

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.

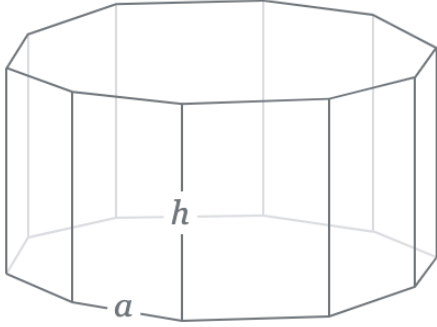
Files to submit to Web-CAT (all three files must be submitted together):

- DecagonalPrism.java
- DecagonalPrismList.java
- DecagonalPrismListApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines DecagonalPrism objects, the second class defines DecagonalPrismList objects, and the third, DecagonalPrismListApp, reads in a file name entered by the user then reads the list name and DecagonalPrism data from the file, creates DecagonalPrism objects and stores them in an ArrayList of DecagonalPrism objects, creates an DecagonalPrismList object with the list name and ArrayList, prints the DecagonalPrismList object, and then prints summary information about the DecagonalPrismList object.

A **decagonal prism** is a 3-D geometric shape formed by ten square side faces and two regular decagon caps. The decagonal prism has 12 faces, 30 edges, and 20 vertices. A decagonal prism can be defined by its base edge (a) and height (h) as depicted below. The formulas are provided to assist you in computing return values for the respective methods in the DecagonalPrism class described in this project.

	<p>Nomenclature</p> <p>Base edge: a Height: h</p> <p>Surface area: A Base area: A_B Lateral surface area: A_L Volume: V</p>	<p>Formulas</p> $A = 10 a h + 5 a^2 \sqrt{5 + 2\sqrt{5}}$ $A_B = \frac{5}{2} a^2 \sqrt{5 + 2\sqrt{5}}$ $A_L = 10 a h$ $V = \frac{5}{2} a^2 \sqrt{5 + 2\sqrt{5}} h$
---	--	---

- **DecagonalPrism.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 DecagonalPrismList.java on page 4. Otherwise, you will need to create DecagonalPrism.java as part of this project.)

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

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

- (1) **Fields** (instance variables): label of type String, base edge 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 DecagonalPrism class, and these should be the only instance variables in the class.
- (2) **Constructor:** Your DecagonalPrism class must contain a 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 DecagonalPrism 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.

```
DecagonalPrism ex1 = new DecagonalPrism("Small", 2.0, 5.0);
```

```
DecagonalPrism ex2 = new DecagonalPrism(" Medium ", 10.4, 32.6);
```

```
DecagonalPrism ex3 = new DecagonalPrism("Large", 50, 100);
```

- (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 DecagonalPrism are described below. See the formulas above and “Code and Test” below for details. When implementing the formulas, be sure to use the methods `Math.pow` and `Math.sqrt` as appropriate.
 - `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 trimmed String is set to the label field 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. If the edge is non-negative, sets edge field 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. If the height is non-negative, sets height field 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 edge and height.
- `baseArea`: Accepts no parameters and returns the double value for the base area calculated using the edge.
- `lateralSurfaceArea`: Accepts no parameters and returns the double value for the slant height calculated using the edge and height.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using edge and height.
- `toString`: Returns a String containing the information about the DecagonalPrism object formatted as shown below, including decimal formatting ("#,##0.0###") for the double values. Newline and tab 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: `surfaceArea()`, `baseArea()`, `lateralSurfaceArea()`, and `volume()`. Each line should have no leading and no trailing spaces (e.g., there should be no spaces before a newline (\n) other than tab (\t) as appropriate). The toString value for ex1, ex2, and ex3 respectively are shown below (the blank lines are not part of the toString values; that is, the return value should not begin with a newline (\n) character and should not end with a newline (\n) character).

```
A decagonal prism "Small" with edge = 2.0 units and height = 5.0 units, has:  
  surface area = 161.554 square units  
  base area = 30.777 square units  
  lateral surface area = 100.0 square units  
  volume = 153.884 cubic units
```

```
A decagonal prism "Medium" with edge = 10.4 units and height = 32.6 units, has:  
  surface area = 5,054.811 square units  
  base area = 832.206 square units  
  lateral surface area = 3,390.4 square units  
  volume = 27,129.903 cubic units
```

```
A decagonal prism "Large" with edge = 50.0 units and height = 100.0 units, has:  
  surface area = 88,471.044 square units  
  base area = 19,235.522 square units  
  lateral surface area = 50,000.0 square units  
  volume = 1,923,552.211 cubic units
```

Code and Test: As you implement your DecagonalPrism 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 DecagonalPrism 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 DecagonalPrism 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 DecagonalPrism then prints it out.

