPrismList object, (7) Edit a HexagonalPrism object in the HexagonalPrismList object, and (8) quit the program.

  Design: The main method should print a list of options with the action code and a short description followed by a line with just the action codes prompting the user to select an action. After the user enters an action code, the action is performed, including output if any. Then the line with just the action codes prompting the user to select an action is printed again to accept the next code. The first action a user would normally perform is 'R' to read in the file and create a

HexagonalPrismList object. However, the other action codes should work even if an input file has not been processed. The user may continue to perform actions until 'Q' is entered to quit (or end) the program. Note that your program should accept both uppercase and lowercase action codes (see items 3 and 4 in the **Code and Test** section below).
Below is output produced by running HexagonalPrismListMenuApp and printing the action codes with short descriptions followed by the prompt with the abbreviated action codes waiting for the user to select from the menu.

Example 1

```
HexagonalPrism List System Menu
R - Read File and Create HexagonalPrism List
P - Print HexagonalPrism List
S - Print Summary
A - Add HexagonalPrism
D - Delete HexagonalPrism
F - Find HexagonalPrism
E - Edit HexagonalPrism
Q - Quit
Enter Code [R, P, S, A, D, F, E, or Q]:
```

Below shows the output after the user entered 'r' and then (when prompted) the file name. Notice the output from this action was "File read in and HexagonalPrism List created". This is followed by the prompt with the action codes waiting for the user to make the next selection. You should use the *HexagonalPrism_data_1.txt* file from Project 5 to test your program.

Example 2

```
Enter Code [R, P, S, A, D, F, E, or Q]: r
   File name: HexagonalPrismList_data_1.txt
   File read in and HexagonalPrism List created

Enter Code [R, P, S, A, D, F, E, or Q]:
```

The result of the user selecting 'p' to Print HexagonalPrism List is shown below and next page.

Example 3

```
Enter Code [R, P, S, A, D, F, E, or Q]: p

----- HexagonalPrism Test List -----

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

Enter Code [R, P, S, A, D, F, E, or Q]:
```

The result of the user selecting ‘s’ to print the summary for the list is shown below.

Example 4

```
Enter Code [R, P, S, A, D, F, E, or Q]: s

----- Summary for HexagonalPrism Test List -----
Number of HexagonalPrisms: 3
Total Surface Area: 16,612.943 square units
Total Volume: 65,506.272 cubic units
Average Surface Area: 5,537.648 square units
Average Volume: 21,835.424 cubic units

Enter Code [R, P, S, A, D, F, E, or Q]:
```

The result of the user selecting ‘a’ to add a HexagonalPrism object is shown below. Note that after ‘a’ was entered, the user was prompted for label, edge, and height. Then after the HexagonalPrism object is added to the HexagonalPrism List, the message “*** HexagonalPrism added ***” was printed. This is followed by the prompt for the next action. After you do an “add”, you should do a “print” or a “find” to confirm that the “add” was successful.

Example 5

```
Enter Code [R, P, S, A, D, F, E, or Q]: a
   Label: New HP
   Edge: 22.2
   Height: 77.7
   *** HexagonalPrism added ***

Enter Code [R, P, S, A, D, F, E, or Q]:
```

Here is an example of the successful "delete" for a HexagonalPrism object, followed by an attempt that was not successful (i.e., the HexagonalPrism object was not found). Do "p" to confirm the "delete".

Example 6

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
2	`   Label: Ex 2`
3	`   "Ex 2" deleted`
4	
5	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
6	`   Label: Ex 11`
7	`   "Ex 11" not found`
8	
9	`Enter Code [R, P, S, A, D, F, E, or Q]:`

In the example below, there is a successful "find" for a HexagonalPrism object, followed by an attempt that was <u>not</u> successful (i.e., the HexagonalPrism object with label Fake was not found), and finally an invalid code Y was entered.

