

## 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** (all three files must be submitted together):

- Dodecahedron.java
- DodecahedronList2.java
- DodecahedronList2MenuApp.java

## Specifications – **Use arrays in this project; ArrayLists are not allowed!**

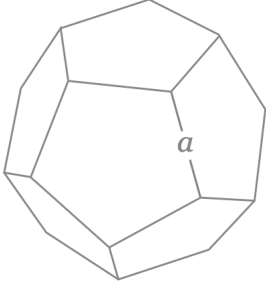
**Overview:** The objective is to modify your Project 6 to use arrays instead of ArrayList objects. You will write a program this week that is composed of three classes: the first class defines Dodecahedron objects, the second class defines DodecahedronList2 objects, and the third, DodecahedronList2MenuApp, presents a menu to the user with eight options and implements these: (1) read input file (which creates a DodecahedronList2 object), (2) print report, (3) print summary, (4) add a Dodecahedron object to the DodecahedronList2 object, (5) delete a Dodecahedron object from the DodecahedronList2 object, (6) find a Dodecahedron object in the DodecahedronList2 object, (7) Edit a Dodecahedron in the DodecahedronList2 object, and (8) quit the program. **[You should create a new “Project 8” folder and copy your Project 6 files (Dodecahedron.java, DodecahedronList.java, DodecahedronListMenuApp.java, dodecahedron\_data\_1.txt, and dodecahedron\_data\_0.txt) to it, rather than work in the same folder as Project 6 files.]**

**To rename an existing .java file, open the file in jGRASP, change the name of the class and the name of the constructor (if it has one) in the source file, and then click the Save button. In the dialog that pops up, click the “Rename and Save” button. This will rename save the .java file and delete the old associated .class file.**

- **Dodecahedron.java (assuming that you successfully created this class in Project 4, 5, or 6 just copy the file to your new Project 8 folder and go on to DodecahedronList2.java on page 4. Otherwise, you will need to create Dodecahedron.java as part of this project.)**

**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).

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

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

**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 these should be the only instance variables in the class.
- (2) **Constructor:** Your Dodecahedron 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 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, which should each be public, 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.

- `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 (does not begin with `\n`) 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.

- **DodecahedronList2.java** – You must use arrays instead of ArrayList objects. (Assuming that you successfully created this class in Project 6, just copy DodecahedronList.java to your new Project 8 folder and then open in jGRASP and rename as DodecahedronList2. To rename an existing .java file, open the file in jGRASP, change the name of the class and the name of the constructor in the source code, and then click the Save button. In the dialog that pops up, click the “Rename and Save” button. This will rename and save the .java file and then delete old associated .class file. Otherwise, you will need to create all of DodecahedronList2.java as part of this project.) In the requirements below, DodecahedronList has been changed to DodecahedronList2. Be sure to make these changes in your methods as necessary.

**Requirements:** Create a DodecahedronList2 class that stores the name of the list and an array of Dodecahedron objects. It also includes methods that return the name of the list, number of Dodecahedron objects in the DodecahedronList2, total surface area, total volume, average surface area, average volume, and average surface to volume ratio for all Dodecahedron objects in the DodecahedronList2. The toString method returns a String containing the name of the list followed by each Dodecahedron in the array, and a summaryInfo method returns summary information about the list (see below).

**Design:** The DodecahedronList2 class has three fields, a constructor, and methods as outlined below.

- (1) **Fields** (or instance variables): (1) a String representing the name of the list, (2) an array of Dodecahedron objects, and (3) an `int` representing the number of elements in the array of Dodecahedron objects. These are the only fields (or instance variables) that this class should have.
- (2) **Constructor:** Your DodecahedronList2 class must contain a constructor that accepts a parameter of type String representing the name of the list, a parameter of type `Dodecahedron[]`, representing the list of Dodecahedron objects, and a parameter of type `int` representing the number of elements in the Dodecahedron array. These parameters should be used to assign the fields described above (i.e., the instance variables).
- (3) **Methods:** The methods for DodecahedronList2 are described below.
  - `getName`: Returns a String representing the name of the list.
  - `numberOfDodecahedrons`: Returns an `int` representing the number of Dodecahedron objects in the DodecahedronList2. If there are zero Dodecahedron objects in the list, zero should be returned.
  - `totalSurfaceArea`: Returns a double representing the total surface areas for all Dodecahedron objects in the list. If there are zero Dodecahedron objects in the list, zero should be returned.
  - `totalVolume`: Returns a double representing the total volumes for all Dodecahedron objects in the list. If there are zero Dodecahedron objects in the list, zero should be returned.

