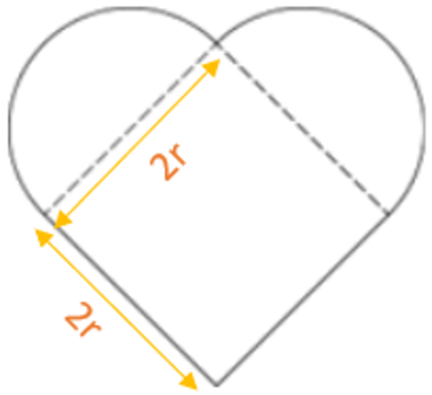
Deliverables

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):

- HeartShapedBox.java
- HeartShapedBoxList.java
- HeartShapedBoxListMenuApp.java

A **Heart Shaped Box** has heart shaped top and bottom with height **h**. The heart shape, shown in the figure below, is composed of two half circles, each with radius **r**, and a square with sides of length **2r**. The formulas are provided to assist you in computing return values for the respective methods in the HeartShapedBox class described in this project.



Formulas for area of heart shape (A_{Heart}), lateral surface area ($A_{Lateral}$), total surface area (A_{Total}), and volume (V) are shown below where r is the *radius* and h is the *height* (the distance between the top and bottom of the box), both of which will be read in.

$$A_{Heart} = \pi r^2 + 4r^2$$

$$A_{Lateral} = h(4r + 2\pi r)$$

$$A_{Total} = 2\,A_{Heart} + A_{Lateral}$$

$$V = h\,A_{Heart}$$

Specifications

Overview: You will write a program this week that is composed of three classes: HeartShapedBox, HeartShapedBoxList, and HeartShapedBoxListMenuApp, which presents a menu to the user with eight options and implements these options: (1) read input file (which creates a HeartShapedBoxList object), (2) print report, (3) print summary, (4) add a HeartShapedBox object to the HeartShapedBoxList object, (5) delete a HeartShapedBox object from the HeartShapedBoxList object, (6) find a HeartShapedBox object in the HeartShapedBoxList object, (7) Edit a HeartShapedBox in the HeartShapedBoxList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (HeartShapedBox.java, HeartShapedBoxList.java, HeartShapedBox_data_1.txt, and HeartShapedBox_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

- **HeartShapedBox.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 HeartShapedBoxList.java on page 4. Otherwise, you will need to create HeartShapedBox.java as part of this project</u>.)**

  Requirements: Create a HeartShapedBox class that stores the label, radius, and height. The HeartShapedBox class also includes methods to set and get each of these fields, as well as methods to calculate the area of the heart shape, lateral surface area, total surface area, and volume of a HeartShapedBox object, and a method to provide a String value of a HeartShapedBox object (i.e., a class instance).

  Design: The HeartShapedBox 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 doubles to zero in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the HeartShapedBox class, and <u>these should be the only instance variables in the class</u>.

  (2) **Constructor**: Your HeartShapedBox 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 HeartShapedBox 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.

```
HeartShapedBox ex1 = new HeartShapedBox ("Ex 1", 4.0, 1.5);

HeartShapedBox ex2 = new HeartShapedBox ("   Ex 2   ", 10.0, 2.4);

HeartShapedBox ex3 = new HeartShapedBox ("Ex 3", 15.5, 4.5);
```

  (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 HeartShapedBox, 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 parameter is greater than zero, sets the radius field to the double passed in and returns true. Otherwise, the method returns false, and the radius field is not set.
- `getHeight`: Accepts no parameters and returns a double representing the height field.
- `setHeight`: Accepts a double parameter and returns a boolean as follows. If the parameter is greater than zero, sets the height field to the double passed in and returns true. Otherwise, the method returns false, and the height field is not set.
- `heartShapedArea`: Accepts no parameters and returns the double value for the area of the heart shape calculated using the formula above.
- `lateralSurfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.
- `totalSurfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using the formula above.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using the using the formula above.
- `toString`: Returns a String containing the information about the HeartShapedBox 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: heartShapedArea(), lateralSurfaceArea(), totalSurfaceArea(), 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 results of printing the toString value of ex1, ex2, and ex3 respectively are shown below.

```
HeartShapedBox "Ex 1" with radius 4.0 units and height 1.5 units:
   heart shaped area = 114.265 square units
   lateral surface area = 61.699 square units
   total surface area = 290.23 square units
   volume = 171.398 cubic units

HeartShapedBox "Ex 2" with radius 10.0 units and height 2.4 units:
   heart shaped area = 714.159 square units
   lateral surface area = 246.796 square units
   total surface area = 1,675.115 square units
   volume = 1,713.982 cubic units


HeartShapedBox "Ex 3" with radius 15.5 units and height 4.5 units:
   heart shaped area = 1,715.768 square units
   lateral surface area = 717.252 square units
   total surface area = 4,148.787 square units
   volume = 7,720.954 cubic units
```

Code and Test: As you implement your HeartShapedBox 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 HeartShapedBox 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 HeartShapedBox 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 HeartShapedBox then prints it out. This would be similar to the HeartShapedBoxApp class you created in the previous project, except that in the HeartShapedBoxApp class you read in the values and then create and print the object.

