Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. You may submit your files to the <u>skeleton code</u> 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 <u>completed code</u> 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.
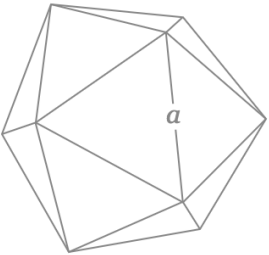
Files to submit to Web-CAT (all three files must be submitted together):
- Icosahedron.java
- IcosahedronList.java
- IcosahedronListMenuApp.java

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, IcosahedronListMenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates an icosahedronList object), (2) print report, (3) print summary, (4) add an icosahedron object to the IcosahedronList object, (5) delete an icosahedron object from the IcosahedronList object, (6) find an icosahedron object in the IcosahedronList object, (7) Edit an icosahedron in the IcosahedronList object, and (8) quit the program. **[You should create a new "Project 6" folder and copy your Project 5 files** (Icosahedron.java, IcosahedronList.java, icosahedron_data_1.txt, and icosahedron_data_0.txt) **to it, rather than work in the same folder as Project 5 files.]**

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)	$A = 5\sqrt{3}\,a^2$ $V = \dfrac{5\,(3+\sqrt{5})}{12}\,a^3$

- **Icosahedron.java** (<u>assuming that you successfully created this class in Project 4, just copy the file to your new Project 6 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.

- 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 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** – extended from Project 5 by **adding the last six methods below. (Assuming that you successfully created this class in Project 5, just copy IcosahedronList.java to your new Project 6 folder and then add the indicated methods. Otherwise, you will need to create all of IcosahedronList.java as part of this project.)**

  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.

- o `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 <u>lines 2 through 19</u> in the output below from IcosahedronListApp for the *icosahedron_data_1.txt* input file. [Note that <u>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.</u>]

- o `summaryInfo`: Returns a String (does <u>not</u> 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 <u>lines 20 through 26</u> 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.

- **The following six methods are new in Project 6:**
  - o `getList`: Returns the ArrayList of Icosahedron objects (the second field above).

  - o `readFile`: Takes a String parameter representing the file name, reads in the file, storing the list name and creating an ArrayList of Icosahedron objects, uses the list name and the ArrayList to create an icosahedronList object, and then returns the IcosahedronList object. See note #1 under <u>Important Considerations</u> for the IcosahedronListMenuApp class (last page) to see how this method should be called.

  - o `addIcosahedron`: Returns nothing but takes three parameters (label, color, and edge), creates a new Icosahedron object, and adds it to the IcosahedronList object.

  - o `findIcosahedron`: Takes a label of an icosahedron as the String parameter and returns the corresponding Icosahedron object if found in the IcosahedronList object; otherwise returns null. <u>Case should be ignored when attempting to match the label</u>.

  - o `deleteIcosahedron`: Takes a String as a parameter that represents the label of the Icosahedron and returns the Icosahedron if it is found in the IcosahedronList object and deleted; otherwise returns null. <u>Case should be ignored when attempting to match the label; consider calling/using `findIcosahedron` in this method</u>.

  - o `editIcosahedron`: Takes three parameters (label, color, and edge), uses the label to find the corresponding the Icosahedron object. If found, sets the color and edge to the values passed in as parameters, and returns true. If not found, returns false.

Code and Test: Remember to import java.util.ArrayList, java.util.Scanner, java.io.File, java.io.FileNotFoundException. These classes will be needed in the readFile method which will require a throws clause for FileNotFoundException. Some of the methods above require that you use a loop to go through the objects in the ArrayList. You may want to implement the class below in parallel with this one to facilitate testing. That is, after implementing one to the methods above, you can implement the corresponding "case" in the switch for menu described below in the IcosahedronListMenuApp class.

