Due: Skeleton Code (ungraded - checks class, method names, etc.)
Completed Code – Monday, February 29, 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
- HexagonalPrismListApp.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, HexagonalPrismListApp, reads in a file name from the user then reads the list name and hexagonal prism data from the file, creates HexagonalPrism objects and stores them in an ArrayList, creates a 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 Project 4, just copy the file to your new Project 5 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**

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 below 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 below 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.

Code and Test: Remember to import java.util.ArrayList. Each of the methods above 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. Alternatively, you can create a class with a simple main method that creates a HexagonalPrismList object and calls its methods.

- **HexagonalPrismListApp.java**

Requirements: Create a HexagonalPrismListApp class with a main method that reads in the name of the input file from the user and then reads list name and HexagonalPrism data from the file, creates HexagonalPrism objects, stores them in a local ArrayList, 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 should 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 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. After each set of HexagonalPrism data is read in, a HexagonalPrism object should be created and stored in the local ArrayList. After the file has been read in and the ArrayList created, 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 followed by 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 produced after reading in the *hexagonal_prism_1.dat* file and the second after reading in the *hexagonal_prism_0.dat* file. Your program output should be formatted exactly as follows):

Line #	Program output
	----jGRASP exec: java HexagonalPrismListApp
1	Enter file name: hexagonal_prism_1.dat
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	----- Hexagonal Prism List 1 Summary -----
23	Number of Hexagonal Prisms: 4
24	Total Surface Area: 1,204,343.885
25	Total Volume: 93,596,405.168
26	Total Base Perimeter: 2,016.6
27	Total Base Area: 235,397.162
28	Average Surface Area: 301,085.971
29	Average Volume: 23,399,101.292
30	
	----jGRASP: operation complete.

Line #	Program output
	----jGRASP exec: java HexagonalPrismListApp
1	Enter file name: hexagonal_prism_0.dat
2	
3	Hexagonal Prism List 0
4	
5	----- Hexagonal Prism List 0 Summary -----
6	Number of Hexagonal Prisms: 0
7	Total Surface Area: 0.0
8	Total Volume: 0.0
9	Total Base Perimeter: 0.0
10	Total Base Area: 0.0
11	Average Surface Area: 0.0
12	Average Volume: 0.0
13	
	----jGRASP: operation complete.

Code: Remember to import java.util.ArrayList, java.util.Scanner, and java.io.File, and java.io.IOException prior to the class declaration. Your main method declaration should indicate that main `throws IOException`. After your program reads in the file name from the keyboard, it should read in the 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 file is the name of the list, and then each set of three lines contains the data from which a 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 hexagonal prism 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 hexagonal prism data items (label, side, and height) should be assigned, after which the HexagonalPrism object should be created and added to a local ArrayList. 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 a local ArrayList, 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 should be used to create a HexagonalPrismList object. The HexagonalPrismList object should then be printed. Finally, the summary information is printed by printing the value returned by the summaryInfo method invoked on the HexagonalPrismList object.

Test: You should test your program minimally (1) by reading in the *hexagonal_prism_1.dat* input file, which should produce the first output above, and (2) by reading in the *hexagonal_prism_0.dat* input file, which should produce the second output above. Although your program may not use all of the methods in HexagonalPrismList and HexagonalPrism, you should ensure that all of 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.