Project: HexagonalPrism 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. 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.

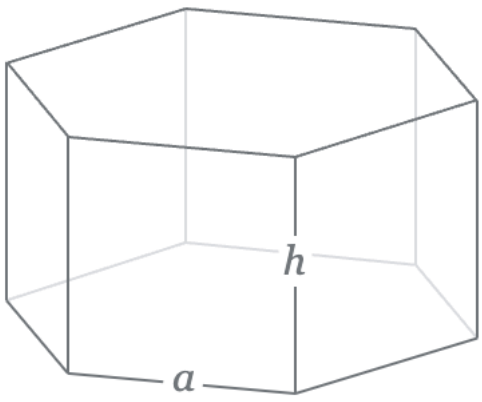
Files to submit to Web-CAT (both files must be submitted together):

- HexagonalPrism.java
- HexagonalPrismApp.java

Specifications

Overview: You will write a program this week that is composed of two classes: (1) HexagonalPrism, which defines right regular HexagonalPrism objects (base and top of prism are hexagons with all sides equal), and (2) HexagonalPrismApp, which has a main method that reads in data, creates an HexagonalPrism object, and then prints the object.

The **Hexagonal Prism** is a prism with hexagonal base. This polyhedron has 8 faces, 18 edges, and 12 vertices. The formulas are provided to assist you in computing return values for the respective methods in the HexagonalPrism class described in this project.
(Sources: "Google" Hexagonal Prism; https://en.wikipedia.org/wiki/Hexagonal_prism)



Formulas for surface area (***A***), lateral surface area ($\boldsymbol{A_L}$), base area ($\boldsymbol{A_B}$), and volume (***V***) are shown below where ***a*** is the *base edge length* and ***h*** is *height*, both of which will be read in.

$$A = 6ah + 3\sqrt{3}\,a^2$$

$$A_L = 6ah$$

$$A_B = 3\sqrt{3}\,\frac{a^2}{2}$$

$$V = \frac{3\sqrt{3}}{2}\,a^2h$$

- **HexagonalPrism.java**

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

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

  (1) **Fields** (instance variables): label of type String, edge 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 HexagonalPrism class, and these should be the only instance variables in the class.

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

```
HexagonalPrism ex1 = new HexagonalPrism ("Ex 1", 3.0, 5.0);

HexagonalPrism ex2 = new HexagonalPrism (" Ex 2   ", 21.3, 10.4);

HexagonalPrism ex3 = new HexagonalPrism ("Ex 3", 10.0, 204.5);
```

  (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 HexagonalPrism, 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.
  - `getEdge`: Accepts no parameters and returns a double representing the edge field.
  - `setEdge`: Accepts a double parameter and returns a boolean as follows. If the parameter is greater than zero, sets the edge field to the double passed in and returns true. Otherwise, the method returns false, and the edge 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.
  - `surfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.

- `lateralSurfaceArea`: Accepts no parameters and returns the double value for the surface area calculated using the formula above.
- `baseArea`: Accepts no parameters and returns the double value for the 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 HexagonalPrism 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: lateralSurfaceArea(), baseArea(), 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 results of printing the toString value of ex1, ex2, and ex3 respectively are shown below.

```
HexagonalPrism "Ex 1" has 8 faces, 18 edges, and 12 vertices.
   edge = 3.0 units
   height = 5.0 units
   lateral surface area = 90.0 square units
   base area = 23.383 square units
   surface area = 136.765 square units
   volume = 116.913 cubic units

HexagonalPrism "Ex 2" has 8 faces, 18 edges, and 12 vertices.
   edge = 21.3 units
   height = 10.4 units
   lateral surface area = 1,329.12 square units
   base area = 1,178.721 square units
   surface area = 3,686.562 square units
   volume = 12,258.7 cubic units

HexagonalPrism "Ex 3" has 8 faces, 18 edges, and 12 vertices.
   edge = 10.0 units
   height = 204.5 units
   lateral surface area = 12,270.0 square units
   base area = 259.808 square units
   surface area = 12,789.615 square units
   volume = 53,130.659 cubic units
```

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

- **HexagonalPrismApp.java**

  Requirements: Create an HexagonalPrismApp class with a main method that reads in values for label, edge, and height. After the values have been read in, main creates an HexagonalPrism object and then prints a new line and the object.

  Design: The main method should prompt the user to enter the label, edge, and height. After a value is read in for the edge 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 edge and height are positive, an HexagonalPrism object should be created and printed. Below is an example where the user has entered a negative value for edge followed by an example using the values from the first example above for label, edge, and height. Your program input/output should be **exactly** as follows.

Example #0A

Line #	Program input/output
1	`Enter label, edge, and height for a hexagonal prism.`
2	`   label: Ex 0A`
3	`   edge: -4.0`
4	`Error: edge must be greater than zero.`
5	

Example #0B

Line #	Program input/output
1	`Enter label, edge, and height for a hexagonal prism.`
2	`   label: Ex 0B`
3	`   edge: 4.0`
4	`   height: 0.0`
5	`Error: height must be greater than zero.`
6	

Example #1

Line #	Program input/output
1	`Enter label, edge, and height for a hexagonal prism.`
2	`   label: Ex 1`
3	`   edge: 3.0`
4	`   height: 5.0`
5	
6	`HexagonalPrism "Ex 1" has 8 faces, 18 edges, and 12 vertices.`
7	`   edge = 3.0 units`
8	`   height = 5.0 units`
9	`   lateral surface area = 90.0 square units`
10	`   base area = 23.383 square units`
11	`   surface area = 136.765 square units`
12	`   volume = 116.913 cubic units`
13	

Example #2

Line #	Program input/output
1	`Enter label, edge, and height for a hexagonal prism.`
2	`   label:  Ex 2`
3	`   edge: 21.3`
4	`   height: 10.4`
5	
6	`HexagonalPrism "Ex 2" has 8 faces, 18 edges, and 12 vertices.`
7	`   edge = 21.3 units`
8	`   height = 10.4 units`
9	`   lateral surface area = 1,329.12 square units`
10	`   base area = 1,178.721 square units`
11	`   surface area = 3,686.562 square units`
12	`   volume = 12,258.7 cubic units`
13	

Example #3

Line #	Program input/output
1	`Enter label, edge, and height for a hexagonal prism.`
2	`   label: Ex 3`
3	`   edge: 10.0`
4	`   height: 204.5`
5	
6	`HexagonalPrism "Ex 3" has 8 faces, 18 edges, and 12 vertices.`
7	`   edge = 10.0 units`
8	`   height = 204.5 units`
9	`   lateral surface area = 12,270.0 square units`
10	`   base area = 259.808 square units`
11	`   surface area = 12,789.615 square units`
12	`   volume = 53,130.659 cubic units`
13	

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, edge, and height, use a tab "\t" rather than three spaces. After reading in the values, create the new HexagonalPrism, say `hp`, then print it: `System.out.println("\n" + hp);`

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 HexagonalPrism, 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 an HexagonalPrism object. Web-CAT will test all the methods specified above for HexagonalPrism 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 System.out.println methods) should be in the main method. Hence, none of your methods in the HexagonalPrism 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 no interest in the current context. For example, when setEdge(3.5) is invoked, it returns true to let the caller know the edge field was set; whereas setEdge(-3.5) will return false since the edge field was not set. So, if the caller knows that x is positive, then the return value of setEdge(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 main and should be greater than or equal to zero. If time is less than 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;
```