- **DecagonalPrismList.java**

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

Design: The DecagonalPrismList 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 DecagonalPrism objects. These instance variables should be private so that they are not directly accessible from outside of the DecagonalPrismList class, and these should be the only instance variables in the class.
- (2) **Constructor:** Your DecagonalPrismList class must contain a constructor that accepts a parameter of type String representing the name of the list and a parameter of type ArrayList<DecagonalPrism> representing the list of DecagonalPrism objects. These parameters should be used to assign the fields described above (i.e., the instance variables).
- (3) **Methods:** The methods for DecagonalPrismList are described below.
 - o `getName`: Returns a String representing the name of the list.
 - o `numberOfDecagonalPrisms`: Returns an int representing the number of DecagonalPrism objects in the DecagonalPrismList. If there are zero DecagonalPrism objects in the list, zero should be returned.
 - o `totalSurfaceArea`: Returns a double representing the total surface areas for all DecagonalPrism objects in the list. If there are zero DecagonalPrism objects in the list, zero should be returned.
 - o `totalBaseArea`: Returns a double representing the total for the base areas for all DecagonalPrism objects in the list. If there are zero DecagonalPrism objects in the list, zero should be returned.
 - o `totalLateralSurfaceArea`: Returns a double representing the total for the lateral surface area for all DecagonalPrism objects in the list. If there are zero DecagonalPrism objects in the list, zero should be returned.
 - o `totalVolume`: Returns a double representing the total volumes for all DecagonalPrism objects in the list. If there are zero DecagonalPrism objects in the list, zero should be returned.

- `averageSurfaceArea`: Returns a double representing the average surface area for all `DecagonalPrism` objects in the list. If there are zero `DecagonalPrism` objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all `DecagonalPrism` objects in the list. If there are zero `DecagonalPrism` objects in the list, zero should be returned.
- `toString`: Returns a String (which does not begin with `\n`) containing the name of the list followed by each `DecagonalPrism` 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 `DecagonalPrism` object in the list. Be sure to include appropriate newline escape sequences. For an example, see lines 3 through 24 in the output below from `DecagonalPrismListApp` for the `DecagonalPrism_data_1.txt` input file. [Note that the `toString` result should **not** include the summary items in lines 26 through 35. These lines represent the return value of the `summaryInfo` method below.]
- `summaryInfo`: Returns a String (which does not begin with `\n`) containing the name of the list (which can change depending of the value read from the file) followed by various summary items: number of `DecagonalPrisms`, total surface area, total base area, total lateral surface area, total volume, average surface area, average volume (with decimal formatting pattern `"#,##0.0###"` for the double values). For an example, see lines 26 through 35 in the output below from `DecagonalPrismListApp` for the `DecagonalPrism_data_1.txt` input file. The second example below shows the output from `DecagonalPrismListApp` for the `DecagonalPrism_data_0.txt` input file which contains a list name but no `DecagonalPrism` 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 `DecagonalPrismList` 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 `DecagonalPrismList` object and calls its methods.