- `averageSurfaceArea`: Returns a double representing the average surface area for all Dodecahedron objects in the list. If there are zero Dodecahedron objects in the list, zero should be returned.
- `averageVolume`: Returns a double representing the average volume for all Dodecahedron objects in the list. If there are zero Dodecahedron objects in the list, zero should be returned.
- `averageSurfaceToVolumeRatio`: Returns a double representing the average surface to volume ratio for all Dodecahedron objects in the list. If there are zero Dodecahedron objects in the list, zero should be returned.
- `toString`: Returns a String (does not begin with `\n`) containing the name of the list followed by each Dodecahedron in the list. In the process of creating the return result, this `toString()` method should include a while loop that calls the `toString()` method for each Dodecahedron 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 DodecahedronList2App for the *Dodecahedron\_data\_1.txt* input file. [Note that the `toString` result should **not** include the return value of `summaryInfo()`.]
- `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 Dodecahedrons, 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.
- `getList`: Returns the array of Dodecahedron objects (the second field above).
- `readFile`: Takes a String parameter representing the file name, reads in the file, storing the list name and creating an array of Dodecahedron objects, uses the list name, the array, and number of Dodecahedron objects in the array to create a DodecahedronList2 object, and then returns the DodecahedronList2 object. See note #1 under [Important Considerations](#) for the DodecahedronList2MenuApp class (last page) to see how this method should be called.
- `addDodecahedron`: Returns nothing but takes three parameters (label, color, and edge), creates a new Dodecahedron object, and adds it to the DodecahedronList2 object.
- `findDodecahedron`: Takes a label of a Dodecahedron as the String parameter and returns the corresponding Dodecahedron object if found in the DodecahedronList2 object; otherwise returns null. Case should be ignored when attempting to match the label.
- `deleteDodecahedron`: Takes a String as a parameter that represents the label of the Dodecahedron and returns the Dodecahedron if it is found in the DodecahedronList2 object and deleted; otherwise returns null. Case should be ignored when attempting to match the label; consider calling/using `findDodecahedron` in this method. When an element is deleted from an array, elements to the right of the deleted element must be shifted to the left. After shifting the items to the left, the last Dodecahedron element in the array should be set to null. Finally, the number of elements field must be decremented.
- `editDodecahedron`: Takes three parameters (label, color, and edge), uses the label to find the corresponding the Dodecahedron object in the list. 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.Scanner`, `java.io.File`, `java.io.IOException`. These classes will be needed in the `readFile` method which will require a `throws` clause for `IOException`. Some of the methods above require that you use a loop to go through the objects in the array. 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 `DodecahedronList2MenuApp` class.

- **DodecahedronList2MenuApp.java** (replaces `DodecahedronListMenuApp` class from Project 6; the file and class name in the file must be renamed to reflect `DodecahedronList2MenuApp`). **To rename an existing .java file, open the file in jGRASP, change the name of the class in the source code, and then click the Save button. In the dialog that pops up, click the “Rename and Save” button. This will rename the .java file and delete the old associated .class file. Be sure to make these changes in your main method as necessary.**

**Requirements:** Create a `DodecahedronList2MenuApp` 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 a `DodecahedronList2` object, (2) print the `DodecahedronList2` object, (3) print the summary for the `DodecahedronList2` object, (4) add a `Dodecahedron` object to the `DodecahedronList2` object, (5) delete a `Dodecahedron` object from the `DodecahedronList2` object, (6) find a `Dodecahedron` object in the `DodecahedronList2` object, (7) Edit a `Dodecahedron` object in the `DodecahedronList2` 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 a `DodecahedronList2` 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	Dodecahedron List System Menu
2	R - Read File and Create Dodecahedron List
3	P - Print Dodecahedron List
4	S - Print Summary
5	A - Add Dodecahedron
6	D - Delete Dodecahedron
7	F - Find Dodecahedron
8	E - Edit Dodecahedron
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 Dodecahedron 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 *dodecahedron\_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]: <b>r</b>
2	File name: <b>dodecahedron_data_1.txt</b>
3	File read in and Dodecahedron List created
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]:

