Project 6: Cone List Menu App

Due: Skeleton Code (ungraded - checks class, method names, etc.)
Completed Code (100%) – Monday, Oct 10, 2016 by 11:59 PM

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until the project due date but should try to do this by Friday (there is no late penalty since this is ungraded for this 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 to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

Files to submit to Web-CAT:

- Cone.java
- ConeList.java
- ConeListMenuApp.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, ConeListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates a ConeList object), (2) print report, (3) print summary, (4) add a Cone object to the ConeList object, (5) delete a Cone object from the ConeList object, (6) find a Cone object in the ConeList object, (7) Edit a Cone in the ConeList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (Cone.java, ConeList.java, cone_0.dat, and cone_1.dat) **to it, rather than work in the same folder as Project 5 files.]**

- **Cone.java (assuming that you successfully created this class in Project 4, just copy the file to your new Project 5 folder and go on to ConeList.java on page 4. Otherwise, you will need to create Cone.java as part of this project.)**

  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., a class instance).

  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 these should be the only instance variables in the class.

(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 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 "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.

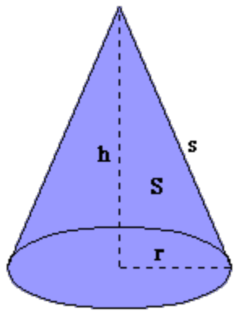
- `toString`: Returns a String containing the information about the Cone object formatted as shown below, including decimal formatting ("#,##0.0##") for the double values. Newline escape sequences should be used to achieve the proper layout. In addition to the field values (or corresponding "get" methods), 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 Part 2, except that in in Part 2 you will read in the values and then create the object.

The formulas below are provided to assist you in computing return values for the respective cone methods described above. `Math.PI` should be used for Pi. (adapted from http://mathforum.org/dr.math/faq/formulas/faq.cone.html).

height: h
radius of base: r
perimeter of base: P = Pi 2r
base area: B = Pi r^2
slant height: s = sqrt(r^2+h^2)
side area: S = Pi rs
total surface area: T = Pi r(r+s)
Volume: V = Pi r^2h/3



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

  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 are the only fields (or instance variables) that this class should have.

  (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 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 <u>lines 3 through 21</u> in the output below from ConeListApp for the *cone_1.dat* input file. [Note that t<u>he toString result should **not** include the summary items in lines 22 through 30 of the example. These lines represent the return value of the summaryInfo method below.</u>]
- `summaryInfo`: Returns a String 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 <u>lines 22 through 30</u> in the output below from ConeListApp for the *cone_1.dat* input file. The second example below shows the output from ConeListApp for the *cone_0.dat* input file which contains a list name but no cone data.

- **<u>The following six methods are new in Project 6</u>:**
  - `getList`: Returns the ArrayList of Cone 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 Cone objects, uses the list name and the ArrayList to create a ConeList object, and then returns the ConeList object. See note #1 under <u>Important Considerations</u> for the ConeListMenuApp class (last page) to see how this method should be called.
  - `addCone`: Returns nothing but takes three parameters (label, height, and radius), creates a new Cone object, and adds it to the ConeList object.
  - `findCone`: Takes a label of a Cone as the String parameter and returns the corresponding Cone object if found in the ConeList object; otherwise returns null. This method should <u>ignore case when attempting to match the label</u>.
  - `deleteCone`: Takes a String as a parameter that represents the label of the Cone and returns the Cone if it is found in the ConeList object and deleted; otherwise returns null. <u>This method should use the String equalsIgnoreCase method when attempting to match a label in the Cone object to delete</u>.
  - `editCone`: Takes three parameters (label, height, and radius), uses the label to find the corresponding the Cone object. If found, sets the height and radius to the values passed in as parameters, and returns true. If not found, simply returns false.

Code and Test: Remember to import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.IOException. These classes will be needed in the readFile method which will require a throws clause for IOException. 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 ConeListMenuApp class.

