Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. You may submit your files to the underline 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.
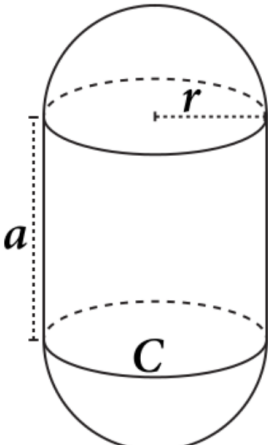
Files to submit to Web-CAT (all three files must be submitted together):
- Spherocylinder.java
- SpherocylinderList.java
- SpherocylinderListMenuApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines Spherocylinder objects, the second class defines SpherocylinderList objects, and the third, SpherocylinderListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates a SpherocylinderList object), (2) print report, (3) print summary, (4) add a Spherocylinder object to the SpherocylinderList object, (5) delete a Spherocylinder object from the SpherocylinderList object, (6) find a Spherocylinder object in the SpherocylinderList object, (7) Edit a Spherocylinder in the SpherocylinderList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (Spherocylinder.java, SpherocylinderList.java, Spherocylinder_data_1.txt, and Spherocylinder_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

A spherocylinder (or capsule) is a 3-dimensional object made up of two hemispheres connected by a cylinder as shown below. The formulas are provided to assist you in computing return values for the respective Spherocylinder methods described in this project. Source for figures and formulas: https://en.wikipedia.org/wiki/Capsule_(geometry)

The variables are abbreviated as follows:
- *r* is radius
- *a* is cylinder height
- *C* is Circumference
- *SA* is Surface Area
- *V* is Volume

$$C = 2\pi r$$

$$SA = 2\pi r(2r + a)$$

$$V = \pi r^2 \left(\frac{4}{3}r + a \right)$$

Spherocylinder.java (<u>assuming that you successfully created this class in Project 4, just copy the file to your new Project 6 folder and go on to SpherocylinderList.java on page 4. Otherwise, you will need to create Spherocylinder.java as part of this project.</u>)

Requirements: Create a Spherocylinder class that stores the label, radius, and cylinder height where both radius and cylinder height are non-negative. The Spherocylinder class also includes methods to set and get each of these fields, as well as methods to calculate the circumference, surface area, and volume of a Spherocylinder object, and a method to provide a String value that describes a Spherocylinder object.

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

(1) **Fields** (instance variables): label of type `String`, radius of type `double`, and cylinder height of type `double`. Initialize the `String` to `""` and the `double` variables to 0 in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the Spherocylinder class, and <u>these should be the only instance variables (fields) in the class</u>.

(2) **Constructor**: Your Spherocylinder class must contain a public constructor that accepts three parameters (see types of above) representing the label, radius, and cylinder height. Instead of assigning the parameters directly to the fields, the respective set method for each field (described below) should be called since they are checking the validity of the parameter. For example, instead of using the statement `label = labelIn;` use the statement `setLabel(labelIn);` Below are examples of how the constructor could be used to create Spherocylinder 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.

```
Spherocylinder example1 = new Spherocylinder("Small Example", 0.5, 0.25);

Spherocylinder example2 = new Spherocylinder(" Medium Example ", 10.8, 10.1);

Spherocylinder example3 = new Spherocylinder("Large Example", 98.32, 99.0);
```

(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 Spherocylinder, which should each be public, are described below. See the formulas in the figure above and the Code and Test section below for information on constructing these methods.

   o `getLabel`: Accepts no parameters and returns a `String` representing the label field.

   o `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 is not set.

   o `getRadius`: Accepts no parameters and returns a `double` representing the radius field.

o setRadius: Takes a double parameter and returns a boolean. If the double parameter is non-negative, then the parameter is set to the radius field and the method returns true. Otherwise, the method returns false and the radius field is not set.

o getCylinderHeight: Accepts no parameters and returns a double representing the cylinder height field.

