Project: Dodecahedron 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 by Friday (there is no late penalty since the skeleton code assignment is ungraded for this project). 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. 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 TA before the deadline.
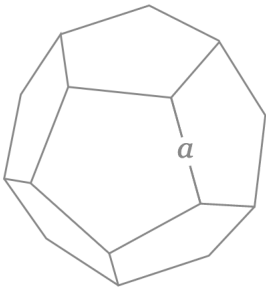
Files to submit to Web-CAT (both files must be submitted together):

- Dodecahedron.java
- DodecahedronApp.java

Specifications

Overview: You will write a program this week that is composed of two classes: (1) one named Dodecahedron that defines Dodecahedron objects, and (2) the other, DodecahedronApp, which has a main method that reads in data, creates a Dodecahedron object, and then prints the object.

A dodecahedron has 12 equal pentagonal faces, 20 edges, and 30 vertices as depicted below. The formulas are provided to assist you in computing return values for the respective Dodecahedron methods described in this project.

	Surface Area (A) Volume (V) Edge length (a) Surface/Volume ratio (A/V)	$A = 3\sqrt{25+10\sqrt{5}}\,a^2$ $V = \dfrac{15+7\sqrt{5}}{4}a^3$

- **Dodecahedron.java**

  Requirements: Create a Dodecahedron class that stores the label, color, and edge (i.e., length of an edge, which must be greater than zero). The Dodecahedron class also includes methods to set and get each of these fields, as well as methods to calculate the surface area, volume, and surface to volume ratio of a Dodecahedron object, and a method to provide a String value of a Dodecahedron object (i.e., a class instance).

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

(1) **Fields** (instance variables): label of type String, color of type String, and edge of type double. Initialize the Strings to `""` and the double to 0 in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the Dodecahedron class, and <u>these should be the only instance variables in the class</u>.

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

```
Dodecahedron example1 = new Dodecahedron("Small Example", "blue", 0.25);

Dodecahedron example2 = new Dodecahedron(" Medium Example ", "orange", 10.1);

Dodecahedron example3 = new Dodecahedron("Large Example", "silver  ", 200.5);
```

(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 Dodecahedron 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 "trimmed" String is set to the label field and the method returns true. Otherwise, the method returns false and the label is not set.

   o `getColor`: Accepts no parameters and returns a String representing the color field.

   o `setColor`: Takes a String parameter and returns a boolean. If the string parameter is not null, then the "trimmed" String is set to the color field and the method returns true. Otherwise, the method returns false and the label is not set.

   o `getEdge`: Accepts no parameters and returns a double representing the edge field.

   o `setEdge`: Accepts a double parameter and returns a boolean as follows. If the edge is greater than zero, sets the edge field to the double passed in and returns true. Otherwise, the method returns false and the edge is not set.

   o `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using the value for edge.

   o `volume`: Accepts no parameters and returns the double value for the volume calculated using the value for edge.

   o `surfaceToVolumeRatio`: Accepts no parameters and returns the double value calculated by dividing the total surface area by the volume.

   o `toString`: Returns a String containing the information about the Dodecahedron 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(), volume(), and surfaceToVolumeRatio(). Each line should have no trailing spaces (e.g., there should be no spaces before a newline (\n) character). The toString value for example1, example2, and example3 respectively are shown below (the blank lines are not part of the toString values).

```
Dodecahedron "Small Example" is "blue" with 30 edges of length 0.25 units.
   surface area = 1.29 square units
   volume = 0.12 cubic units
   surface/volume ratio = 10.777

Dodecahedron "Medium Example" is "orange" with 30 edges of length 10.1 units.
   surface area = 2,106.071 square units
   volume = 7,895.319 cubic units
   surface/volume ratio = 0.267

Dodecahedron "Large Example" is "silver" with 30 edges of length 200.5 units.
   surface area = 829,963.459 square units
   volume = 61,765,889.248 cubic units
   surface/volume ratio = 0.013
```

Code and Test: As you implement your Dodecahedron 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 Dodecahedron in interactions (e.g., copy/paste 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 Dodecahedron 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 Dodecahedron then prints it out. This would be similar to the class you will create in Part 2, except that in Part 2 you will read in the values and then create the object.

- **DodecahedronApp.java**

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

   Design: The main method should prompt the user to enter the label, color, and edge. After a value is read in for edge, if the value is less than or equal to zero, an appropriate message (see examples below) should be printed followed by a *return* from main. Assuming that edge is positive, a Dodecahedron object should be created and printed. Below is an example where the user has entered a non-positive value for edge followed by an example using the values from the first example above for label, color, and edge. Your program input/output should be **exactly** as follows.

Line #	Program input/output
1	Enter label, color, and edge length for a dodecahedron.
2	label: Example with bad edge value
3	color: purple
4	edge: 0
5	Error: edge must be greater than 0.

Line #	Program input/output
1	Enter label, color, and edge length for a dodecahedron.
2	label: Small Example
3	color: blue
4	edge: 0.25
5	
6	Dodecahedron "Small Example" is "blue" with 30 edges of length 0.25 units.
7	surface area = 1.29 square units
8	volume = 0.12 cubic units
9	surface/volume ratio = 10.777

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. 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. For the printed lines requesting input for label, color, and edge, use a tab "\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 Dodecahedron, 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 Dodecahedron object. Web-CAT will test all of the methods specified above for Dodecahedron 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 Dodecahedron 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 your main method may not be using the return type of a method, you can ensure that the return type is correct using interactions.