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

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: <b>p</b>
2	Dodecahedron Test List
3	
4	Dodecahedron "Small Example" is "blue" with 30 edges of length 0.25 units.
5	surface area = 1.29 square units
6	volume = 0.12 cubic units
7	surface/volume ratio = 10.777
8	
9	Dodecahedron "Medium Example" is "orange" with 30 edges of length 10.1 units.
10	surface area = 2,106.071 square units
11	volume = 7,895.319 cubic units
12	surface/volume ratio = 0.267
13	
14	Dodecahedron "Large Example" is "silver" with 30 edges of length 200.5 units.
15	surface area = 829,963.459 square units
16	volume = 61,765,889.248 cubic units
17	surface/volume ratio = 0.013
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]: <b>s</b>
2	
3	----- Summary for Dodecahedron Test List -----
4	Number of Dodecahedrons: 3
5	Total Surface Area: 832,070.821
6	Total Volume: 61,773,784.687
7	Average Surface Area: 277,356.94
8	Average Volume: 20,591,261.562
9	Average Surface/Volume Ratio: 3.686
10	
11	Enter Code [R, P, S, A, D, F, E, or Q]:

The result of the user selecting 'a' to add a Dodecahedron object is shown below. Note that after 'a' was entered, the user was prompted for label, color, and edge. Then after the Dodecahedron

object is added to the Dodecahedron List, the message “\*\*\* Dodecahedron 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: d1
3	Color: gold
4	Edge: 5.8
5	*** Dodecahedron added ***
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful “delete” for a Dodecahedron object, followed by an attempt that was not successful (i.e., the Dodecahedron 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: Medium Example
3	"Medium Example" deleted
4	
5	Enter Code [R, P, S, A, D, F, E, or Q]: d
6	Label: not a real object
7	"not a real object" not found
8	
9	Enter Code [R, P, S, A, D, F, E, or Q]:

Here is an example of the successful “find” for a Dodecahedron object, followed by an attempt that was not successful (i.e., the Dodecahedron object was not found).

Line #	Program output
1	Enter Code [R, P, S, A, D, F, E, or Q]: f
2	Label: small EXAMPLE
3	Dodecahedron "Small Example" is "blue" with 30 edges of length 0.25 units.
4	surface area = 1.29 square units
5	volume = 0.12 cubic units
6	surface/volume ratio = 10.777
7	
8	Enter Code [R, P, S, A, D, F, E, or Q]: f
9	Label: tiny obj
10	"tiny obj" not found
11	
12	Enter Code [R, P, S, A, D, F, E, or Q]:

Next is an example of the successful “edit” for a Dodecahedron object, followed by an attempt that was not successful (i.e., the Dodecahedron object was not found). In order to verify the edit, you should do a “find” for “wide example” 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: small EXAMPLE
3	Color: green
4	Edge: 0.1
5	"small EXAMPLE" successfully edited
6	
7	Enter Code [R, P, S, A, D, F, E, or Q]: e
8	Label: not there
9	Color: yellow
10	Edge: 12.4
11	"not there" 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	Enter Code [R, P, S, A, D, F, E, or Q]: b
2	*** invalid code ***
3	
4	Enter Code [R, P, S, A, D, F, E, or Q]: q
5	----jGRASP: operation complete.

### Code and Test:

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

1. At the beginning of your main method, you should declare and create an array of Dodecahedron objects and then declare and create a DodecahedronList2 object using the list name, the array, and 0 as the parameters in the constructor. This will be a DodecahedronList2 object that contains no Dodecahedron objects. For example:

```
String _____ = "**** no list name assigned ****";
Dodecahedron[] _____ = new Dodecahedron[100];
DodecahedronList2 _____ = new DodecahedronList2(_____, _____, _____);
```

The 'R' option in the menu should invoke the `readFile` method on your DodecahedronList2 object. This will return a new DodecahedronList2 object based on the data read from the file, and this new DodecahedronList2 object should replace (be assigned to) your original DodecahedronList2 object variable in main. Since the `readFile` method throws `IOException`, 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 Dodecahedron List" option, you should be able to print the DodecahedronList2 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 drag the items of interest (e.g., the Scanner on the file, your DodecahedronList2 object, etc.) onto the canvas and save it. As you play or step through your program, you'll 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 Dodecahedron and DodecahedronList2 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 DodecahedronList2 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.