o setCylinderHeight: Accepts a double parameter and returns a boolean as follows. If the double parameter is non-negative, then the parameter is set to the cylinder height field and the method returns true. Otherwise, the method returns false and the cylinder height field is not set.

o circumference: Accepts no parameters and returns the double value for the circumference of the Spherocylinder.

o surfaceArea: Accepts no parameters and returns the double value for the total surface area of the Spherocylinder.

o volume: Accepts no parameters and returns the double value for the volume of the Spherocylinder. [*Be sure to avoid integer division in your expression*.]

o toString: Returns a String containing the information about the Spherocylinder 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 within the String but it should not begin or end with a newline. In addition to the field values (or corresponding "get" methods), the following methods should be used to compute appropriate values in the toString method: circumference(), surfaceArea(), and volume(). Each line should have no trailing spaces (e.g., there should be no spaces before a newline (\n) character). The toString value for example1, example2, and example3 respectively are shown below (the blank lines are not part of the toString values).

```
Spherocylinder "Small Example" with radius 0.5 and cylinder height 0.25 has:
   circumference = 3.142 units
   surface area = 3.927 square units
   volume = 0.72 cubic units

Spherocylinder "Medium Example" with radius 10.8 and cylinder height 10.1 has:
   circumference = 67.858 units
   surface area = 2,151.111 square units
   volume = 8,977.666 cubic units

Spherocylinder "Large Example" with radius 98.32 and cylinder height 99.0 has:
   circumference = 617.763 units
   surface area = 182,635.388 square units
   volume = 6,987,754.655 cubic units
```

Code and Test: As you implement your Spherocylinder 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 Spherocylinder in interactions (e.g., copy/paste 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 Spherocylinder object in interactions and invoke each of your methods on the object to make sure the methods are working as intended.

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

SpherocylinderList.java

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

Design: The SpherocylinderList 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 Spherocylinder objects. These are the only fields (or instance variables) that this class should have.

(2) **Constructor**: Your SpherocylinderList class must contain a constructor that accepts a parameter of type String representing the name of the list and a parameter of type ArrayList< Spherocylinder> representing the list of Spherocylinder objects. These parameters should be used to assign the fields described above (i.e., the instance variables).

(3) **Methods**: The methods for SpherocylinderList are described below.
   - `getName`: Returns a String representing the name of the list.
   - `numberOfSpherocylinders`: Returns an int representing the number of Spherocylinder objects in the SpherocylinderList. If there are zero Spherocylinder objects in the list, zero should be returned.
   - `totalSurfaceArea`: Returns a double representing the total surface areas for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
   - `totalVolume`: Returns a double representing the total volumes for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
   - `averageSurfaceArea`: Returns a double representing the average surface area for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
   - `averageVolume`: Returns a double representing the average volume for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
   - `toString`: Returns a String (does <u>not</u> begin with \n) containing the name of the list followed by each Spherocylinder 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 Spherocylinder object in the list (adding a \n <u>before and after</u> each). Be sure to include appropriate newline escape sequences. For an example, see <u>lines 2 through 14</u> in the output below from SpherocylinderListApp for the *Spherocylinder_data_1.txt* input file. [Note that <u>the toString result should **not** include the summary items in lines 21 through 26 of the example from previous project. These lines represent the return value of the summaryInfo method below which will be called from</u> `main`.]

o   `summaryInfo`: Returns a String (<u>does not begin or end with \n</u>) containing the name of the list (which can change depending of the value read from the file) followed by various summary items: number of Spherocylinders, 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 <u>lines 21 through 26</u> in the output for SpherocylinderListApp for the *Spherocylinder_data_1.txt* input file in the <u>previous project</u>. The second example below shows the output from SpherocylinderListApp for the *Spherocylinder_data_0.txt* input file which contains a list name but no Spherocylinder data.

