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.

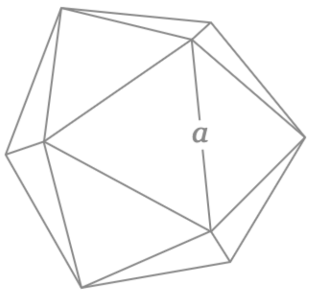
Files to submit to Web-CAT (all three files must be submitted together):

- Icosahedron.java
- IcosahedronList.java
- IcosahedronListApp.java

Specifications

Overview: You will write a program this week that is composed of three classes: the first class defines Icosahedron objects, the second class defines IcosahedronList objects, and the third, IcosahedronListApp, reads in a file name entered by the user then reads the list name and Icosahedron data from the file, creates Icosahedron objects and stores them in an ArrayList, creates a IcosahedronList object with the list name and ArrayList, prints the IcosahedronList object, and then prints summary information about the IcosahedronList object.

An **Icosahedron** has 20 equilateral triangle faces, 12 vertices, and 30 edges as depicted below. The formulas are provided to assist you in computing return values for the respective methods in the Icosahedron class described in this project.

	Surface Area (A) Volume (V) Edge length (a) Surface/Volume ratio (A/V)	$A = 5\sqrt{3}\,a^2$ $V = \frac{5(3+\sqrt{5})}{12}a^3$

- **Icosahedron.java (<u>assuming that you successfully created this class in Project 4, just copy the file to your new Project 5 folder and go on to IcosahedronList.java on page 4. Otherwise, you will need to create Icosahedron.java as part of this project</u>.)**

  Requirements: Create an Icosahedron class that stores the label, color, and edge (i.e., length of an edge, <u>which must be greater than zero</u>). The Icosahedron 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 an Icosahedron object, and a method to provide a String value of an Icosahedron object (i.e., a class instance).

  Design: The Icosahedron 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 Icosahedron class, and <u>these should be the only instance variables in the class</u>.

  (2) **Constructor**: Your Icosahedron class must contain a public 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 Icosahedron 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.

```
Icosahedron example1 = new Icosahedron("Small", "blue", 0.01);

Icosahedron example2 = new Icosahedron("    Medium    ", "orange", 12.3);

Icosahedron example3 = new Icosahedron("Large", "  white  ", 123.4);
```

  (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 Icosahedron, 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 "trimmed" String is set to the label field and the method returns true. Otherwise, the method returns false and the label is not set.
  - `getColor`: Accepts no parameters and returns a String representing the color field.
  - `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.
  - `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 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.
- `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using the value for edge.
- `volume`: Accepts no parameters and returns the double value for the volume calculated using the value for edge.
- `surfaceToVolumeRatio`: Accepts no parameters and returns the double value calculated by dividing the total surface area by the volume.
- `toString`: Returns a String containing the information about the Icosahedron 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).

```
Icosahedron "Small" is "blue" with 30 edges of length 0.01 units.
   surface area = 0.000866 square units
   volume = 0.000002 cubic units
   surface/volume ratio = 396.950723

Icosahedron "Medium" is "orange" with 30 edges of length 12.3 units.
   surface area = 1,310.209833 square units
   volume = 4,059.844212 cubic units
   surface/volume ratio = 0.322724

Icosahedron "Large" is "white" with 30 edges of length 123.4 units.
   surface area = 131,874.537977 square units
   volume = 4,099,581.395236 cubic units
   surface/volume ratio = 0.032168
```

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

- **IcosahedronList.java**

  Requirements: Create an IcosahedronList class that stores the name of the list and an ArrayList of Icosahedron objects. It also includes methods that return the name of the list, number of Icosahedron objects in the IcosahedronList, total surface area, total volume, average surface area, average volume, and average surface to volume ratio for all Icosahedron objects in the IcosahedronList. The toString method returns a String containing the name of the list followed by each Icosahedron in the ArrayList, and a summaryInfo method returns summary information about the list (see below).

