Due: Skeleton Code (ungraded - checks class, method names, etc.)
Completed Code (100%) – Monday, March 7, 2016 by 11:59 PM

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until the project due date but should try to do this by Friday (there is no late penalty since this is ungraded for this 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 to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

F**iles to submit to Web-CAT:**

- HexagonalPrism.java
- HexagonalPrismList.java
- HexagonalPrismListMenuApp.java

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, hexagonal_prism_0.dat, and hexagonal_prism_1.dat) **to it, rather than work in the same folder as Project 5 files.]**

- **HexagonalPrism.java (assuming that you successfully created this class in Project 4 or 5, just copy the file to your new Project 6 folder and go on to HexagonalPrismList.java on page 3. Otherwise, you will need to create HexagonalPrism.java as part of this project.)**

  Requirements: Create a HexagonalPrism class that stores the label, side, and height. The HexagonalPrism class also includes methods to set and get each of these fields, as well as methods to calculate the base perimeter, base area, surface 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, side of type double, and height of type double. 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 constructor that accepts three parameters (see types of above) representing the label, side, and height. The value for label should be trimmed of leading and trailing spaces prior to setting the field (hint: use trim method from the String class). 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 example1 = new HexagonalPrism("Short Example", 4.0, 6.0);

HexagonalPrism example2 = new HexagonalPrism(" Wide Example ", 22.1, 10.6);

HexagonalPrism example3 = new HexagonalPrism("Tall Example", 10, 200);
```

(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 are described 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 null, then the method returns false and the label is not set. Otherwise, the "trimmed" String is set to the label field and the method returns true.
- `getSide`: Accepts no parameters and returns a double representing the side field.
- `setSide`: Accepts a double parameter, sets side field, and returns nothing.
- `getHeight`: Accepts no parameters and returns a double representing the height field.
- `setHeight`: Accepts a double parameter, sets height field, and returns nothing.
- `basePerimeter`: Accepts no parameters and returns the double value for the base perimeter of the top (or bottom) hexagon calculated using side.
- `baseArea`: Accepts no parameters and returns the double value for the base area calculated using side.
- `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using side and height.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using side, and height.
- `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: basePerimeter(), 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 example1, example2, and example3 respectively are shown below.

```
"Short Example" is a hexagonal prism with side = 4.0 units and height = 6.0 units,
which has base perimeter = 24.0 units, base area = 41.569 square units,
surface area = 227.138 square units, and volume = 249.415 cubic units.

"Wide Example" is a hexagonal prism with side = 22.1 units and height = 10.6 units,
which has base perimeter = 132.6 units, base area = 1,268.926 square units,
surface area = 3,943.413 square units, and volume = 13,450.62 cubic units.

"Tall Example" is a hexagonal prism with side = 10.0 units and height = 200.0 units,
which has base perimeter = 60.0 units, base area = 259.808 square units,
surface area = 12,519.615 square units, and volume = 51,961.524 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 (see the examples above). 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 a 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.

- **HexagonalPrismList.java –** extended from Project 5 by **adding the last six methods below. (Assuming that you successfully created this class in Project 5, just copy HexagonalPrismList.java to your new Project 6 folder and then add the indicated methods. Otherwise, you will need to create all of HexagonalPrismList.java as part of this project.)**

  Requirements: Create a 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, total base perimeter, total base area, average surface area, 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.

  (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 areas 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 volumes for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `totalBasePerimeter`: Returns a double representing the total for the base perimeters for all HexagonalPrism objects in the list. If there are zero HexagonalPrism objects in the list, zero should be returned.
- `totalBaseArea`: Returns a double representing the total for the base areas 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 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. Be sure to include appropriate newline escape sequences. For an example, see lines 3 through 21 in the output in Project 5 from HexagonalPrismListApp for the *hexagonal_prism_1.dat* input file. [Note that the toString result should **not** include the summary items in lines 22 through 30 of the example. These lines represent the return value of the summaryInfo method below.]
- `summaryInfo`: Returns a String containing the name of the list (which can change depending of the value read from the file) followed by various summary items: number of hexagonal prisms, total surface area, total volume, total base perimeter, total base area, average surface area, average volume. For an example, see lines 22 through 30 in the output in Project 5 from HexagonalPrismListApp for the *hexagonal_prism_1.dat* input file. The second example below shows the output from HexagonalPrismListApp for the *hexagonal_prism_0.dat* input file which contains a list name but no hexagonal prism 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 an 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.

- `addHexagonalPrism`: Returns nothing but takes three parameters (label, side, and height) creates a new HexagonalPrism object and adds it to the HexagonalPrismList object.
- `deleteHexagonalPrism`: Takes the 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.
- `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.
- `editHexagonalPrism`: Takes three parameters (label, side, and height), uses the label to find the corresponding the HexagonalPrism object. If found, sets the side and height to the values passed in as parameters, and returns true. If not found, simply returns false.