- **IcosahedronListMenuApp.java** (replaces IcosahedronListApp class from Project 5)

Requirements: Create an icosahedronListMenuApp class with a main method that presents the user with a menu with eight options, each of which is implemented to do the following: (1) read the input file and create an icosahedronList object, (2) print the IcosahedronList object, (3) print the summary for the IcosahedronList object, (4) add an icosahedron object to the IcosahedronList object, (5) delete an icosahedron object from the IcosahedronList object, (6) find an icosahedron object in the IcosahedronList object, (7) Edit an icosahedron object in the IcosahedronList object, and (8) quit the program.

Design: The main method should print a list of options with the action code and a short description followed by a line with just the action codes prompting the user to select an action. After the user enters an action code, the action is performed, including output if any. Then the line with just the action codes prompting the user to select an action is printed again to accept the next code. The first action a user would normally perform is 'R' to read in the file and create an icosahedronList object. However, the other action codes should work even if an input file has not been processed. The user may continue to perform actions until 'Q' is entered to quit (or end) the program. Note that your program should accept both uppercase and lowercase action codes. Below is output produced after printing the action codes with short descriptions followed by the prompt with the action codes waiting for the user to make a selection.

Line #	Program output
1	Icosahedron List System Menu
2	R – Read File and Create Icosahedron List
3	P – Print Icosahedron List
4	S – Print Summary
5	A – Add Icosahedron
6	D – Delete Icosahedron
7	F – Find Icosahedron
8	E – Edit Icosahedron
9	Q – Quit
10	Enter Code [R, P, S, A, D, F, E, or Q]:

Below shows the screen after the user entered 'r' and then (when prompted) the file name. Notice the output from this action was "File read in and Icosahedron List created". This is followed by the prompt with the action codes waiting for the user to make the next selection. You should use the *icosahedron_data_1.txt* file from Project 5 to test your program.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: r
2	File name: icosahedron_data_1.txt
3	File read in and Icosahedron List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'p' to Print Icosahedron List is shown below and next page.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: p
2	Icosahedron Test List
3	
4	Icosahedron "Small" is "blue" with 30 edges of length 0.01 units.
5	surface area = 0.000866 square units
6	volume = 0.000002 cubic units
7	surface/volume ratio = 396.950723
8	
9	Icosahedron "Medium" is "orange" with 30 edges of length 12.3 units.
10	surface area = 1,310.209833 square units
11	volume = 4,059.844212 cubic units
12	surface/volume ratio = 0.322724
13	
14	Icosahedron "Large" is "white" with 30 edges of length 123.4 units.
15	surface area = 131,874.537977 square units
16	volume = 4,099,581.395236 cubic units
17	surface/volume ratio = 0.032168
18	
19	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 's' to print the summary for the list is shown below.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: s
2	
3	----- Summary for Icosahedron Test List -----
4	Number of Icosahedrons: 3
5	Total Surface Area: 133,184.749
6	Total Volume: 4,103,641.239
7	Average Surface Area: 44,394.916
8	Average Volume: 1,367,880.413
9	Average Surface/Volume Ratio: 132.435
10	
11	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'a' to add an icosahedron object is shown below. Note that after 'a' was entered, the user was prompted for label, color, and edge. Then after the Icosahedron object is added to the Icosahedron List, the message "*** Icosahedron added ***" was printed. This is followed by the prompt for the next action. After you do an "add", you should do a "print" or a "find" to confirm that the "add" was successful.

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: a
2	Label: NewIcosahedron
3	Color: navy blue
4	Edge: 12.5
5	*** Icosahedron added ***
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful "delete" for an icosahedron object, followed by an attempt that was not successful (i.e., the Icosahedron object was not found). Do "p" to confirm the "d".

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
2	`    Label: small`
3	`    "Small" deleted`
4	
5	`Enter Code [R, P, S, A, D, F, E, or Q]: d`
6	`    Label: giant`
7	`    "giant" not found`
8	
9	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "find" for an icosahedron object, followed by an attempt that was not successful (i.e., the Icosahedron object was not found).

