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.

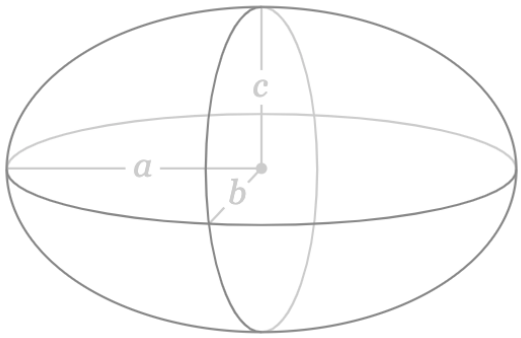
Files to submit to Web-CAT (all three files must be submitted together):

- Ellipsoid.java
- EllipsoidList.java
- EllipsoidListMenuApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines Ellipsoid objects, the second class defines EllipsoidList objects, and the third, EllipsoidListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates an EllipsoidList object), (2) print report, (3) print summary, (4) add an Ellipsoid object to the EllipsoidList object, (5) delete an Ellipsoid object from the EllipsoidList object, (6) find an Ellipsoid object in the EllipsoidList object, (7) Edit an Ellipsoid in the EllipsoidList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (Ellipsoid.java, EllipsoidList.java, Ellipsoid_data_1.txt, and Ellipsoid_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

An **Ellipsoid** is a 3-D object whose plane sections are ellipses defined by three axes (*a*, *b*, *c*) as depicted below. The formulas are provided to assist you in computing return values for the respective methods in the Ellipsoid class described in this project.



Formulas for volume (V) and surface area (S) are shown below.

$$V = \frac{4\pi abc}{3}$$

$$S \approx 4\pi\left(\frac{(a\,b)^{1.6}+(a\,c)^{1.6}+(b\,c)^{1.6}}{3}\right)^{1/1.6}$$

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

  Requirements: Create an Ellipsoid class that stores the label and three axes a, b, and c. The values of the axes must be greater than zero. The Ellipsoid class also includes methods to set and get each of these fields, as well as methods to calculate the volume and surface area of the Ellipsoid object, and a method to provide a String value of an Ellipsoid object (i.e., a class instance).

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

  (1) **Fields** (instance variables): label of type String, and axes a, b, and c of type double. Initialize the String variable 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 Ellipsoid class, and these should be the only instance variables (i.e., fields) in the class.

  (2) **Constructor**: Your Ellipsoid class must contain a public constructor that accepts four parameters (see types of above) representing the label, a, b, and c. 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 Ellipsoid 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.

```
Ellipsoid ex1 = new Ellipsoid ("Ex 1", 1, 2, 3);

Ellipsoid ex2 = new Ellipsoid (" Ex 2   ", 2.3, 5.5, 7.4);

Ellipsoid ex3 = new Ellipsoid ("Ex 3", 123.4, 234.5, 345.6);
```

  (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 Ellipsoid, which should each be public, are described below. See formulas in Code and Test 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 not null, then the label field is set to the "trimmed" String and the method returns true. Otherwise, the method returns false and the label field is not set.
  - `getA`: Accepts no parameters and returns a double representing field *a*.
  - `setA`: Accepts a double parameter and returns a boolean as follows. If the double is greater than zero, sets field *a* to the double passed in and returns true. Otherwise, the method returns false and does not set the field.

- `getB`: Accepts no parameters and returns a double representing field *b*.
- `setB`: Accepts a double parameter and returns a boolean as follows. If the double is greater than zero, sets field *b* to the double passed in and returns true. Otherwise, the method returns false and does not set the field.
- `getC`: Accepts no parameters and returns a double representing field *c*.
- `setC`: Accepts a double parameter and returns a boolean as follows. If the double is greater than zero, sets field *c* to the double passed in and returns true. Otherwise, the method returns false and does not set the field.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using formula above and the values of axes fields *a*, *b*, *c*.
- `surfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using formula above and the values of axes fields *a*, *b*, *c*.
- `toString`: Returns a String containing the information about the Ellipsoid 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: `volume()` and `surfaceArea()`. Each line should have no trailing spaces (e.g., there should be no spaces before a newline (\n) character). The `toString` value for `ex1`, `ex2`, and `ex3` respectively are shown below (the blank lines are not part of the `toString` values).

```
Ellipsoid "Ex 1" with axes a = 1.0, b = 2.0, c = 3.0 units has:
   volume = 25.1327 cubic units
   surface area = 48.9366 square units

Ellipsoid "Ex 2" with axes a = 2.3, b = 5.5, c = 7.4 units has:
   volume = 392.1127 cubic units
   surface area = 317.9245 square units

Ellipsoid "Ex 3" with axes a = 123.4, b = 234.5, c = 345.6 units has:
   volume = 41,890,963.5508 cubic units
   surface area = 674,164.7034 square units
```

Code and Test: As you implement your Ellipsoid 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 Ellipsoid in interactions (e.g., copy/paste the examples above on page 2). 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 an Ellipsoid 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 Ellipsoid then prints it out. This would be similar to the EllipsoidApp class from a previous project, except that in the EllipsoidApp class you read in the values and then create and print the object.

