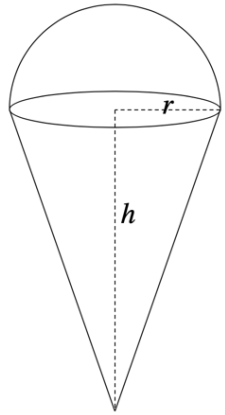## Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. You may submit your files to the <u>skeleton code</u> 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 <u>completed code</u> 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):
- IceCreamCone.java
- IceCreamConeList.java
- IceCreamConeListMenuApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines IceCreamCone objects, the second class defines IceCreamConeList objects, and the third, IceCreamConeListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates an IceCreamConeList object), (2) print report, (3) print summary, (4) add an IceCreamCone object to the IceCreamConeList object, (5) delete an IceCreamCone object from the IceCreamConeList object, (6) find an IceCreamCone object in the IceCreamConeList object, (7) Edit an IceCreamCone in the IceCreamConeList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (IceCreamCone.java, IceCreamConeList.java, IceCreamCone_data_1.txt, and IceCreamCone_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

An **ice cream cone** is a cone with a hemisphere on top as depicted below with cone height h and cone radius r, where r is also the radius of the hemisphere. The formulas are provided to assist you in computing return values for the respective methods in the IceCreamCone class described in this project.

	radius (r) height (h) Cone Side Area (cA) Hemisphere Area (hA) Surface Area (A) Cone Volume (cV) Hemisphere Volume (hV) Volume (V)	$cA = \pi r \sqrt{h^2 + r^2}$ $hA = 2\pi r^2$ $A = cA + hA$ $cV = h\pi r^2/3$ $hV = 2\pi r^3/3$ $V = cV + hV$

- **IceCreamCone.java (<u>assuming that you successfully created this class in Project 4 or 5, just copy the file to your new Project 6 folder and go on to IceCreamConeList.java on page 4. Otherwise, you will need to create IceCreamCone.java as part of this project.</u>)**

Requirements: Create an IceCreamCone class that stores the label, radius, and height. The radius and height <u>must be greater than zero</u>. The IceCreamCone class also includes methods to set and get each of these fields, as well as methods to calculate the surface area and volume of the cone, hemisphere, and the IceCreamCone object, and a method to provide a String value of an IceCreamCone object (i.e., a class instance).

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

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

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

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

IceCreamCone ex2 = new IceCreamCone(" Ex 2   ", 12.3, 25.5);

IceCreamCone ex3 = new IceCreamCone("Ex 3", 123.4, 900);
```

(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 IceCreamCone, 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 "<u>trimmed</u>" String and the method returns true. Otherwise, the method returns false and the label field is not set.
   - `getRadius`: Accepts no parameters and returns a double representing the radius field.
   - `setRadius`: Accepts a double parameter and returns a boolean as follows. If the double is greater than zero, sets the radius field to the double passed in and returns true. Otherwise, the method returns false and does not set the radius field.
   - `getHeight`: Accepts no parameters and returns a double representing the height field.

- o `setHeight`: Accepts a double parameter and returns a boolean as follows. If the double is greater than zero, sets the height field to the double passed in and returns true. Otherwise, the method returns false and does not set the height field.

- o `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using formula above and the values of the radius and height fields.

- o `volume`: Accepts no parameters and returns the double value for the volume calculated using formula above and the values of the radius and height fields.

- o `toString`: Returns a String containing the information about the IceCreamCone 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: `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 `ex1`, `ex2`, and `ex3` respectively are shown below (the blank lines are not part of the `toString` values).

```
IceCreamCone "Ex 1" with radius = 1.0 and height = 2.0 units has:
   surface area = 13.308 square units
   volume = 4.1887902 cubic units

IceCreamCone "Ex 2" with radius = 12.3 and height = 25.5 units has:
   surface area = 2,044.5837657 square units
   volume = 7,937.3689278 cubic units

