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.

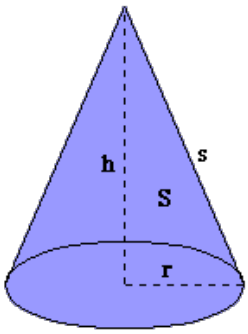
Files to submit to Web-CAT (all three files must be submitted together):

- Cone.java
- ConeList.java
- ConeListApp.java

Specifications

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

A **cone** is a 3-D geometric shape that tapers smoothly from a flat base to a point called the apex or vertex. A cone can be defined by its height and radius (h, r) as depicted below. The formulas are provided to assist you in computing return values for the respective methods in the Cone class described in this project. [Wikipedia]



Nomenclature

height: h
radius of base: r
perimeter of base: P
base area: B
slant height: s
side area: S
total surface area: T
Volume: V

Formulas

$P = 2\pi r$
$B = \pi r^2$
$s = \sqrt{r^2 + h^2}$
$S = \pi rs$
$T = \pi r(r + s)$
$V = \pi r^2 h/3$

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

  Requirements: Create a Cone class that stores the label, height, and radius (height and radius each must be greater than zero). The Cone class also includes methods to set and get each of these fields, as well as methods to calculate the base perimeter, base area, slant height, side area, surface area, and volume of a Cone object, and a method to provide a String value of a Cone object (i.e., an instance of the Cone class).

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

  (1) Fields (instance variables): label of type String, height of type double, and radius of type double. These instance variables should be private so that they are not directly accessible from outside of the Cone class, and <u>these should be the only instance variables in the class</u>.

  (2) Constructor: Your Cone class must contain a constructor that accepts three parameters (see types of above) representing the label, height, and radius. 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 Cone 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.

```
Cone example1 = new Cone("Short Example", 3.0, 4.0);
Cone example2 = new Cone(" Wide Example ", 10.6, 22.1);
Cone example3 = new Cone("Tall Example", 100, 20);
```

  (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 Cone are described below. See the formulas above and "Code and Test" below for details. When implementing the formulas, be sure to use the Math constant Math.PI for π and 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 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 greater than zero, sets height field and returns true. Otherwise, the method returns false and the height is not set.
  - `getRadius`: Accepts no parameters and returns a double representing the radius field.

- `setRadius`: Accepts a double parameter and returns a boolean. If the radius is greater than zero, sets radius field and returns true. Otherwise, the method returns false and the radius is not set.
- `basePerimeter`: Accepts no parameters and returns the double value for the perimeter of the base circle of the cone calculated using radius.
- `baseArea`: Accepts no parameters and returns the double value for the base area calculated using radius.
- `slantHeight`: Accepts no parameters and returns the double value for the slant height calculated using height and radius.
- `sideArea`: Accepts no parameters and returns the double value for the side area calculated using radius and slant height.
- `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using the base area and side area.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using height and radius.
- `toString`: Returns a String containing the information about the Cone object formatted as shown below, including decimal formatting ("#,##0.0##") for the double values. Newline 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: basePerimeter(), baseArea(), slantHeight(), sideArea(), surfaceArea(), and volume(). Each line should have no leading and 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).

```
"Short Example" is a cone with height = 3.0 units and radius = 4.0 units,
which has base perimeter = 25.133 units, base area = 50.265 square units,
slant height = 5.0 units, side area = 62.832 square units,
surface area = 113.097 square units, and volume = 50.265 cubic units.

"Wide Example" is a cone with height = 10.6 units and radius = 22.1 units,
which has base perimeter = 138.858 units, base area = 1,534.385 square units,
slant height = 24.511 units, side area = 1,701.752 square units,
surface area = 3,236.137 square units, and volume = 5,421.495 cubic units.

"Tall Example" is a cone with height = 100.0 units and radius = 20.0 units,
which has base perimeter = 125.664 units, base area = 1,256.637 square units,
slant height = 101.98 units, side area = 6,407.617 square units,
surface area = 7,664.254 square units, and volume = 41,887.902 cubic units.
```

Code and Test: As you implement your Cone 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 Cone 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 Cone 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 Cone then prints is out. This would be similar to the class you will create in ConeApp, except that in ConeApp you will read in the values and then create the object.

- **ConeList.java**

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

Design: The ConeList 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 Cone objects. These instance variables should be private so that they are not directly accessible from outside of the ConeList class, and these should be the only instance variables in the class.

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

(3) Methods: The methods for ConeList are described below.

- `getName`: Returns a String representing the name of the list.
- `numberOfCones`: Returns an int representing the number of Cone objects in the ConeList. If there are zero Cone objects in the list, zero should be returned.
- `totalBasePerimeter`: Returns a double representing the total for the base perimeters for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `totalBaseArea`: Returns a double representing the total for the base areas for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `totalSlantHeight`: Returns a double representing the total for the slant heights for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `totalSideArea`: Returns a double representing the total for the side areas for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `totalSurfaceArea`: Returns a double representing the total surface areas for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `totalVolume`: Returns a double representing the total volumes for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.

- `averageSurfaceArea`: Returns a double representing the average surface area for all Cone objects in the list. If there are zero Cone objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all Cone objects in the list. If there are zero Cone 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 Cone 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 Cone object in the list. Be sure to include appropriate newline escape sequences. For an example, see lines 3 through 24 in the output below from ConeListApp for the *cone_1.txt* input file. [Note that the toString result should **not** include the summary items in lines 26 through 35. These lines represent the return value of the summaryInfo method below.]
- `summaryInfo`: Returns a String (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 cones, total base perimeter, total base area, total slant heights, total side areas, total surface area, total volume, average surface area, average volume. For an example, see lines 26 through 35 in the output below from ConeListApp for the *cone_1.txt* input file. The second example below shows the output from ConeListApp for the *cone_0.txt* input file which contains a list name but no cone data.