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
2	`    Label: medium`
3	`Icosahedron "Medium" is "orange" with 30 edges of length 12.3 units.`
4	`    surface area = 1,310.209833 square units`
5	`    volume = 4,059.844212 cubic units`
6	`    surface/volume ratio = 0.322724`
7	
8	`Enter Code [R, P, S, A, D, F, E, or Q]: f`
9	`    Label: really big`
10	`    "really big" not found`
11	
12	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Here is an example of the successful "edit" for an icosahedron object, followed by an attempt that was not successful (i.e., the Icosahedron object was not found). In order to verify the edit, you should do a "find" for "medium" or you could do a "print" to print the whole list.

Line #	Program output
1	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
2	`    Label: medium`
3	`    Color: bright orange`
4	`    Edge: 25`
5	`    "medium" successfully edited`
6	
7	`Enter Code [R, P, S, A, D, F, E, or Q]: e`
8	`    Label: fake one`
9	`    Color: red`
10	`    Edge: 0.1`
11	`    "fake one" not found`
12	
13	`Enter Code [R, P, S, A, D, F, E, or Q]:`

Finally, below is an example of entering an invalid code, followed by an example of entering a 'q' to quit the application with no message.

Line #	Program output

1
2
3
4
5
	`Enter Code [R, P, S, A, D, F, E, or Q]: g`
	`  *** invalid code ***`
	
	`Enter Code [R, P, S, A, D, F, E, or Q]: q`
	
	`  ----jGRASP: operation complete.`

Code and Test:

Important considerations: This class should import java.util.Scanner, java.util.ArrayList, and java.io.FileNotFoundException. Carefully consider the following information as you develop this class.

1. At the beginning of your main method, you should declare and create an ArrayList of Icosahedron objects and then declare and create an icosahedronList object using the list name and the ArrayList as the parameters in the constructor. This will be an icosahedronList object that contains no Icosahedron objects. For example:

    ```
    String _____ = "*** no list name assigned ***";
    ArrayList<Icosahedron> _____ = new  ArrayList<Icosahedron>();
    IcosahedronList _____ = new IcosahedronList(_____,_____);
    ```

    The 'R' option in the menu should invoke the readFile method on your IcosahedronList object. This will return a new IcosahedronList object based on the data read from the file, and this new IcosahedronList object should replace (be assigned to) your original IcosahedronList object variable in main. Since the readFile method throws FileNotFoundException, your main method needs to do this as well.

2. **Very Important**: **You should declare only one Scanner on System.in for your entire program, and this should be done in the main method.** That is, all input from the keyboard (System.in) must be done in your *main* method.   Declaring more than one Scanner on System.in in your program will likely result in a very low score from Web-CAT.

3. For the menu, your switch statement expression should evaluate to a char, and each case should be a char; alternatively, your switch statement expression should evaluate to a String with a length of 1, and each case should be a String with a length of 1.

After printing the menu of actions with descriptions, you should have a *do-while* loop that prints the prompt with just the action codes followed by a switch statement that performs the indicated action. The *do-while* loop ends when the user enters 'q' to quit. You should strongly consider using a *for-each* loop as appropriate in the new methods that require you to search the list. You should be able to test your program by exercising each of the action codes. After you implement the "Print Icosahedron List" option, you should be able to print the IcosahedronList object after operations such as 'A' and 'D' to see if they worked. You may also want to run in debug mode

with a breakpoint set at the switch statement so that you can step-into your methods if something is not working. In conjunction with running the debugger, you should also create a canvas and drag the items of interest (e.g., the Scanner on the file, your IcosahedronList object, etc.) onto the canvas and save it. As you play or step through your program, you will be able to see the state of these objects change when the 'R', 'A', and 'D' options are selected.

Although your program may not use all of the methods in your Icosahedron and IcosahedronList classes, you should ensure that all of your methods work according to the specification. You can run your program in the canvas and then after the file has been read in, you can call methods on the IcosahedronList object in interactions or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.