<u>The following six methods are new in Project 6</u>:
o   `getList`: Returns the ArrayList of Spherocylinder objects (the second field above).

o   `readFile`: Takes a String parameter representing the file name, reads in the file, storing the list name and creating an ArrayList of Spherocylinder objects, uses the list name and the ArrayList to create a SpherocylinderList object, and then returns the SpherocylinderList object. See note #2 under <u>Important Considerations</u> for the SpherocylinderListMenuApp class to see how this method should be called.

o   `addSpherocylinder`: Returns nothing but takes three parameters (label, radius, and cylinder height), creates a new Spherocylinder object, and adds it to the SpherocylinderList object.

o   `findSpherocylinder`: Takes a label of a Spherocylinder as the String parameter and returns the corresponding Spherocylinder object if found in the SpherocylinderList object; otherwise returns null. <u>Letter case should be ignored when attempting to match the label (e.g., "Small Example" and "sMaLL EXAMPLE" should be a match)</u>.

o   `deleteSpherocylinder`: Takes a String as a parameter that represents the label of the Spherocylinder. Returns the Spherocylinder object if it is found in the SpherocylinderList and deleted; otherwise returns `null`. <u>Letter case should be ignored when attempting to match the label; consider calling/using</u> `findSpherocylinder` <u>in this method</u>.

o   `editSpherocylinder`: Takes three parameters (label, radius, and cylinder height), uses the label to find the corresponding the Spherocylinder object. If found, sets the radius and cylinder height to the values passed in as parameters, and returns true. If not found, returns false.

Code and Test: You must import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.FileNotFoundException as these classes will be needed in the readFile method, which will require a throws clause for FileNotFoundException. You may want to consider implementing the SpherocylinderListMenuApp class below in parallel with the new methods in SpherocylinderList. For example, after implementing the `readFile` method, you can implement the corresponding "case" in the switch for menu described below in the SpherocylinderListMenuApp class that calls

the method. This will allow you quickly test it by entering the 'R' and 'P' options to read in the file and print the list.

- **SpherocylinderListMenuApp.java** (replaces SpherocylinderListApp class from Project 5)

Requirements: Create a SpherocylinderListMenuApp 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 SpherocylinderList object, (2) print the SpherocylinderList object, (3) print the summary for the SpherocylinderList object, (4) add a Spherocylinder object to the SpherocylinderList object, (5) delete a Spherocylinder object from the SpherocylinderList object, (6) find a Spherocylinder object in the SpherocylinderList object, (7) Edit a Spherocylinder object in the SpherocylinderList 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 SpherocylinderList 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	Spherocylinder List System Menu
2	R – Read File and Create Spherocylinder List
3	P – Print Spherocylinder List
4	S – Print Summary
5	A – Add Spherocylinder
6	D – Delete Spherocylinder
7	F – Find Spherocylinder
8	E – Edit Spherocylinder
9	Q – Quit
10	Enter Code [R, P, S, A, D, F, E, or Q]:

Note that after an option code is entered, each sub-prompt has a leading tab (\t) (e.g., lines 2 and 3 below). The example below shows the screen after the user entered 'r' and when prompted, the file name. Notice the output from this action was "File read in and Spherocylinder List created". This is followed by the prompt with the action codes waiting for the user to make the next selection. Use *Spherocylinder_data_1.txt* 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: spherocylinder_data_1.txt
3	File read in and Spherocylinder List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'p' to Print Spherocylinder 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	`Spherocylinder Test List`
4	
5	`Spherocylinder "Small Example" with radius 0.5 and cylinder height 0.25 has:`
6	`    circumference = 3.142 units`
7	`    surface area = 3.927 square units`
8	`    volume = 0.72 cubic units`
9	
10	`Spherocylinder "Medium Example" with radius 10.8 and cylinder height 10.1 has:`
11	`    circumference = 67.858 units`
12	`    surface area = 2,151.111 square units`
13	`    volume = 8,977.666 cubic units`
14	
15	`Spherocylinder "Large Example" with radius 98.32 and cylinder height 99.0 has:`
16	`    circumference = 617.763 units`
17	`    surface area = 182,635.388 square units`
18	`    volume = 6,987,754.655 cubic units`
19	
20	`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 Spherocylinder Test List -----`
4	`Number of Spherocylinders: 3`
5	`Total Surface Area: 184,790.426`
6	`Total Volume: 6,996,733.041`
7	`Average Surface Area: 61,596.809`
8	`Average Volume: 2,332,244.347`
9	
10	`Enter Code [R, P, S, A, D, F, E, or Q]:`