IceCreamCone "Ex 3" with radius = 123.4 and height = 900.0 units has:
   surface area = 447,847.2056927 square units
   volume = 18,287,175.0307675 cubic units
```

Code and Test: As you implement your IceCreamCone 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 IceCreamCone 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 an IceCreamCone 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 IceCreamCone then prints it out. This would be similar to the IceCreamConeApp class you will create below, except that in the IceCreamConeApp class you will read in the values and then create and print the object.

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

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

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

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

(3) **Methods**: The methods for IceCreamConeList are described below.
   - `getName`: Returns a String representing the name of the list.
   - `numberOfIceCreamCones`: Returns an int representing the number of IceCreamCone objects in the IceCreamConeList. If there are zero IceCreamCone objects in the list, zero should be returned.
   - `totalSurfaceArea`: Returns a double representing the total surface area for all IceCreamCone objects in the list. If there are zero IceCreamCone objects in the list, zero should be returned.
   - `totalVolume`: Returns a double representing the total volume for all IceCreamCone objects in the list. If there are zero IceCreamCone objects in the list, zero should be returned.
   - `averageSurfaceArea`: Returns a double representing the average surface area for all IceCreamCone objects in the list. If there are zero IceCreamCone objects in the list, zero should be returned.
   - `averageVolume`: Returns a double representing the average volume for all IceCreamCone objects in the list. If there are zero IceCreamCone 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 IceCreamCone 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 IceCreamCone object in the list (adding a \n before and after each). Be sure to include appropriate newline escape sequences. For an example, see <u>lines 3 through 16</u> in the output below from IceCreamConeListApp for the *IceCreamCone_data_1.txt* input file. [Note that <u>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.</u>]

o `summaryInfo`: Returns a String (does<u> not </u>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 IceCreamCone objects, 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 18 through 24</u> in the output below from IceCreamConeListApp for the *IceCreamCone_data_1.txt* input file. The second example below shows the output from IceCreamConeListApp for the *IceCreamCone_data_0.txt* input file which contains a list name but no IceCreamCone data.

• **The following six methods are new in Project 6:**
   o `getList`: Returns the ArrayList of IceCreamCone 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 IceCreamCone objects, uses the list name and the ArrayList to create an IceCreamConeList object, and then returns the IceCreamConeList object. See note #1 under <u>Important Considerations</u> for the IceCreamConeListMenuApp class (last page) to see how this method should be called.

   o `addIceCreamCone`: Returns nothing but takes three parameters (label, radius, and height), creates a new IceCreamCone object, and adds it to the IceCreamConeList object.

   o `findIceCreamCone`: Takes a label of an IceCreamCone as the String parameter and returns the corresponding IceCreamCone object if found in the IceCreamConeList object; otherwise returns null. <u>Case should be ignored when attempting to match the label</u>.

   o `deleteIceCreamCone`: Takes a String as a parameter that represents the label of the IceCreamCone and returns the IceCreamCone if it is found in the IceCreamConeList object and deleted; otherwise returns null. <u>Case should be ignored when attempting to match the label; consider calling/using `findIceCreamCone` in this method</u>.

   o `editIceCreamCone`: Takes three parameters (label, radius, and height), uses the label to find the corresponding the IceCreamCone object. If found, sets the radius and height to the values passed in as parameters, and returns true. If not found, returns false. This method should <u>not</u> 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 IceCreamConeListMenuApp class.

- **IceCreamConeListMenuApp.java** (replaces IceCreamConeListApp class from Project 5)

Requirements: Create an IceCreamConeListMenuApp 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 IceCreamConeList object, (2) print the IceCreamConeList object, (3) print the summary for the IceCreamConeList object, (4) add an IceCreamCone object to the IceCreamConeList object, (5) delete an IceCreamCone object from the IceCreamConeList object, (6) find an IceCreamCone object in the IceCreamConeList object, (7) edit an IceCreamCone object in the IceCreamConeList 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 IceCreamConeList 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 <u>accept both uppercase and lowercase</u> 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	IceCreamCone List System Menu
2	R – Read File and Create IceCreamCone List
3	P – Print IceCreamCone List
4	S – Print Summary
5	A – Add IceCreamCone
6	D – Delete IceCreamCone
7	F – Find IceCreamCone
8	E – Edit IceCreamCone
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 IceCreamCone 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 *IceCreamCone_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: IceCreamCone_data_1.txt
3	File read in and IceCreamCone List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'p' to Print IceCreamCone List is shown below and next page.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: p
2	IceCreamCone Test List
3	
4	IceCreamCone "Ex1" with radius = 1.0 and height = 2.0 units has:
5	surface area = 13.308 square units
6	volume = 4.1887902 cubic units
7	
8	IceCreamCone "Ex 2" with radius = 12.3 and height = 25.5 units has:
9	surface area = 2,044.5837657 square units
10	volume = 7,937.3689278 cubic units
11	
12	IceCreamCone "Ex 3" with radius = 123.4 and height = 900.0 units has:
13	surface area = 447,847.2056927 square units
14	volume = 18,287,175.0307675 cubic 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 IceCreamCone Test List -----
4	Number of IceCreamCone Objects: 3
5	Total Surface Area: 449,905.097
6	Total Volume: 18,295,116.588
7	Average Surface Area: 149,968.366
8	Average Volume: 6,098,372.196
9	
10	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'a' to add an IceCreamCone object is shown below. Note that after 'a' was entered, the user was prompted for label, radius, and height. Then after the IceCreamCone object is added to the IceCreamCone List, the message "*** IceCreamCone 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	Radius: 15.8
4	Height: 33.0
5	*** IceCreamCone added ***
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful "delete" for an IceCreamCone object, followed by an attempt that was not successful (i.e., the IceCreamCone object was not found). You should do "p" to confirm the "d".

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 10
7	"ex 10" 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 IceCreamCone object, followed by an attempt that was not successful (i.e., the IceCreamCone 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	IceCreamCone "Ex 2" with radius = 12.3 and height = 25.5 units has:
4	surface area = 2,044.5837657 square units
5	volume = 7,937.3689278 cubic 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 IceCreamCone object, followed by an attempt that was <u>not</u> successful (i.e., the IceCreamCone object was not found). In order to verify the edit, you should do a "find" for "medium" 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: Ex 2
3	Radius: 27.5
4	Height: 50.2
5	"Ex 2" successfully edited
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: e
8	Label: ex 99
9	Radius: 23
10	Height: 34
11	"ex 99" 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 which successfully terminates the program.

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

Code and Test:

<u>Important considerations</u>: 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 IceCreamCone objects and then declare and create an IceCreamConeList object using the list name and the ArrayList as the parameters in the constructor. This will be an IceCreamConeList object that contains no IceCreamCone objects. For example:

```
String _____ = "*** no list name assigned ***";
ArrayList<IceCreamCone> _____ = new  ArrayList<IceCreamCone>();
IceCreamConeList _____ = new IceCreamConeList(_____,_____);
```

The 'R' option in the menu should invoke the readFile method on your IceCreamConeList object. This will return a new IceCreamConeList object based on the data read from the file, and this new IceCreamConeList object should replace (be assigned to) your original IceCreamConeList 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 IceCreamCone List" option, you should be able to print the IceCreamConeList 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 IceCreamConeList 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 IceCreamCone and IceCreamConeList 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 IceCreamConeList 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.