Code and Test: Remember to import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.IOException. These classes will be needed in the readFile method which will require a throws clause for IOException. 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 menu described below in the HexagonalPrismListMenuApp class.

- **HexagonalPrismListMenuApp.java** (replaces the previous HexagonalPrismListApp class in Project 5)

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 (7) 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.

Below is output produced after printing the action codes with short descriptions followed by the prompt with the action codes waiting for the user to make a selection.

Line #	Program output
1	Hexagonal Prism List System Menu
2	R - Read File and Create Hexagonal Prism List
3	P - Print Hexagonal Prism List
4	S - Print Summary
5	A - Add Hexagonal Prism
6	D - Delete Hexagonal Prism
7	F - Find Hexagonal Prism
8	E - Edit Hexagonal Prism
9	Q - Quit
10	Enter Code [R, P, S, A, D, F, E, or Q]:

Below shows the screen after the user entered 'r' and then (when prompted) the file name. Notice the output from this action was "File read in and Hexagonal Prism 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 *hexagonal_prism_1.dat* file from Project 5 to test your program.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: r
2	File name: hexagonal_prism_1.dat
3	File read in and Hexagonal Prism List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

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

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: p
2	
3	Hexagonal Prism List 1
4	
5	"Short Example" is a hexagonal prism with side = 4.0 units and height = 6.0 units,
6	which has base perimeter = 24.0 units, base area = 41.569 square units,
7	surface area = 227.138 square units, and volume = 249.415 cubic units.
8	
9	"Wide Example" is a hexagonal prism with side = 22.1 units and height = 10.6 units,
10	which has base perimeter = 132.6 units, base area = 1,268.926 square units,
11	surface area = 3,943.413 square units, and volume = 13,450.62 cubic units.
12	
13	"Tall Example" is a hexagonal prism with side = 10.0 units and height = 200.0 units,
14	which has base perimeter = 60.0 units, base area = 259.808 square units,
15	surface area = 12,519.615 square units, and volume = 51,961.524 cubic units.
16	
17	"Really Large Example" is a hexagonal prism with side = 300.0 units and height = 400.0
18	units,
19	which has base perimeter = 1,800.0 units, base area = 233,826.859 square units,
20	surface area = 1,187,653.718 square units, and volume = 93,530,743.609 cubic units.
21	

22	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.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: s
2	
3	----- Hexagonal Prism List 1 Summary -----
4	Number of Hexagonal Prisms: 4
5	Total Surface Area: 1,204,343.885
6	Total Volume: 93,596,405.168
7	Total Base Perimeter: 2,016.6
8	Total Base Area: 235,397.162
9	Average Surface Area: 301,085.971
10	Average Volume: 23,399,101.292
11	
12	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, side, and height. Then after the HexagonalPrism object is added to the HexagonalPrismList, the message "*** Hexagonal Prism 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.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: a
2	Label: my new one
3	Side: 11
4	Height: 22
5	*** Hexagonal Prism added ***
6	
7	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 "d".

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

Here is an example of the successful "find" for a HexagonalPrism object, followed by an attempt that was not successful (i.e., the HexagonalPrism object was not found), and then an invalid code.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: f
2	Label: tall example
3	"Tall Example" is a hexagonal prism with side = 10.0 units and height = 200.0 units,
4	which has base perimeter = 60.0 units, base area = 259.808 square units,
5	surface area = 12,519.615 square units, and volume = 51,961.524 cubic units.
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: f
8	Label: really tall example
9	"really tall example" not found
10	
11	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful "edit" for a HexagonalPrism object, followed by an attempt that was not successful (i.e., the HexagonalPrism object was not found). In order to verify the edit, you should do a "find" for "wide example" or you could do a "print" to print the whole list.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: e
2	Label: wide example
3	Side: 22
4	Label: 3
5	"wide example" successfully edited
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: e
8	Label: really
9	Side: 22
10	Label: 3
11	"really" not found
12	
13	Enter Code [R, P, S, A, D, F, E, or Q]:

Finally, entering a 'q' should quit the application with no message.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: q
2	
3	----jGRASP: operation complete.

Code and Test:
Important considerations: This class should import java.util.Scanner, java.util.ArrayList, and java.io.IOException. Carefully consider the following information 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 IOException, your main method need 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. For the menu, your switch statement expression should evaluate to a char and each case should be a char; alternatively, your switch statement expression should evaluate to a String with a length of 1 and each case should be a String with a length of 1.

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 Hexagonal Prism 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 of the methods in your HexagonalPrism and HexagonalPrismList classes, you should ensure that all of 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.