The result of the user selecting 'a' to add a Spherocylinder object is shown below. Note that after 'a' was entered, the user was prompted for label, radius, and cylinder height. Then after the Spherocylinder object is added to the Spherocylinder List, the message "*** Spherocylinder added ***" was printed. Note that lines 2 - 5 below have leading tabs (\t). 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: Test of Add`
3	`    Radius: 1.5`
4	`    Cylinder Height: 3.0`
5	`    *** Spherocylinder added ***`
6	
7	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "delete" for a Spherocylinder object, followed by an attempt that was not successful (i.e., the Spherocylinder 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: Medium Example
3	"Medium Example" deleted
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]: d
6	Label: not a real object
7	"not a real object" 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 Spherocylinder object, followed by an attempt that was not successful (i.e., the Spherocylinder object was not found).

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: f
2	Label: small EXAMPLE
3	Spherocylinder "Small Example" with radius 0.5 and cylinder height 0.25 has:
4	circumference = 3.142 units
5	surface area = 3.927 square units
6	volume = 0.72 cubic units
7	
8	Enter Code [R, P, S, A, D, F, E, or Q]: F
9	Label: tiny example
10	"tiny example" not found
11	
12	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful "edit" for a Spherocylinder object, followed by an attempt that was not successful (i.e., the Spherocylinder 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: small EXAMPLE
3	Radius: 0.1
4	Cylinder Height: 0.2
5	"small EXAMPLE" successfully edited
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: e
8	Label: not a real object
9	Radius: 2.0
10	Cylinder Height: 3.5
11	"not a real object" not found
12	
13	Enter Code [R, P, S, A, D, F, E, or Q]:

Finally, below is an example of entering an invalid code, followed by an example of entering a 'q' to quit the application with no message.

Line #	Program output

1
2
3
4
	`Enter Code [R, P, S, A, D, F, E, or Q]: c`
	`   *** invalid code ***`
	
	`Enter Code [R, P, S, A, D, F, E, or Q]: q`

Code and Test: Important considerations

1. This class should import java.util.Scanner, java.util.ArrayList, and java.io.FileNotFoundException. Carefully consider the following information as you develop this class.

2. At the beginning of your main method, you should declare and create an ArrayList of Spherocylinder objects and then declare and create a SpherocylinderList object using the list name and the ArrayList as the parameters in the constructor. This will be a SpherocylinderList object that contains no Spherocylinder objects. For example:
    ```
    String _____ = "*** no list name assigned ***";
    ArrayList<Spherocylinder> _____ = new ArrayList<Spherocylinder>();
    SpherocylinderList _____ = new SpherocylinderList(_____,_____);
    ```

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

3. **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.

4. 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.

5. 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 Spherocylinder List" option, you should be able to print the SpherocylinderList 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 SpherocylinderList 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.

    a.   For option P, when you print the SpherocylinderList object (e.g., sList) be sure to append a leading "\n" in println:

```
System.out.println("\n" + sList);
```

    b.   For option S, when you print the summary for SpherocylinderList object (e.g., sList) be sure to append a leading and trailing "\n" in the .println:

```
System.out.println("\n" + sList.summaryInfo() + "\n");
```

Although your program may not use all of the methods in your Spherocylinder and SpherocylinderList 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 SpherocylinderList 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.