

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.

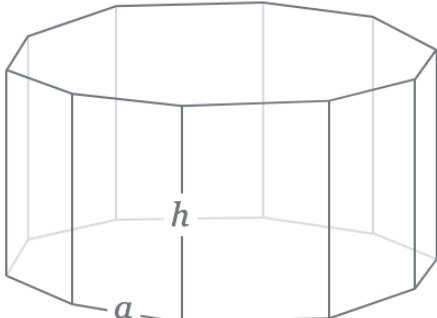
Files to submit to Web-CAT (both files must be submitted together):

- DecagonalPrism.java
- DecagonalPrismApp.java

Specifications

Overview: You will write a program this week that is composed of two classes: (1) DecagonalPrism that defines object for 3-D geometric shape objects with ten square side faces and two regular decagon caps) where edge and height are non-negative, and (2) DecagonalPrismApp, which has a main method that reads in data, creates a DecagonalPrism object, and then prints the object. A_B

A **decagonal prism** is a 3-D geometric shape formed by ten square side faces and two regular decagon caps. The decagonal prism has 12 faces, 30 edges, and 20 vertices. A decagonal prism can be defined by its base edge (a) and height (h) as depicted below. The formulas are provided to assist you in computing return values for the respective methods in the DecagonalPrism class described in this project.

	<p>Nomenclature</p> <p>Base edge: a Height: h</p> <p>Surface area: A Base area: A_B Lateral surface area: A_L Volume: V</p>	<p>Formulas</p> $A = 10ah + 5a^2\sqrt{5+2\sqrt{5}}$ $A_B = \frac{5}{2}a^2\sqrt{5+2\sqrt{5}}$ $A_L = 10ah$ $V = \frac{5}{2}a^2\sqrt{5+2\sqrt{5}}h$
-------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------

- DecagonalPrism.java

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

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

- (1) **Fields** (instance variables): label of type String, base edge of type double, and height of type double. These instance variables should be private so that they are not directly accessible from outside of the DecagonalPrism class, and these should be the only instance variables in the class.
- (2) **Constructor:** Your DecagonalPrism class must contain a 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 DecagonalPrism 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.

```
DecagonalPrism ex1 = new DecagonalPrism("Small Example", 2.0, 5.0);
```

```
DecagonalPrism ex2 = new DecagonalPrism(" Medium Example ", 10.4, 32.6);
```

```
DecagonalPrism ex3 = new DecagonalPrism("Large Example", 50, 100);
```

- (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 DecagonalPrism are described below. See the formulas above and “Code and Test” below for details. When implementing the formulas, be sure to use the methods `Math.pow` and `Math.sqrt` as appropriate.
 - `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 trimmed String is set to the label field 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. If the edge is non-negative, sets edge field 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. If the height is non-negative, sets height field 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 edge and height.
 - `baseArea`: Accepts no parameters and returns the double value for the base area calculated using the edge.
 - `lateralSurfaceArea`: Accepts no parameters and returns the double value for the slant height calculated using the edge and height.

- volume: Accepts no parameters and returns the double value for the volume calculated using edge and height.
- toString: Returns a String containing the information about the DecagonalPrism 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(), baseArea(), lateralSurfaceArea(), and volume(). Each line should have no leading and no trailing spaces (e.g., there should be no spaces before a newline (\n) other than tab (\t) as appropriate). The toString value for ex1, ex2, and ex3 respectively are shown below (the blank lines are not part of the toString values; that is, the return value should not begin with a newline (\n) character and should not end with a newline (\n) character).

```
A decagonal prism "ex1" with edge = 2.0 units and height = 5.0 units, has:  
  surface area = 161.554 units  
  base area = 30.777 square units  
  lateral surface area = 100.0 square units  
  volume = 153.884 cubic units
```

```
A decagonal prism "ex2" with edge = 10.4 units and height = 32.6 units, has:  
  surface area = 5,054.811 units  
  base area = 832.206 square units  
  lateral surface area = 3,390.4 square units  
  volume = 27,129.903 cubic units
```

```
A decagonal prism "ex3" with edge = 50.0 units and height = 100.0 units, has:  
  surface area = 88,471.044 units  
  base area = 19,235.522 square units  
  lateral surface area = 50,000.0 square units  
  volume = 1,923,552.211 cubic units
```

Code and Test: As you implement your DecagonalPrism 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 DecagonalPrism in interactions (see 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 a DecagonalPrism 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 DecagonalPrism then prints it out. This would be similar to the class you will create in DecagonalPrismApp, except that in DecagonalPrismApp you will read in the values and then create the object.

- **DecagonalPrismApp.java**

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

Design: The main method should prompt the user to enter the label, edge, and height. After a value is read in for edge, if the value is less than zero, an appropriate message (see examples below) should be printed followed by a *return* from main. Do the same after a value for height is read in. Assuming that both edge and height are non-negative, a DecagonalPrism object should be created and printed. Below are examples where the user has entered negative values for edge and height 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.

Line #	Program input/output
1	Enter label, edge, and height for a decagonal prism.
2	label: Bad edge value
3	edge: -2
4	Error: edge must be non-negative.
5	

Line #	Program input/output
1	Enter label, edge, and height for a decagonal prism.
2	label: Bad height value
3	edge: 0
4	height: -5.8
5	Error: height must be non-negative.
6	

Line #	Program input/output
1	Enter label, edge, and height for a decagonal prism.
2	label: ex1
3	edge: 2.0
4	height: 5.0
5	A decagonal prism "ex1" with edge = 2.0 units and height = 5.0 units, has:
6	surface area = 161.554 units
7	base area = 30.777 square units
8	lateral surface area = 100.0 square units
9	volume = 153.884 cubic units

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`. If you need to convert the `String` to a `double`, 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`. For the printed lines requesting input for label, edge, and height, use a `"\t"` rather than three spaces.

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 `DecagonalPrism`, 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

DecagonalPrism object. Web-CAT will test all of the methods specified above for the DecagonalPrism class 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 DecagonalPrism 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 setHeight(3.5) is invoked, it returns true to let the caller know the radius field was set; whereas setHeight(-3.5) will return false since the radius field is not set. If the caller knows that x is positive, then the return value of setHeight(x) can safely be ignored since it can be assumed to be true.
3. Even though your main method may not be using the return type of a method, you can ensure that the return type is correct using interactions.