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

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

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

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

  (3) Methods: The methods for HeartShapedBoxList are described below.
  - `getName`: Returns a String representing the name of the list.
  - `numberOfHeartShapedBoxes`: Returns an int representing the number of HeartShapedBox objects in the HeartShapedBoxList. If there are zero HeartShapedBox objects in the list, zero should be returned.
  - `grandTotalSurfaceArea`: Returns a double representing the sum of the individual total surface areas for all HeartShapedBox objects in the list. If there are zero HeartShapedBox objects in the list, zero should be returned.

- `totalVolume`: Returns a double representing the total volume for all HeartShapedBox objects in the list. If there are zero HeartShapedBox objects in the list, zero should be returned.
- `averageTotalSurfaceArea`: Returns a double representing the average total surface area for all HeartShapedBox objects in the list. If there are zero HeartShapedBox objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all HeartShapedBox objects in the list. If there are zero HeartShapedBox 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 HeartShapedBox 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 HeartShapedBox 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 22</u> in the output in the previous project from HeartShapedBoxListApp for the *HeartShapedBox_data_1.txt* input file. [Note that t<u>he toString result should **not** include the summary items in lines 24 through 30 of Example 1. These lines represent the return value of the summaryInfo method below.</u>]
- `summaryInfo`: Returns a String (does <u>not</u> begin with \n) containing the name of the list (which can change depending on the value read from the file) followed by various summary items: number of HeartShapedBox objects, total surface area, total volume, average surface area, and average volume. Use `"#,##0.0##"` as the pattern to format the double values. In Example 1, see <u>lines 24 through 30</u> in the output in the previous project from HeartShapedBoxListApp for the *HeartShapedBox_data_1.txt* input file. Example 2 shows the output from HeartShapedBoxListApp for the *HeartShapedBox_data_0.txt* input file which contains a list name but no HeartShapedBox data.

<u>**The following six methods are new in this Project**</u>**:**

- `getList`: Returns the ArrayList of HeartShapedBox 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 HeartShapedBox objects, uses the list name and the ArrayList to create a HeartShapedBoxList object, and then returns the HeartShapedBoxList object. See note #1 under <u>Important Considerations</u> for the HeartShapedBoxListMenuApp class (last page) to see how this method should be called. The `readFile` method header should include the `throws FileNotFoundException` clause.
- `addHeartShapedBox`: Returns nothing but takes three parameters (label, radius, and height), creates a new HeartShapedBox object, and adds it to the HeartShapedBoxList object.
- `findHeartShapedBox`: Takes a label of a HeartShapedBox as the String parameter and returns the corresponding HeartShapedBox object if found in the HeartShapedBoxList object; otherwise returns null. This method should <u>ignore case when attempting to match the label</u>.
- `deleteHeartShapedBox`: Takes a String as a parameter that represents the label of the HeartShapedBox and returns the HeartShapedBox if it is found in the HeartShapedBoxList object and deleted; otherwise returns null. <u>This method should use</u>

the String equalsIgnoreCase method when attempting to match a label in the HeartShapedBox object to delete.

- `editHeartShapedBox`: Takes three parameters (label, radius, and height), uses the label to find the corresponding the HeartShapedBox object. If found, sets the radius and height to the values passed in as parameters, and returns true. If not found, simply returns false. Note that this method should not set 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 the menu described below in the HeartShapedBoxListMenuApp class.

- **HeartShapedBoxListMenuApp.java** (replaces HeartShapedBoxListApp class from the previous project)

Requirements: Create a HeartShapedBoxListMenuApp 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 HeartShapedBoxList object, (2) print the HeartShapedBoxList object, (3) print the summary for the HeartShapedBoxList object, (4) add a HeartShapedBox - object to the HeartShapedBoxList object, (5) delete a HeartShapedBox object from the HeartShapedBoxList object, (6) find a HeartShapedBox object in the HeartShapedBoxList object, (7) Edit a HeartShapedBox object in the HeartShapedBoxList 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 a HeartShapedBoxList 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 (see items 3 and 4 in the **Code and Test** section below).

Example 1 (next page) shows the output produced by running HeartShapedBoxListMenuApp and printing the action codes with short descriptions followed by the prompt with the abbreviated action codes waiting for the user to select from the menu.

Example 1

Program output

```
HeartShapedBox List System Menu
R - Read File and Create HeartShapedBox List
P - Print HeartShapedBox List
S - Print Summary
A - Add HeartShapedBox
D - Delete HeartShapedBox
F - Find HeartShapedBox
E - Edit HeartShapedBox
Q - Quit
Enter Code [R, P, S, A, D, F, E, or Q]:
```

Below shows the output after the user entered ‘r’ and then (when prompted) entered the file name. Notice the output from this action was “File read in and HeartShapedBox 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 *HeartShapedBox_data_1.txt* file from Project 5 to test your program.