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

- **ConeListApp.java**

Requirements: Create a ConeListApp class with a main method that (1) reads in the name of the input file entered by the user and then (2) reads list name and Cone data from the file, (3) creates Cone objects, stores them in a local ArrayList of Cone objects, and finally (4) creates an ConeList object with the name of the list and the ArrayList of Cone objects, and then prints the ConeList object followed summary information about the ConeList object. **All input and output for this project should be done in the main method**.

Design: The main method should prompt the user to enter a file name, and then it should read in the file. The first record (or line) in the file contains the name of the list. This is followed by the data for the Cone objects. After each set of Cone data is read in, a Cone object should be created and stored in the local ArrayList. After the file has been read in and the ArrayList created, the main method should create a ConeList object with the name of the list and the ArrayList of Cone objects as parameters in the constructor. It should then print the ConeList object followed by summary information about the ConeList (i.e., print the value returned by the summaryInfo method for the ConeList). The output from two runs of the main method in ConeListApp is shown below: the first produced after reading in the *cone_1.txt* file and the second after reading in the *cone_0.txt* file. Your program output should be formatted exactly as follows).

Line #	Program output
	----jGRASP exec: java ConeListApp
1	Enter file name: cone_1.txt
2	
3	Cone List 1
4	
5	"Short Example" is a cone with height = 3.0 units and radius = 4.0 units,
6	which has base perimeter = 25.133 units, base area = 50.265 square units,
7	slant height = 5.0 units, side area = 62.832 square units,
8	surface area = 113.097 square units, and volume = 50.265 cubic units.
9	
10	"Wide Example" is a cone with height = 10.6 units and radius = 22.1 units,
11	which has base perimeter = 138.858 units, base area = 1,534.385 square units,
12	slant height = 24.511 units, side area = 1,701.752 square units,
13	surface area = 3,236.137 square units, and volume = 5,421.495 cubic units.
14	
15	"Tall Example" is a cone with height = 100.0 units and radius = 20.0 units,
16	which has base perimeter = 125.664 units, base area = 1,256.637 square units,
17	slant height = 101.98 units, side area = 6,407.617 square units,
18	surface area = 7,664.254 square units, and volume = 41,887.902 cubic units.
19	
20	"Really Large Example" is a cone with height = 300.0 units and radius = 400.0 units,
21	which has base perimeter = 2,513.274 units, base area = 502,654.825 square units,
22	slant height = 500.0 units, side area = 628,318.531 square units,
23	surface area = 1,130,973.355 square units, and volume = 50,265,482.457 cubic units.
24	
25	
26	----- Summary for Cone List 1 -----
27	Number of Cones: 4
28	Total Base Perimeter: 2,802.929
29	Total Base Area: 505,496.112
30	Total Slant Height: 631.491
31	Total Side Area: 636,490.731
32	Total Surface Area: 1,141,986.844
33	Total Volume: 50,312,842.12
34	Average Surface Area: 285,496.711
35	Average Volume: 12,578,210.53
	----jGRASP: operation complete.

Line #	Program output
	----jGRASP exec: java ConeListApp
1	Enter file name: cone_0.txt
2	
3	Empty List of Cones
4	
5	
6	----- Summary for Empty List of Cones -----
7	Number of Cones: 0
8	Total Base Perimeter: 0.0
9	Total Base Area: 0.0
10	Total Slant Height: 0.0
11	Total Side Area: 0.0
12	Total Surface Area: 0.0
13	Total Volume: 0.0
14	Average Surface Area: 0.0
15	Average Volume: 0.0

```
----jGRASP: operation complete.
```

Code: Remember to import java.util.ArrayList, java.util.Scanner, and java.io.File, and java.io.IOException prior to the class declaration. Your main method declaration should indicate that main `throws IOException`. After your program reads in the file name from the keyboard, it should read in the file using a Scanner object that was created on a file using the file name entered by the user.

```
... = new Scanner(new File(fileName));
```

You can assume that the first line in the file is the name of the list, and then each set of three lines contains the data from which a Cone 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 cone data. The boolean expression for the while loop should be `(_____.hasNext())` where the blank is the name of the Scanner you created on the file. Each iteration through the loop reads three lines. As each of the lines is read from the file, the respective local variables for the cone data items (label, side, and height) should be assigned, after which the Cone object should be created and added to a local ArrayList. The next iteration of the loop should then read the next set of three lines then create the next Cone object and add it to a local ArrayList, and so on. After the file has been processed (i.e., when the loop terminates after the hasNext method returns false), name of the list and the ArrayList should be used to create a ConeList object. The ConeList object should then be printed. Finally, the summary information is printed by printing the value returned by the summaryInfo method invoked on the ConeList object.

Test: You should test your program minimally (1) by reading in the *cone_1.txt* input file, which should produce the first output above, and (2) by reading in the *cone_0.txt* input file, which should produce the second output above. Although your program may not use all of the methods in ConeList and Cone, 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 System.out.println methods) should be in the main method. Hence, none of your methods in the Cone class should do any input/output (I/O).

2. Be sure to download the test data files (*cone_1.txt* and *cone_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.