- **ConeListMenuApp.java** (replaces the previous ConeListApp class in Project 5)

  Requirements: Create a ConeListMenuApp 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 ConeList object, (2) print the ConeList object, (3) print the summary for the ConeList object, (4) add a Cone object to the ConeList object, (5) delete a Cone object from the ConeList object, (6) find a Cone object in the ConeList object, (7) Edit a Cone object in the ConeList object, and (7) 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 ConeList 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.

```
Cone List System Menu
R - Read File and Create Cone List
P - Print Cone List
S - Print Summary
A - Add Cone
D - Delete Cone
F - Find Cone
E - Edit Cone
Q - Quit
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 Cone 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 *cone_1.dat* file from Project 5 to test your program.

```
Enter Code [R, P, S, A, D, F, E, or Q]: r
   File name: cone_1.dat
   File read in and Cone List created

Enter Code [R, P, S, A, D, F, E, or Q]:
```

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

```
Enter Code [R, P, S, A, D, F, E, or Q]: p

Cone List 1

"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.

"Really Large Example" is a cone with height = 300.0 units and radius = 400.0 units,
which has base perimeter = 2,513.274 units, base area = 502,654.825 square units,
slant height = 500.0 units, side area = 628,318.531 square units,
surface area = 1,130,973.355 square units, and volume = 50,265,482.457 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.

```
Enter Code [R, P, S, A, D, F, E, or Q]: s

----- Summary for Cone List 1 -----
Number of Cones: 4
Total Base Perimeter: 2,802.929
Total Base Area: 505,496.112
Total Slant Height: 631.491
Total Side Area: 636,490.731
Total Surface Area: 1,141,986.844
Total Volume: 50,312,842.12
Average Surface Area: 285,496.711
Average Volume: 12,578,210.53

Enter Code [R, P, S, A, D, F, E, or Q]:
```

The result of the user selecting ‘a’ to add a Cone object is shown below. Note that after ‘a’ was entered, the user was prompted for label, height, and radius. Then after the Cone object is added to the Cone List, the message “*** Cone 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.

```
Enter Code [R, P, S, A, D, F, E, or Q]: a
   Label: my new cone
   Height: 7.5
   Radius: 9.5
   *** Cone added ***

Enter Code [R, P, S, A, D, F, E, or Q]:
```

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

```
Enter Code [R, P, S, A, D, F, E, or Q]: d
   Label: Tall Example
   "Tall Example" deleted

Enter Code [R, P, S, A, D, F, E, or Q]: d
   Label: my fake cone
   "my fake cone" not found

Enter Code [R, P, S, A, D, F, E, or Q]:
```

Here is an example of the successful “find” for a Cone object, followed by an attempt that was not successful (i.e., the Cone object was not found), and finally an example of entering an invalid code.

```
Enter Code [R, P, S, A, D, F, E, or Q]: f
   Label: Wide Example
"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.

Enter Code [R, P, S, A, D, F, E, or Q]: f
   Label: another fake cone
   "another fake cone" not found

Enter Code [R, P, S, A, D, F, E, or Q]: x
   *** invalid code ***

Enter Code [R, P, S, A, D, F, E, or Q]:
```

Here is an example of the successful "edit" for a Cone object, followed by an attempt that was not successful (i.e., the Cone 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: wide example
3          Height: 11.5
4          Radius: 33.5
5          "wide example" successfully edited
6
7       Enter Code [R, P, S, A, D, F, E, or Q]: e
8          Label: fake
9          Height: 12
10         Radius: 14
11         "fake" 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
3        ----jGRASP: operation complete.
```

Code and Test:
Important considerations: This class should import java.util.Scanner, java.util.ArrayList, and java.io.IOException. 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 Cone objects and then declare and create a ConeList object using the list name and the ArrayList as the parameters in the constructor. This will be a ConeList object that contains no Cone objects. For example:
```
String _______ = "*** no list name assigned ***";
ArrayList<Cone> _________ = new  ArrayList<Cone>();
ConeList _________ = new ConeList(_________,_________);
```
   The 'R' option in the menu should invoke the readFile method on your ConeList object. This will return a new ConeList object based on the data read from the file, and this new ConeList object should replace (be assigned to) your original ConeList object variable in main. Since the readFile method throws IOException, your main method need 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 Cone List” option, you should be able to print the ConeList 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 ConeList 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 of the methods in your Cone and ConeList 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 ConeList 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.