Project: HeartShapedBox List App

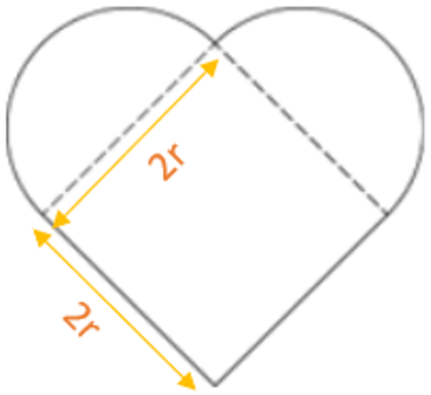
Deliverables

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
- HeartShapedBoxListApp.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: (1) HeartShapedBox, (2) HeartShapedBoxList, and (3) HeartShapedBoxListApp, which reads in a file name entered by the user then reads the list name and HeartShapedBox data from the file, creates HeartShapedBox objects and stores them in an ArrayList of HeartShapedBox objects, creates an HeartShapedBoxList object with the list name and ArrayList, prints the HeartShapedBoxList object, and then prints summary information about the HeartShapedBoxList object.

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

  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 these should be the only instance variables in the class.

  (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.
  - o `getLabel`: Accepts no parameters and returns a String representing the label field.
  - o `setLabel`: Takes a String parameter and returns a boolean. If the string parameter is not null, then the label field is set to the "trimmed" String and the method returns true. Otherwise, the method returns false, and the label field is not set.
  - o `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 a 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 will create below, except that in the HeartShapedBoxApp class you will read in the values and then create and print the object.

- **HeartShapedBoxList.java**

  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 not 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 lines 3 through 22 in the output below from HeartShapedBoxListApp for the *HeartShapedBox_data_1.txt* input file. [Note that the 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.]
- `summaryInfo`: Returns a String (does not 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 lines 24 through 30 in the output below 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.

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

- **HeartShapedBoxListApp.java**

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

- **Design**: The main method should prompt the user to enter a file name, and then it should read in the data file. The first record (or line) in the file contains the name of the list. This is followed by the data for the HeartShapedBox objects. Within a while loop, each set of HeartShapedBox data (i.e., label, radius, and height) is read in, and then a HeartShapedBox object should be created and added to the local ArrayList of HeartShapedBox objects. After the file has been read in and the ArrayList has been populated, the main method should create a HeartShapedBoxList object with the name of the list and the ArrayList of HeartShapedBox objects as parameters in the constructor. It should then print the HeartShapedBoxList object, and then print the summary information about the HeartShapedBoxList (i.e., print the value returned by the summaryInfo method for the HeartShapedBoxList). The output from two runs of the main method in HeartShapedBoxListApp is shown below. The first is produced after reading in the *HeartShapedBox_data_1.txt* file, and the second is produced after reading in the *HeartShapedBox_data_0.txt* file. Your program output should be formatted exactly as shown on the next page.

Example 1

Program output

```
Enter file name: HeartShapedBox_data_1.txt

----- 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


----- 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

```

Example 2

Program output

```
Enter file name: HeartShapedBox_data_0.txt

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


----- Summary for HeartShapedBox Empty Test List -----
Number of HeartShapedBoxes: 0
Total Surface Area: 0.0 square units
Total Volume: 0.0 cubic units
Average Surface Area: 0.0 square units
Average Volume: 0.0 cubic units

```

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

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

You can assume that the first line in the data file is the name of the list, and then each set of three lines contains the data from which an HeartShapedBox 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 HeartShapedBox 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 HeartShapedBox data items (label, radius, height) should be assigned, after which the HeartShapedBox object should be created and added to a local ArrayList of HeartShapedBox objects. The next iteration of the loop should then read the next set of three lines then create the next HeartShapedBox object and add it to the local ArrayList of HeartShapedBox objects, and so on. After the file has been processed (i.e., when the loop terminates after the hasNext method returns false), name of the list and the ArrayList of HeartShapedBox objects should be used to create an HeartShapedBoxList object. Then the list should be printed by printing a leading \n and the HeartShapedBoxList object. Finally, the summary information is printed by printing a leading \n and the value returned by the summaryInfo method invoked on the HeartShapedBoxList object.

Test: You should test your program minimally (1) by reading in the *HeartShapedBox_data_1.txt* input file, which should produce the first output above, and (2) by reading in the *HeartShapedBox_data_0.txt* input file, which should produce the second output above. Although your program may not use all the methods in the HeartShapedBoxList and HeartShapedBox classes, you should ensure that all your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.

General Notes

1. All input from the keyboard and all output to the screen should done in the main method. Only one Scanner object on System.in should be created and this should be done in the main method. All printing (i.e., using the System.out.print and/or System.out.println methods) should be in the main method. Hence, none of your methods in the HeartShapedBox and HeartShapedBoxList classes should do any input/output (I/O).
2. Be sure to download the test data files (*HeartShapedBox_data_1.txt* and *HeartShapedBox_data_0.txt*) and store them in same folder as your source files. It may be useful examine the contents of the data files. Find the data files in the jGRASP Browse tab and then open each data file in jGRASP to see the items that your program will be reading from the file. Be sure to close the data files without changing them.