Example 7

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
2	`   Label: ex 3`
3	`HexagonalPrism "Ex 3" has 8 faces, 18 edges, and 12 vertices.`
4	`   edge = 10.0 units`
5	`   height = 204.5 units`
6	`   lateral surface area = 12,270.0 square units`
7	`   base area = 259.808 square units`
8	`   surface area = 12,789.615 square units`
9	`   volume = 53,130.659 cubic units`
10	
11	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
12	`   Label: Not A Real Label`
13	`   "Not A Real Label" not found`
14	
15	`Enter Code [R, P, S, A, D, F, E, or Q]:`
16	
17	

Below is an example of the successful "edit" for a HexagonalPrism object, followed by an attempt that was <u>not</u> successful. To verify the edit, you should do a "find" for "ex 3" or you could do a "print" to see the whole list. <u>Note the actual label, Ex 3, is shown on line 5 rather than ex 3 which was entered by the user. Edit does not modify the label</u>.

Example 8

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
2	`   Label: ex 3`
3	`   Edge: 5.555`
4	`   Height: 7.777`
5	`   "Ex 3" successfully edited`

```

Enter Code [R, P, S, A, D, F, E, or Q]: e
   Label: Also Not A Real Label
   Edge: 1.0
   Height: 3.0
   "Also Not A Real Label" not found

Enter Code [R, P, S, A, D, F, E, or Q]:
```

Entering a ‘q’ should quit the application with no message.

Example 9

```
Enter Code [R, P, S, A, D, F, E, or Q]: q

```

Finally, if a code other than the ones in the “Enter Code” list is entered, the code is considered invalid and an appropriate message is printed, then the user is asked to enter a code again as shown below.

Example 10

```
Enter Code [R, P, S, A, D, F, E, or Q]: T
   *** invalid code ***

Enter Code [R, P, S, A, D, F, E, or Q]:
```

Code and Test: This class should import java.util.Scanner, java.util.ArrayList, and java.io.FileNotFoundException. Carefully consider the following **important considerations** as you develop this class.

1. At the beginning of your main method, you should declare and create an ArrayList of HexagonalPrism objects and then declare and create a HexagonalPrismList object using the list name and the ArrayList as the parameters in the constructor. This will be a HexagonalPrismList object that contains no HexagonalPrism objects. For example:

```
String _______ = "*** no list name assigned ***";
ArrayList<HexagonalPrism> _________ = new  ArrayList<HexagonalPrism>();
HexagonalPrismList _________ = new HexagonalPrismList(_________,_________);
```

   The ‘R’ option in the menu should invoke the readFile method on your HexagonalPrismList object. This will return a new HexagonalPrismList object based on the data read from the file, and this new HexagonalPrismList object should replace (be assigned to) your original HexagonalPrismList object variable in main. Since the readFile method throws FileNotFoundException, your main method needs to do this as well.

2. **Very Important**: **You should declare only one Scanner on System.in for your entire program, and this should be done in the main method.** That is, all input from the keyboard (System.in) must be done in your *main* method. Declaring more than one Scanner

on System.in in your program will likely result in a very low score from Web-CAT.

3. After you read in the code as a String, you should invoke the toUpperCase() method on the code and then convert the first character of the String to a char as shown in the two statements below.
   ```
   code = code.toUpperCase();
   char codeChar = code.charAt(0);
   ```

4. For the menu, your switch statement expression should evaluate to a char and each case should be a char.

After printing the menu of actions with descriptions, you should have a *do-while* loop that prints the prompt with just the action codes followed by a switch statement that performs the indicated action. The *do-while* loop ends when the user enters 'q' to quit. You should strongly consider using a *for-each* loop as appropriate in the new methods that require you to search the list. You should be able to test your program by exercising each of the action codes. After you implement the "Print HexagonalPrism List" option, you should be able to print the HexagonalPrismList object after operations such as 'A' and 'D' to see if they worked. You may also want to run in debug mode with a breakpoint set at the switch statement so that you can step-into your methods if something is not working. In conjunction with running the debugger, you should also create a canvas drag the items of interest (e.g., the Scanner on the file, your HexagonalPrismList object, etc.) onto the canvas and save it. As you play or step through your program, you'll be able to see the state of these objects change when the 'R', 'A', and 'D' options are selected.

Although your program may not use all the methods in your HexagonalPrism and HexagonalPrismList classes, you should ensure that all your methods work according to the specification. You can run your program in the canvas and then after the file has been read in, you can call methods on the HexagonalPrismList object in interactions, or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.