

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.

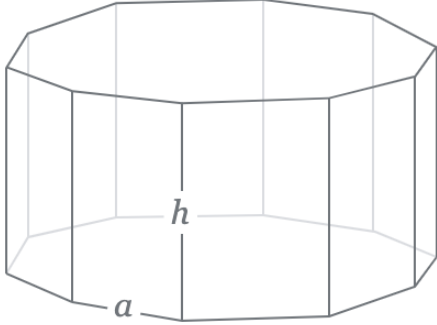
Files to submit to Web-CAT (all three files must be submitted together):

- DecagonalPrism.java
- DecagonalPrismList.java
- DecagonalPrismListMenuApp.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, DecagonalPrismListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates a DecagonalPrismList object), (2) print report, (3) print summary, (4) add a DecagonalPrism object to the DecagonalPrismList object, (5) delete a DecagonalPrism object from the DecagonalPrismList object, (6) find a DecagonalPrism object in the DecagonalPrismList object, (7) Edit a DecagonalPrism in the DecagonalPrismList object, and (8) quit the program. **[You should create a new “Project 6” folder and copy your Project 5 files (DecagonalPrism.java, DecagonalPrismList.java, DecagonalPrism_data_1.txt, and DecagonalPrism_data_0.txt) to it, rather than work in the same folder as Project 5 files.]**

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.

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

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

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

- `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 28 in the output from DecagonalPrismListApp for the *DecagonalPrism_data_1.txt* input file. [Note that the toString result should not include the summary items in lines 30 through 38. 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 30 through 38 in the output from DecagonalPrismListApp for the *DecagonalPrism_data_1.txt* input file. The second example shows the output from DecagonalPrismListApp for the *DecagonalPrism_data_0.txt* input file which contains a list name but no DecagonalPrism data.
- **The following six methods are new in Project 6:**
 - `getList`: Returns the ArrayList of DecagonalPrism 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 DecagonalPrism objects, uses the list name and the ArrayList to create a DecagonalPrismList object, and then returns the DecagonalPrismList object. See note #1 under Important Considerations for the DecagonalPrismListMenuApp class (last page) to see how this method should be called. The `readFile` method header should include the `throws FileNotFoundException` clause.
 - `addDecagonalPrism`: Returns nothing but takes three parameters (label, edge, and height), creates a new DecagonalPrism object, and adds it to the DecagonalPrismList object.
 - `findDecagonalPrism`: Takes a label of a DecagonalPrism as the String parameter and returns the corresponding DecagonalPrism object if found in the DecagonalPrismList object; otherwise returns null. This method should ignore case when attempting to match the label.
 - `deleteDecagonalPrism`: Takes a String as a parameter that represents the label of the DecagonalPrism and returns the DecagonalPrism if it is found in the DecagonalPrismList object and deleted; otherwise returns null. This method should use

the String equalsIgnoreCase method when attempting to match a label in the DecagonalPrism object to delete.

- editDecagonalPrism: Takes three parameters (label, edge, and height), uses the label to find the corresponding the DecagonalPrism 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 menu described below in the DecagonalPrismListMenuApp class.

- **DecagonalPrismListMenuApp.java** (replaces DecagonalPrismListApp class from the previous project)

Requirements: Create a DecagonalPrismListMenuApp 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 DecagonalPrismList object, (2) print the DecagonalPrismList object, (3) print the summary for the DecagonalPrismList object, (4) add a DecagonalPrism - object to the DecagonalPrismList object, (5) delete a DecagonalPrism object from the DecagonalPrismList object, (6) find a DecagonalPrism object in the DecagonalPrismList object, (7) Edit a DecagonalPrism object in the DecagonalPrismList 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 DecagonalPrismList 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 by running DecagonalPrismListMenuApp and printing the action codes with short descriptions followed by the prompt with the abbreviated action codes waiting for the user to make a selection.

Line #	Program output
1	----jGRASP exec: java DecagonalPrismListMenuApp
2	DecagonalPrism List System Menu
3	R - Read File and Create DecagonalPrism List
4	P - Print DecagonalPrism List
5	S - Print Summary
6	A - Add DecagonalPrism
7	D - Delete DecagonalPrism
8	F - Find DecagonalPrism
9	E - Edit DecagonalPrism
10	Q - Quit
11	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 DecagonalPrism 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 *DecagonalPrism_data_1.txt* 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: DecagonalPrism_data_1.txt
3	File read in and DecagonalPrism List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'p' to Print DecagonalPrism 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	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 lateral surface area = 3,000,000.0 square units volume = 692,478,795.864 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.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: s
2	
3	----- Summary for Decagonal Prism List 1 -----
4	Number of Decagonal Prisms: 4
5	Total Surface Area: 4,478,645.001
6	Total Base Area: 712,577.3
7	Total Lateral Surface Area: 3,053,490.4
8	Total Volume: 694,429,631.863
9	Average Surface Area: 1,119,661.25
10	Average Volume: 173,607,407.966
11	
12	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'a' to add a DecagonalPrism object is shown below. Note that after 'a' was entered, the user was prompted for label, edge, and height. Then after the DecagonalPrism object is added to the DecagonalPrism List, the message "*** DecagonalPrism 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: newwww
3	Edge: 33
4	Height: 44
5	*** DecagonalPrism added ***
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: p

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

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: d
2	Label: large
3	"Large" deleted
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]: d
6	Label: fake one

7	"fake one" 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 DecagonalPrism object, followed by an attempt that was not successful (i.e., the DecagonalPrism object with label gigantic was not found), and finally an example of entering an invalid code Z.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: f
2	Label: medium
3	A decagonal prism "Medium" with edge = 10.4 units and height = 32.6 units, has:
4	surface area = 5,054.811 square units
5	base area = 832.206 square units
6	lateral surface area = 3,390.4 square units
7	volume = 27,129.903 cubic units
8	Enter Code [R, P, S, A, D, F, E, or Q]: f
9	Label: gigantic
10	"gigantic" not found
11	
12	Enter Code [R, P, S, A, D, F, E, or Q]: Z
13	*** invalid code ***
14	
15	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful “edit” for a DecagonalPrism object, followed by an attempt that was not successful (i.e., the DecagonalPrism object was not found). To verify the edit, you should do a “find” for “giant” or you could do a “print” to see the whole list.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: e
2	Label: giant
3	Edge: 150
4	Height: 500
5	"giant" successfully edited
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: e
8	Label: fake two
9	Edge: 12
10	Height: 23
11	"fake two" 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	

-----jGRASP: operation complete.

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 `DecagonalPrism` objects and then declare and create a `DecagonalPrismList` object using the list name and the `ArrayList` as the parameters in the constructor. This will be a `DecagonalPrismList` object that contains no `DecagonalPrism` objects. For example:

```
String _____ = "*** no list name assigned ***";
ArrayList<DecagonalPrism> _____ = new ArrayList<DecagonalPrism>();
DecagonalPrismList _____ = new DecagonalPrismList(_____, _____);
```

The 'R' option in the menu should invoke the `readFile` method on your `DecagonalPrismList` object. This will return a new `DecagonalPrismList` object based on the data read from the file, and this new `DecagonalPrismList` object should replace (be assigned to) your original `DecagonalPrismList` 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. 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 DecagonalPrism List" option, you should be able to print the `DecagonalPrismList` 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 `DecagonalPrismList` 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 `DecagonalPrism` and `DecagonalPrismList` 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 DecagonalPrismList 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.