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

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

  Design: The EllipsoidList 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 Ellipsoid objects. These are the only fields (or instance variables) that this class should have, and both should be private.

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

  (3) Methods: The methods for EllipsoidList are described below.
  - `getName`: Returns a String representing the name of the list.
  - `numberOfEllipsoids`: Returns an int representing the number of Ellipsoid objects in the EllipsoidList. If there are zero Ellipsoid objects in the list, zero should be returned.
  - `totalVolume`: Returns a double representing the total volume for all Ellipsoid objects in the list. If there are zero Ellipsoid objects in the list, zero should be returned.
  - `totalSurfaceArea`: Returns a double representing the total surface area for all Ellipsoid objects in the list. If there are zero Ellipsoid objects in the list, zero should be returned.
  - `averageVolume`: Returns a double representing the average volume for all Ellipsoid objects in the list. If there are zero Ellipsoid objects in the list, zero should be returned.
  - `averageSurfaceArea`: Returns a double representing the average surface area for all Ellipsoid objects in the list. If there are zero Ellipsoid objects in the list, zero should be returned.
  - `toString`: Returns a String (does not begin with \n) containing the name of the list followed by each Ellipsoid 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 Ellipsoid object in the list (adding a \n before and after each). Be sure to include appropriate newline escape sequences. For an example, in the previous project see lines 3 through 16 in the output below from EllipsoidListApp for the *Ellipsoid_data_1.txt* input file. [Note that the toString result should **not** include the summary items in lines 18 through 24 of the example. These lines represent the return value of the summaryInfo method below.]

- `summaryInfo`: Returns a String (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 Ellipsoid objects, total volume, total surface area, average volume, and average surface area. Use "#,##0.0##" as the pattern to format the double values. For an example, in the previous project see lines 18 through 24 in the output below from EllipsoidListApp for the *Ellipsoid_data_1.txt* input file. The second example below shows the output from EllipsoidListApp for the *Ellipsoid_data_0.txt* input file which contains a list name but no Ellipsoid data.

- **The following six methods are new in Project 6:**
  - `getList`: Returns the ArrayList of Ellipsoid 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 Ellipsoid objects, uses the list name and the ArrayList to create an EllipsoidList object, and then returns the EllipsoidList object. See note #1 under Important Considerations for the EllipsoidListMenuApp class (last page) to see how this method should be called.
  - `addEllipsoid`: Returns nothing but takes four parameters (label, a, b, and c), creates a new Ellipsoid object, and adds it to the EllipsoidList object (i.e., adds it to the ArrayList of Ellipsoid objects in the EllipsoidList object).
  - `findEllipsoid`: Takes a label of an Ellipsoid as the String parameter and returns the corresponding Ellipsoid object if found in the EllipsoidList object; otherwise returns null. Case should be ignored when attempting to match the label.
  - `deleteEllipsoid`: Takes a String as a parameter that represents the label of the Ellipsoid and returns the Ellipsoid if it is found in the EllipsoidList object and deleted; otherwise returns null. Case should be ignored when attempting to match the label; consider calling/using `findEllipsoid` in this method.
  - `editEllipsoid`: Takes four parameters (label, a, b, and c), uses the label to find the corresponding the Ellipsoid object. If found, sets the a, b, and c to the values passed in as parameters, and returns the Ellipsoid object. If not found, returns null. ***This method should not change 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 EllipsoidListMenuApp class.

- **EllipsoidListMenuApp.java** (replaces EllipsoidListApp class from the previous project)

  Requirements: Create an EllipsoidListMenuApp 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 an EllipsoidList object, (2) print the EllipsoidList object, (3) print the summary for the EllipsoidList object, (4) add an Ellipsoid object to the EllipsoidList object, (5) delete an Ellipsoid object from the EllipsoidList object, (6) find an Ellipsoid object in the EllipsoidList object, (7) edit an Ellipsoid object in the EllipsoidList 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 an EllipsoidList 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 select.

