

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. 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 (both files must be submitted together):

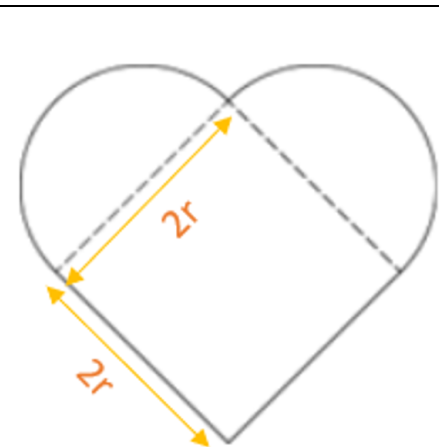
- HeartShapedBox.java
- HeartShapedBoxApp.java

Specifications

Overview: You will write a program this week that is composed of two classes:

(1) HeartShapedBox, which defines HeartShapedBox objects (a box with a heart shaped top and bottom), and (2) HeartShapedBoxApp, which has a main method that reads in data, creates an HeartShapedBox object, and then prints the object.

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.

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

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

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

$$V = h A_{Heart}$$

• HeartShapedBox .java

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.
 - **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 “trimmed” 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 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.

- **HeartShapedBoxApp.java**

Requirements: Create a `HeartShapedBoxApp` class with a main method that reads in values for label, radius, and height. After the values have been read in, main creates a `HeartShapedBox` object and then prints a new line and the object.

Design: The main method should prompt the user to enter the label, radius, and height. After a value is read in for the radius or height, if the value is less than or equal to zero, an appropriate message (see examples below) should be printed followed by a *return* statement. Assuming that the radius and height are positive, a `HeartShapedBox` object should be created and printed. Below is an example where the user has entered a negative value for radius followed by an

example using the values from the first example above for label, radius, and height. Your program input/output should be **exactly** as follows.

Example #0a

Line #	Program input/output
1	Enter label, radius, and height for a heart shaped box.
2	label: Ex 0a
3	radius: -1.2
4	Error: radius must be greater than zero.
5	

Example #0b

Line #	Program input/output
1	Enter label, radius, and height for a heart shaped box.
2	label: Ex 0b
3	radius: 4.0
4	height: 0
5	Error: height must be greater than zero.
6	

Example #1

Line #	Program input/output
1	Enter label, radius, and height for a heart shaped box.
2	label: Ex 1
3	radius: 4.0
4	height: 1.5
5	
6	HeartShapedBox "Ex 1" with radius 4.0 units and height 1.5 units:
7	heart shaped area = 114.265 square units
8	lateral surface area = 61.699 square units
9	total surface area = 290.23 square units
10	volume = 171.398 cubic units
11	

Example #2

Line #	Program input/output
1	Enter label, radius, and height for a heart shaped box.
2	label: Ex 2
3	radius: 10.0
4	height: 2.4
5	
6	HeartShapedBox "Ex 2" with radius 10.0 units and height 2.4 units:
7	heart shaped area = 714.159 square units
8	lateral surface area = 246.796 square units
9	total surface area = 1,675.115 square units
10	volume = 1,713.982 cubic units
11	

Example #3

Line #	Program input/output
1	Enter label, radius, and height for a heart shaped box.
2	label: Ex 3
3	radius: 15.5
4	height: 4.5
5	
6	HeartShapedBox "Ex 3" with radius 15.5 units and height 4.5 units:
7	heart shaped area = 1,715.768 square units
8	lateral surface area = 717.252 square units
9	total surface area = 4,148.787 square units
10	volume = 7,720.954 cubic units
11	

Code: Your program should use the `nextLine` method of the `Scanner` class to read user input. Note that this method returns the input as a `String`, even when it appears to be numeric value. Whenever necessary, you can use the `Double.parseDouble` method to convert the input `String` to a `double`. For example, `Double.parseDouble(s1)` will return the double value represented by `String s1`, assuming `s1` represents a numeric value. For the printed lines requesting input for label, radius, and height, use a tab `"\t"` rather than three spaces. After reading in the values, create the new `HeartShapedBox`, say `hsb`, then print it: `System.out.println("\n" + hsb);`

Test: You should test several sets of data to make sure that your program is working correctly. Although your main method may not use all the methods in `HeartShapedBox`, you should ensure that all your methods work according to the specification. You can use interactions in `jGRASP` or you can write another class and main method to exercise the methods. The viewer canvas should also be helpful, especially using the “Basic” viewer and the “toString” viewer for a `HeartShapedBox` object. Web-CAT will test all the methods specified above for `HeartShapedBox` to determine your project grade.

General Notes

1. All input from the keyboard and all output to the screen should be done in the main method. Only one `Scanner` object on `System.in` should be created and this should be done in the main method. All printing (i.e., using the `System.out.print` and `System.out.println` methods) should be in the main method. Hence, none of your methods in the `HeartShapedBox` class should do any input/output (I/O).
2. When a method has a return value, you can ignore the return value if it is not of interest in the current context. For example, when `setRadius(3.5)` is invoked, it returns `true` to let the caller know the radius field was set; whereas `setRadius(-3.5)` will return `false` since the radius field was not set. So, if the caller knows that `x` is positive, then the return value of `setRadius(x)` can safely be ignored since it can be assumed to be `true`.
3. Even though you may not be using the return value of a method, you can ensure that the return value is correct using interactions in `jGRASP`.
4. In a method with a void return type (e.g., `main`), the return statement (`return;`) can be used immediately return from the method. This is commonly used to exit the method when some condition is not met. For example, suppose a value for time is read in by the main method and should be greater

than zero. If time is less than or equal to zero, an error message should be printed, and the program should immediately end by returning from main as shown in the code below.

```
if (time <= 0) {  
    System.out.println("Error: time must be non-negative.");  
    return;  
}
```