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

  (1) Fields (or instance variables): (1) a String representing the name of the list and (2) an ArrayList of Icosahedron objects. These are the only fields (or instance variables) that this class should have, and both should be private.

  (2) Constructor: Your IcosahedronList class must contain a constructor that accepts a parameter of type String representing the name of the list and a parameter of type ArrayList< Icosahedron> representing the list of Icosahedron objects. These parameters should be used to assign the fields described above (i.e., the instance variables).

  (3) Methods: The methods for IcosahedronList are described below.
  - `getName`: Returns a String representing the name of the list.
  - `numberOfIcosahedrons`: Returns an int representing the number of Icosahedron objects in the IcosahedronList. If there are zero Icosahedron objects in the list, zero should be returned.
  - `totalSurfaceArea`: Returns a double representing the total surface areas for all Icosahedron objects in the list. If there are zero Icosahedron objects in the list, zero should be returned.
  - `totalVolume`: Returns a double representing the total volumes for all Icosahedron objects in the list. If there are zero Icosahedron objects in the list, zero should be returned.
  - `averageSurfaceArea`: Returns a double representing the average surface area for all Icosahedron objects in the list. If there are zero Icosahedron objects in the list, zero should be returned.
  - `averageVolume`: Returns a double representing the average volume for all Icosahedron objects in the list. If there are zero Icosahedron objects in the list, zero should be returned.
  - `averageSurfaceToVolumeRatio`: Returns a double representing the average surface to volume ratio for all Icosahedron objects in the list. If there are zero Icosahedron objects in the list, zero should be returned.
  - `toString`: Returns a String (does <u>not</u> begin with \n) containing the name of the list followed by each Icosahedron in the ArrayList. In the process of creating the return result, this toString() method should include a while loop that calls the toString() method for each Icosahedron object in the list (adding a \n before and after each). Be sure to

include appropriate newline escape sequences. For an example, see lines 2 through 19 in the output below from IcosahedronListApp for the *icosahedron_data_1.txt* input file. [Note that the toString result should **not** include the summary items in lines 20 through 26 of the example. These lines represent the return value of the summaryInfo method below.]

- `summaryInfo`: Returns a String (does not begin with \n) containing the name of the list (which can change depending of the value read from the file) followed by various summary items: number of Icosahedrons, total surface area, total volume, average surface area, average volume, and average surface to volume ratio. Use "#,##0.0##" as the pattern to format the double values. For an example, see lines 20 through 26 in the output below from IcosahedronListApp for the *icosahedron_data_1.txt* input file. The second example below shows the output from IcosahedronListApp for the *icosahedron_data_0.txt* input file which contains a list name but no Icosahedron data.

Code and Test: Remember to import java.util.ArrayList. Each of the methods above requires that you use a loop (i.e., a while loop) to retrieve each object in the ArrayList. As you implement your IcosahedronList class, you can compile it and then test it using interactions. Alternatively, you can create a class with a simple main method that creates an IcosahedronList object and calls its methods.

- **IcosahedronListApp.java**

  Requirements: Create an IcosahedronListApp class with a main method that reads in the name of the data file entered by the user and then reads list name and Icosahedron data from the file, creates Icosahedron objects, stores them in a local ArrayList of Icosahedron objects, creates a IcosahedronList object with the name of the list and the ArrayList of Icosahedron objects, and then prints the IcosahedronList object followed summary information about the IcosahedronList object. **All input and output for this project should be done in the main method**.

- **Design**: The main method should prompt the user to enter a file name, and then it should read in the data file. The first record (or line) in the file contains the name of the list. This is followed by the data for the Icosahedron objects. After each set of Icosahedron data is read in, a Icosahedron object should be created and stored in the local ArrayList of Icosahedron objects. After the file has been read in and the ArrayList has been populated, the main method should create a IcosahedronList object with the name of the list and the ArrayList of Icosahedron objects as parameters in the constructor. It should then print the IcosahedronList object, then print the summary information about the IcosahedronList (i.e., print the value returned by the summaryInfo method for the IcosahedronList). The output from two runs of the main method in IcosahedronListApp is shown below. The first is produced after reading in the *icosahedron_data_1.txt* file, and the second is produced after reading in the *icosahedron_data_0.txt* file. Your program output should be formatted exactly as shown.