- **DecagonalPrismListApp.java**

Requirements: Create a `DecagonalPrismListApp` class with a main method that (1) reads in the name of the input file entered by the user and then (2) reads list name and `DecagonalPrism` data from the file, (3) creates `DecagonalPrism` objects, stores them in a local `ArrayList` of `DecagonalPrism` objects, and finally (4) creates an `DecagonalPrismList` object with the name of the list and the `ArrayList` of `DecagonalPrism` objects, and then prints the `DecagonalPrismList` object followed summary information about the `DecagonalPrismList` 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 `DecagonalPrism` objects. After each set of `DecagonalPrism` data is read in, a `DecagonalPrism` 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 `DecagonalPrismList` object with the name of the list and the `ArrayList` of `DecagonalPrism` objects as parameters in the constructor. It should then print the `DecagonalPrismList` object followed by summary

information about the DecagonalPrismList (i.e., print the value returned by the summaryInfo method for the DecagonalPrismList). The output from two runs of the main method in DecagonalPrismListApp is shown below: the first produced after reading in the *DecagonalPrism_data_1.txt* file and the second after reading in the *DecagonalPrism_data_0.txt* file. Your program output should be formatted exactly as follows).

Line #	Program output
1	----jGRASP exec: java DecagonalPrismListApp
2	Enter file name: DecagonalPrism_data_1.txt
3	Decagonal Prism List 1
4	
5	A decagonal prism "Small" with edge = 2.0 units and height = 5.0 units, has:
6	surface area = 161.554 square units
7	base area = 30.777 square units
8	lateral surface area = 100.0 square units
9	volume = 153.884 cubic units
10	
11	A decagonal prism "Medium" with edge = 10.4 units and height = 32.6 units, has:
12	surface area = 5,054.811 square units
13	base area = 832.206 square units
14	lateral surface area = 3,390.4 square units
15	volume = 27,129.903 cubic units
16	
17	A decagonal prism "Large" with edge = 50.0 units and height = 100.0 units, has:
18	surface area = 88,471.044 square units
19	base area = 19,235.522 square units
20	lateral surface area = 50,000.0 square units
21	volume = 1,923,552.211 cubic units
22	
23	A decagonal prism "Giant" with edge = 300.0 units and height = 1,000.0 units, has:
24	surface area = 4,384,957.592 square units
25	base area = 692,478.796 square units
26	lateral surface area = 3,000,000.0 square units
27	volume = 692,478,795.864 cubic units
28	
29	
30	----- Summary for Decagonal Prism List 1 -----
31	Number of Decagonal Prisms: 4
32	Total Surface Area: 4,478,645.001
33	Total Base Area: 712,577.3
34	Total Lateral Surface Area: 3,053,490.4
35	Total Volume: 694,429,631.863
36	Average Surface Area: 1,119,661.25
37	Average Volume: 173,607,407.966
38	
	----jGRASP: operation complete.

Line #	Program output
--------	----------------

	----jGRASP exec: java DecagonalPrismListApp
1	Enter file name: DecagonalPrism_data_0.txt
2	
3	Empty List of Decagonal Prisms
4	
5	
6	----- Summary for Empty List of Decagonal Prisms -----
7	Number of Decagonal Prisms: 0
8	Total Surface Area: 0.0
9	Total Base Area: 0.0
10	Total Lateral Surface Area: 0.0
11	Total Volume: 0.0
12	Average Surface Area: 0.0
13	Average Volume: 0.0
14	
	----jGRASP: operation complete.

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 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 `DecagonalPrism` 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 `DecagonalPrism` 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 `DecagonalPrism` data items (label, edge, and height) should be assigned, after which the `DecagonalPrism` 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 `DecagonalPrism` 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 `DecagonalPrismList` object. The `DecagonalPrismList` object should then be printed. Finally, the summary information is printed by printing the value returned by the `summaryInfo` method invoked on the `DecagonalPrismList` object.

Test: You should test your program minimally (1) by reading in the *DecagonalPrism_1.txt* input file, which should produce the first output above, and (2) by reading in the *DecagonalPrism_0.txt* input file, which should produce the second output above. Although your program may not use all the methods in `DecagonalPrismList` and `DecagonalPrism`, you should ensure that all of your methods work according to the specification. You can either use 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 be 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 System.out.println methods) should be in the main method. Hence, none of your methods in the DecagonalPrism class and DecagonalPrismList class should do any input/output (I/O).
2. Be sure to download the test data files (*DecagonalPrism_data_1.txt* and *DecagonalPrism_data_0.txt*) and store them in same folder as your source files. It may be useful to 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.