Example 2

Program output

```
Enter Code [R, P, S, A, D, F, E, or Q]: r
   File name: HeartShapedBox_data_1.txt
   File read in and HeartShapedBox List created

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

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

Example 3

Program output

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

----- HeartShapedBox Test List -----

HeartShapedBox "Ex 1" with radius 4.0 units and height 1.5 units:
   heart shaped area = 114.265 square units
   lateral surface area = 61.699 square units
   total surface area = 290.23 square units
   volume = 171.398 cubic units

HeartShapedBox "Ex 2" with radius 10.0 units and height 2.4 units:
   heart shaped area = 714.159 square units
   lateral surface area = 246.796 square units
   total surface area = 1,675.115 square units
   volume = 1,713.982 cubic units

HeartShapedBox "Ex 3" with radius 15.5 units and height 4.5 units:
   heart shaped area = 1,715.768 square units
   lateral surface area = 717.252 square units
   total surface area = 4,148.787 square units
   volume = 7,720.954 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.

Example 4

Line # Program output

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

----- Summary for HeartShapedBox Test List -----
Number of HeartShapedBoxes: 3
Total Surface Area: 6,114.133 square units
Total Volume: 9,606.335 cubic units
Average Surface Area: 2,038.044 square units
Average Volume: 3,202.112 cubic units

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

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

Example 5

Line # Program output

```
Enter Code [R, P, S, A, D, F, E, or Q]: a
   Label: New HSB
   Radius: 19.5
   Height: 2
   *** HeartShapedBox added ***

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

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

Example 6

Line # Program output

```
Enter Code [R, P, S, A, D, F, E, or Q]: d
   Label: Ex 2
   "Ex 2" deleted

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

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

In the example below, there is a successful "find" for a HeartShapedBox object with label ex 3 (with case ignored), followed by an attempted "find" for Ex 99 that was not successful (i.e., the HeartShapedBox object with label Ex 99 was not found).

Example 7

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
2	`   Label: ex 3`
3	`HeartShapedBox "Ex 3" with radius 15.5 units and height 4.5 units:`
4	`   heart shaped area = 1,715.768 square units`
5	`   lateral surface area = 717.252 square units`
6	`   total surface area = 4,148.787 square units`
7	`   volume = 7,720.954 cubic units`
8	
9	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
10	`   Label: Ex 99`
11	`   "Ex 99" not found`
12	
13	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Below is an example of the successful "edit" for a HeartShapedBox object, followed by an attempt that was not successful. To verify the edit, you should do a "find" for "ex 1" or you could do a "print" to see the whole list. Note the actual label, Ex 1, is shown on line 5 rather than ex 1 which was entered by the user. **Edit does not modify the label**.

Example 8

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
2	`   Label: ex 1`
3	`   Radius: 4.55`
4	`   Height: 1.25`
5	`   "Ex 1" successfully edited`
6	
7	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
8	`   Label: Small Heart Box`
9	`   Radius: 4.0`
10	`   Height: 1.2`
11	`   "Small Heart Box" not found`
12	
13	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Entering a 'q' should quit the application with no message.

Example 9

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: q`
2	

Finally, if a code other than the ones in the "Enter Code" list is entered, the code is considered invalid and an appropriate message is printed, then the user is asked to enter a code again as shown below.

Example 10

Line # | Program output

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

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

Code and Test: This class should import java.util.Scanner, java.util.ArrayList, and java.io.FileNotFoundException. Carefully consider the following **important considerations** as you develop this class.

1. At the beginning of your main method, you should declare and create an ArrayList of HeartShapedBox objects and then declare and create a HeartShapedBoxList object using the list name and the ArrayList as the parameters in the constructor. This will be a HeartShapedBoxList object that contains no HeartShapedBox objects. For example:
```
String _______ = "*** no list name assigned ***";
ArrayList<HeartShapedBox> _________ = new  ArrayList<HeartShapedBox>();
HeartShapedBoxList _________ = new HeartShapedBoxList(_________,_________);
```
   The 'R' option in the menu should invoke the readFile method on your HeartShapedBoxList object. This will return a new HeartShapedBoxList object based on the data read from the file, and this new HeartShapedBoxList object should replace (be assigned to) your original HeartShapedBoxList 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. After you read in the code as a String, you should invoke the toUpperCase() method on the code and then convert the first character of the String to a char as shown in the two statements below.
```
      code = code.toUpperCase();
      char codeChar = code.charAt(0);
```

4. For the menu, your switch statement expression should evaluate to a char and each case should be a char.

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 HeartShapedBox List" option, you should be able to print the HeartShapedBoxList 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 HeartShapedBoxList 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 the methods in your HeartShapedBox and HeartShapedBoxList classes, you should ensure that all 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 HeartShapedBoxList 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.