Line # Program output

```
----jGRASP exec: java IcosahedronListApp
Enter file name: icosahedron_data_1.txt

Icosahedron Test List

Icosahedron "Small" is "blue" with 30 edges of length 0.01 units.
   surface area = 0.000866 square units
   volume = 0.000002 cubic units
   surface/volume ratio = 396.950723

Icosahedron "Medium" is "orange" with 30 edges of length 12.3 units.
   surface area = 1,310.209833 square units
   volume = 4,059.844212 cubic units
   surface/volume ratio = 0.322724

Icosahedron "Large" is "white" with 30 edges of length 123.4 units.
   surface area = 131,874.537977 square units
   volume = 4,099,581.395236 cubic units
   surface/volume ratio = 0.032168


----- Summary for Icosahedron Test List -----
Number of Icosahedrons: 3
Total Surface Area: 133,184.749
Total Volume: 4,103,641.239
Average Surface Area: 44,394.916
Average Volume: 1,367,880.413
Average Surface/Volume Ratio: 132.435

 ----jGRASP: operation complete.
```

Line # Program output

```
----jGRASP exec: java IcosahedronListApp
Enter file name: icosahedron_data_0.txt

Icosahedron Empty Test List


----- Summary for Icosahedron Empty Test List -----
Number of Icosahedrons: 0
Total Surface Area: 0.0
Total Volume: 0.0
Average Surface Area: 0.0
Average Volume: 0.0
Average Surface/Volume Ratio: 0.0

 ----jGRASP: operation complete.
```

Code: Remember to import java.util.ArrayList, java.util.Scanner, and java.io.File, and java.io.FileNotFoundException prior to the class declaration. Your main method declaration should indicate that main `throws FileNotFoundException`. After your program reads in the file name from the keyboard, it should read in the data file using a Scanner object that was created on a file using the file name entered by the user.

```
... = new Scanner(new File(fileName));
```

You can assume that the first line in the data file is the name of the list, and then each set of three lines contains the data from which a Icosahedron object can be created. After the name of the list has been read and assigned to a local variable, a while loop should be used to read in the Icosahedron data. The boolean expression for the while loop should be `(__________.hasNext())` where the blank is the name of the Scanner you created on the file. Each iteration through the loop reads three lines. As each of the lines is read from the file, the respective local variables for the Icosahedron data items (label, color, and edge) should be assigned, after which the Icosahedron object should be created and added to a local ArrayList of Icosahedron objects. The next iteration of the loop should then read the next set of three lines then create the next Icosahedron object and add it to the local ArrayList of Icosahedron objects, and so on. After the file has been processed (i.e., when the loop terminates after the hasNext method returns false), name of the list and the ArrayList of Icosahedron objects should be used to create an IcosahedronList object. The list should be printed by printing a leading \n and the IcosahedronList object. Finally, the summary information is printed by printing a leading \n and the value returned by the summaryInfo method invoked on the IcosahedronList object.

Test: You should test your program minimally (1) by reading in the *icosahedron_data_1.txt* input file, which should produce the first output above, and (2) by reading in the *icosahedron_data_0.txt* input file, which should produce the second output above. Although your program may not use all of the methods in IcosahedronList and Icosahedron, you should ensure that all of your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods 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 Icosahedron class should do any input/output (I/O).

2. Be sure to download the test data files (*icosahedron_data_1.txt* and *icosahedron_data_0.txt*) and store them in same folder as your source files. It may be useful examine the contents of the data files. Find the data files in the jGRASP Browse tab and then open each data file in jGRASP to see the items that your program will be reading from the file. Be sure to close the data files without changing them.