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.

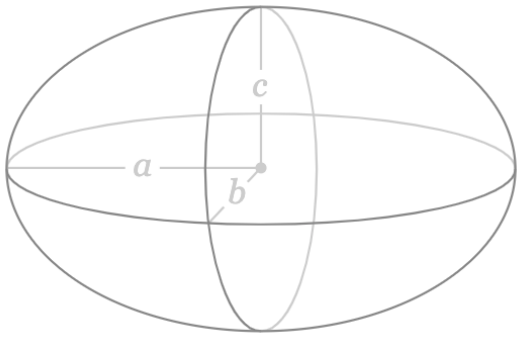
Files to submit to Web-CAT (all three files must be submitted together):

- Ellipsoid.java
- EllipsoidList.java
- EllipsoidListApp.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, EllipsoidListApp, reads in a file name entered by the user then reads the list name and Ellipsoid data from the file, creates Ellipsoid objects and stores them in an ArrayList of Ellipsoid objects, creates an EllipsoidList object with the list name and ArrayList, prints the EllipsoidList object, and then prints summary information about the EllipsoidList object.

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 the previous project, except that in the EllipsoidApp class you will read in the values and then create and print the object.

- **EllipsoidList.java**

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, 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, 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.

Code and Test: Remember to import java.util.ArrayList. Each of the last five methods above requires that you use a loop (i.e., a while loop) to retrieve each object in the ArrayList. As you implement your EllipsoidList class, you can compile it and then test it using interactions. However, it may be easier to create a class with a simple main method that creates an EllipsoidList object and calls its methods.

- **EllipsoidListApp.java**

  Requirements: Create an EllipsoidListApp class with a main method that (1) reads in the name of the data file entered by the user and (2) reads list name and Ellipsoid data from the file, (3) creates Ellipsoid objects, storing them in a local ArrayList of Ellipsoid objects; and finally, (4) creates an EllipsoidList object with the name of the list and the ArrayList of Ellipsoid objects, and then prints the EllipsoidList object followed summary information about the EllipsoidList object. **All input and output for this project must 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 data file. The first record (or line) in the file contains the name of the list. This is followed by the data for the Ellipsoid objects. Within a while loop, each set of Ellipsoid data (i.e., label, and axes a, b, and c) is read in, and an Ellipsoid object should be created and added to the local ArrayList of Ellipsoid objects. After the file has been read in and the ArrayList has been populated, the main method should create an EllipsoidList object with the name of the list and the ArrayList of Ellipsoid objects as parameters in the constructor. It should then print the EllipsoidList object, and then print the summary information about the EllipsoidList (i.e., print the value returned by the summaryInfo method for the EllipsoidList). The output from two runs of the main method in EllipsoidListApp is shown below. The first is produced after reading in the *Ellipsoid_data_1.txt* file, and the second is produced after reading in the *Ellipsoid_data_0.txt* file. Your program output should be formatted exactly as shown on the next page.

Line #	Program output
	----jGRASP exec: java EllipsoidListApp
1	Enter file name: Ellipsoid_data_1.txt
2	
3	Ellipsoid Test List
4	
5	Ellipsoid "Ex 1" with axes a = 1.0, b = 2.0, c = 3.0 units has:
6	volume = 25.1327 cubic units
7	surface area = 48.9366 square units
8	
9	Ellipsoid "Ex 2" with axes a = 2.3, b = 5.5, c = 7.4 units has:
10	volume = 392.1127 cubic units
11	surface area = 317.9245 square units
12	
13	Ellipsoid "Ex 3" with axes a = 123.4, b = 234.5, c = 345.6 units has:
14	volume = 41,890,963.5508 cubic units
15	surface area = 674,164.7034 square units
16	
17	
18	----- Summary for Ellipsoid Test List -----
19	Number of Ellipsoid Objects: 3
20	Total Volume: 41,891,380.796 cubic units
21	Total Surface Area: 674,531.564 square units
22	Average Volume: 13,963,793.599 cubic units
23	Average Surface Area: 224,843.855 square units
24	
	----jGRASP: operation complete.

Line #	Program output
	----jGRASP exec: java EllipsoidListApp
1	Enter file name: Ellipsoid_data_0.txt
2	
3	Ellipsoid Empty Test List
4	
5	
6	----- Summary for Ellipsoid Empty Test List -----
7	Number of Ellipsoid Objects: 0
8	Total Volume: 0.0 cubic units
9	Total Surface Area: 0.0 square units
10	Average Volume: 0.0 cubic units
11	Average Surface Area: 0.0 square units
12	
	----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 data 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 data file is the name of the list, and then each set of four lines contains the data from which an Ellipsoid 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 Ellipsoid 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 four lines. As each of the lines is read from the file, the respective local variables for the Ellipsoid data items (label, a, b, and c) should be assigned, after which the Ellipsoid object should be created and added to a local ArrayList of Ellipsoid objects. The next iteration of the loop should then read the next set of four lines then create the next Ellipsoid object and add it to the local ArrayList of Ellipsoid objects, 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 of Ellipsoid objects should be used to create an EllipsoidList object. The list should be printed by printing a leading \n and the EllipsoidList object. Finally, the summary information is printed by printing a leading \n and the value returned by the summaryInfo method invoked on the EllipsoidList object.

Test: You should test your program minimally (1) by reading in the *Ellipsoid_data_1.txt* input file, which should produce the first output above, and (2) by reading in the *Ellipsoid_data_0.txt* input file, which should produce the second output above. Although your program may not use all of the methods in the EllipsoidList and Ellipsoid classes, you should ensure that all of your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.

General Notes

1. All input from the keyboard and all output to the screen should 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/or System.out.println methods) should be in the main method. Hence, none of your methods in the Ellipsoid class should do any input/output (I/O).

2. Be sure to download the test data files (*Ellipsoid_data_1.txt* and *Ellipsoid_data_0.txt*) and store them in same folder as your source files. It may be useful 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.