# Project 4: Hexagonal Prism App



**Due**: Skeleton Code (ungraded - checks class, method names, etc.)
Completed Code – Thursday, February 18, 2016

## Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until the project due date but should try to do this by Friday, February 12 (there is no late penalty since the skeleton code assignment is ungraded for this project). For an example of skeleton code, see loan.java in the Class Notes for this week (04_Writing_Classes\examples\method_stubs\loan.java). 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. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor 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, one that defines right regular HexagonalPrism objects (base and top of prism are hexagons with all sides equal) and the other, HexagonalPrismApp, which has a main method that reads in data, creates a HexagonalPrism object, and then prints the object.

- **HexagonalPrism.java**

  **Requirements**: Create a HexagonalPrism class that stores the label, side, and height. The HexagonalPrism class also includes methods to set and get each of these fields, as well as methods to calculate the base perimeter, base area, surface 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, side 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 HexagonalPrism class, and these should be the only instance variables in the class.

  (2) **Constructor**: Your HexagonalPrism class must contain a constructor that accepts three parameters (see types of above) representing the label, side, and height. The value for label should be trimmed of leading and trailing spaces prior to setting the field (hint: use trim method from the String class). 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 example1 = new HexagonalPrism("Short Example", 4.0, 6.0);
HexagonalPrism example2 = new HexagonalPrism(" Wide Example ", 22.1, 10.6);
HexagonalPrism example3 = new HexagonalPrism("Tall Example", 10, 200);
```

(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 are described 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 null, then the method returns false and the label is not set. Otherwise, the "trimmed" String is set to the label field and the method returns true.
- `getSide`: Accepts no parameters and returns a double representing the side field.
- `setSide`: Accepts a double parameter, sets side field, and returns nothing.
- `getHeight`: Accepts no parameters and returns a double representing the height field.
- `setHeight`: Accepts a double parameter, sets height field, and returns nothing.
- `basePerimeter`: Accepts no parameters and returns the double value for the base perimeter of the top (or bottom) hexagon calculated using side.
- `baseArea`: Accepts no parameters and returns the double value for the base area calculated using side.
- `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using side and height.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using side, and height.
- `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: basePerimeter(), 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 example1, example2, and example3 respectively are shown below.

```
"Short Example" is a hexagonal prism with side = 4.0 units and height = 6.0 units,
which has base perimeter = 24.0 units, base area = 41.569 square units,
surface area = 227.138 square units, and volume = 249.415 cubic units.

"Wide Example" is a hexagonal prism with side = 22.1 units and height = 10.6 units,
which has base perimeter = 132.6 units, base area = 1,268.926 square units,
surface area = 3,943.413 square units, and volume = 13,450.62 cubic units.

"Tall Example" is a hexagonal prism with side = 10.0 units and height = 200.0 units,
which has base perimeter = 60.0 units, base area = 259.808 square units,
surface area = 12,519.615 square units, and volume = 51,961.524 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 (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 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 is out. This would be similar to the class you will create in Part 2, except that in in Part 2 you will read in the values and then create the object.

- **HexagonalPrismApp.java**

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

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

| Line # | Program input/output |
|---|---|
| 1 | `Enter label, side, and height for a hexagonal prism.` |
| 2 | `        label: Short Example` |
| 3 | `        side: 4.0` |
| 4 | `        height: 6.0` |
| 5 | `"Short Example" is a hexagonal prism with side = 4.0 units and height = 6.0 units,` |
| 6 | `which has base perimeter = 24.0 units, base area = 41.569 square units,` |
| 7 | `surface area = 227.138 square units, and volume = 249.415 cubic units.` |
| 8 | |

**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.

**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 of 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 HexagonalPrism object. Web-CAT will test all of the methods specified above for HexagonalPrism to determine your project grade.