Line #	Program output
1	`Ellipsoid List System Menu`
2	`R - Read File and Create Ellipsoid List`
3	`P - Print Ellipsoid List`
4	`S - Print Summary`
5	`A - Add Ellipsoid`
6	`D - Delete Ellipsoid`
7	`F - Find Ellipsoid`
8	`E - Edit Ellipsoid`
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 Ellipsoid 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 *Ellipsoid_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: Ellipsoid_data_1.txt`
3	`   File read in and Ellipsoid List created`
4	
5	`Enter Code [R, P, S, A, D, F, E, or Q]:`

The result of the user selecting ‘p’ to Print Ellipsoid List is shown below and next page.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: p`
2	`Ellipsoid Test List`
3	
4	`Ellipsoid "Ex 1" with axes a = 1.0, b = 2.0, c = 3.0 units has:`
5	`   volume = 25.1327 cubic units`
6	`   surface area = 48.9366 square units`
7	
8	`Ellipsoid "Ex 2" with axes a = 2.3, b = 5.5, c = 7.4 units has:`
9	`   volume = 392.1127 cubic units`
10	`   surface area = 317.9245 square units`
11	
12	`Ellipsoid "Ex 3" with axes a = 123.4, b = 234.5, c = 345.6 units has:`
13	`   volume = 41,890,963.5508 cubic units`
14	`   surface area = 674,164.7034 square units`
15	
16	`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 Ellipsoid Test List -----`
4	`Number of Ellipsoid Objects: 3`
5	`Total Volume: 41,891,380.796 cubic units`
6	`Total Surface Area: 674,531.564 square units`
7	`Average Volume: 13,963,793.599 cubic units`
8	`Average Surface Area: 224,843.855 square units`
9	
10	`Enter Code [R, P, S, A, D, F, E, or Q]:`

The result of the user selecting 'a' to add an Ellipsoid object is shown below. Note that after 'a' was entered, the user was prompted for label, radius, and height. Then after the Ellipsoid object is added to the Ellipsoid List, the message "*** Ellipsoid 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: Ex 4`
3	`   a: 10.2`
4	`   b: 12.4`
5	`   c: 14.6`
6	`   *** Ellipsoid added ***`
7	
8	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "delete" for an Ellipsoid object, followed by an attempt that was not successful (i.e., the Ellipsoid object was not found). You should do "p" to confirm the "d". Note that if found, the actual label "Ex 3" is printed below rather than "ex 3" which was entered by the user; whereas, if not found, the label entered by the user is printed.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
2	`   label: ex 3`
3	`   "Ex 3" deleted`
4	
5	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
6	`   label: ex 17`
7	`   "ex 17" not found`
8	
9	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "find" for an Ellipsoid object, followed by an attempt that was not successful (i.e., the Ellipsoid object was not found).

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
2	`   label: ex 2`
3	`Ellipsoid "Ex 2" with axes a = 2.3, b = 5.5, c = 7.4 units has:`
4	`   volume = 392.1127 cubic units`
5	`   surface area = 317.9245 square units`
6	
7	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
8	`   label: ex 7`
9	`   "ex 7" not found`
10	
11	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "edit" for an Ellipsoid object, followed by an attempt that was not successful (i.e., the Ellipsoid object was not found). In order to verify the edit, you should do a "find" for "Ex 2" or you could do a "print" to print the whole list. Note that if found, the actual label "Ex 2" is printed below rather than "ex 2" which was entered by the user; whereas, if not found, the label entered by the user is printed.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
2	`   label: ex 2`
3	`   a: 4.6`
4	`   b: 11.0`
5	`   c: 14.8`
6	`   "Ex 2" successfully edited`
7	
8	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
9	`   label: ex 13`
10	`   a: 12`
11	`   b: 13`
12	`   c: 14`
13	`   "ex 13" not found`
14	
15	`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 which successfully terminates the program.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: b`
2	`   *** invalid code ***`
3	
4	`Enter Code [R, P, S, A, D, F, E, or Q]: q`
5	

Code and Test:
Important considerations: This class should import java.util.Scanner, java.util.ArrayList, and java.io.FileNotFoundException. 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 Ellipsoid objects and then declare and create an EllipsoidList object using the list name and the ArrayList as the parameters in the constructor. This will be an EllipsoidList object that contains no Ellipsoid objects. For example:

```
String _______ = "*** no list name assigned ***";
ArrayList<Ellipsoid> _________ = new  ArrayList<Ellipsoid>();
EllipsoidList _________ = new EllipsoidList(_________,_________);
```

   The 'R' option in the menu should invoke the readFile method on your EllipsoidList object. This will return a new EllipsoidList object based on the data read from the file, and this new

EllipsoidList object should replace (be assigned to) your original EllipsoidList 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 Ellipsoid List" option, you should be able to print the EllipsoidList 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 and drag the items of interest (e.g., the Scanner on the file, your EllipsoidList object, etc.) onto the canvas and save it. As you play or step through your program, you will 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 Ellipsoid and EllipsoidList 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 EllipsoidList 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.