

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (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 assignment. 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. The grades for the **Part A Completed Code** submission (two files) and **Part B Completed Code** (four files) will be determined by the tests that you pass or fail in your test files and by the level of coverage attained in your source files as well as usual correctness tests in Web-CAT.

Files to submit to Web-CAT:

Part A

- ConicalFrustum.java, ConicalFrustumTest.java

Part B

- ConicalFrustum.java, ConicalFrustumTest.java
- ConicalFrustumList2.java, ConicalFrustumList2Test.java

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

Overview: This project consists of four classes: (1) ConicalFrustum is a class representing a ConicalFrustum object; (2) ConicalFrustumTest class is a JUnit test class which contains one or more test methods for each method in the ConicalFrustum class; (3) ConicalFrustumList2 is a class representing a ConicalFrustum list object; and (4) ConicalFrustumList2Test class is a JUnit test class which contains one or more test methods for each method in the ConicalFrustumList2 class. Note that there is no requirement for a class with a main method in this project.

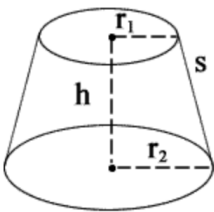
Since you will be modifying classes from the previous project, I strongly recommend that you create a new folder for this project with a copy of your ConicalFrustum.java file and ConicalFrustumList2.java file from the previous project.

You must create a jGRASP project and add your ConicalFrustum.java file and ConicalFrustumList2.java file. With this project is open, your test files will be automatically added to the project when they are created. You will be able to run all test files by clicking the JUnit run button on the Open Projects toolbar.

New requirements and design specifications are underlined in the descriptions below to help you identify them.

- **ConicalFrustum.java** (a modification of the ConicalFrustum class in the previous project; new requirements are underlined below)

Requirements: Create a ConicalFrustum class that stores the label, radius of top, radius of bottom, and height where the radii and height are non-negative. The ConicalFrustum class also includes methods to set and get each of these fields, as well as methods to calculate the volume, slant height, lateral surface area, and total surface area of a ConicalFrustum object, and a method to provide a String value that describes a ConicalFrustum object.

A <i>Conical Frustum</i> is a Frustum created by slicing the top off a cone (with the cut made parallel to the base), forming a lower base and an upper base that are circular and parallel.		
	r_1 radius of top	$V = \frac{\pi * h}{3} (r_1^2 + r_2^2 + (r_1 * r_2))$ $s = \sqrt{(r_1 - r_2)^2 + h^2}$ $S = \pi * (r_1 + r_2) * s$ $A = \pi * (r_1^2 + r_2^2 + (r_1 + r_2) * s)$
	r_2 radius of bottom	
	h height	
	s slant height	
	S lateral surface area	
	V volume	
	A total surface area	

Source for figures and formulas: <https://www.calculatorsoup.com/images/frustum001.gif>

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

- (1) **Fields** (instance variables): label of type `String`, radius1 of type `double`, radius2 of type `double`, and height of type `double`. Initialize the `String` to "" and the `double` variables to 0 in their respective declarations. These instance variables should be private so that they are not directly accessible from outside of the ConicalFrustum class, and these should be the only instance variables (fields) in the class.

Class Variable - count of type `int` should be private and static, and it should be initialized to zero.

- (2) **Constructor:** Your ConicalFrustum class must contain a public constructor that accepts four parameters (see types of above) representing the label, radius1, radius2, and height. Instead of assigning the parameters directly to the fields, the respective set method for each field (described below) should be called since they are checking the validity of the parameter. For example, instead of using the statement `label = labelIn;` use the statement `setLabel(labelIn);` Below are examples of how the constructor could be used to create ConicalFrustum 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.

The constructor should increment the class variable count each time a ConicalFrustum is constructed.

```
ConicalFrustum example1 = new ConicalFrustum("Small", 0.5, 0.75, 0.25);
```

```
ConicalFrustum example2 = new ConicalFrustum(" Medium ", 5.1, 10.2, 15.9);
```

```
ConicalFrustum example3 = new ConicalFrustum("Large", 98.32, 199.0, 250.0);
```

- (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 `ConicalFrustum`, which should each be public, are described below. See the formulas in the figure above and the Code and Test section below for information on constructing these methods.
- `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.
 - `getRadius1`: Accepts no parameters and returns a `double` representing the radius1 field.
 - `setRadius1`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is non-negative, then the parameter is set to the radius1 field and the method returns `true`. Otherwise, the method returns `false` and the radius1 field is not set.
 - `getRadius2`: Accepts no parameters and returns a `double` representing the radius2 field.
 - `setRadius2`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is non-negative, then the parameter is set to the radius2 field and the method returns `true`. Otherwise, the method returns `false` and the radius2 field is not set.
 - `getHeight`: Accepts no parameters and returns a `double` representing the height field.
 - `setHeight`: Accepts a `double` parameter and returns a `boolean` as follows. If the `double` parameter is non-negative, then the parameter is set to the height field and the method returns `true`. Otherwise, the method returns `false` and the height field is not set.
 - `volume`: Accepts no parameters and returns the `double` value for the volume of the `ConicalFrustum`. *[Be sure to avoid integer division in your expression.]*
 - `slantHeight`: Accepts no parameters and returns the `double` value for the slant height of the `ConicalFrustum`.
 - `lateralSurfaceArea`: Accepts no parameters and returns the `double` value for the lateral surface area of the `ConicalFrustum`. Be sure to call your `slantHeight` method as appropriate.
 - `totalSurfaceArea`: Accepts no parameters and returns the `double` value for the total surface area of the `ConicalFrustum`. Be sure to call your `slantHeight` method as appropriate.
 - `toString`: Returns a `String` containing the information about the `ConicalFrustum` 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:

`volume()`, `slantHeight()`, `lateralSurfaceArea()`, and `totalSurfaceArea()`. 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).

ConicalFrustum "Small" with radius1 = 0.5, radius2 = 0.75, and height = 0.25 has:

```
volume = 0.311 cubic units
slant height = 0.354 units
lateral surface area = 1.388 units
total surface area = 3.941 square units
```

ConicalFrustum "Medium" with radius1 = 5.1, radius2 = 10.2, and height = 15.9 has:

```
volume = 3,031.546 cubic units
slant height = 16.698 units
lateral surface area = 802.608 units
total surface area = 1,211.172 square units
```

ConicalFrustum "Large" with radius1 = 98.32, radius2 = 199.0, and height = 250.0 has:

```
volume = 18,020,568.788 cubic units
slant height = 269.512 units
lateral surface area = 251,739.485 units
total surface area = 406,518.914 square units
```

- `getCount`: A static method that accepts no parameters and returns an `int` representing the static count field.
- `resetCount`: A static method that returns nothing, accepts no parameters, and sets the static count field to zero.
- `equals`: An instance method that accepts a parameter of type `Object` and returns `false` if the `Object` is a not a `ConicalFrustum`; otherwise, when cast to a `ConicalFrustum`, if it has the same field values as the `ConicalFrustum` upon which the method was called, it returns `true`; otherwise, it returns `false`. Note that this `equals` method with parameter type `Object` will be called by the JUnit `Assert.assertEquals` method when two `ConicalFrustum` objects are checked for equality.

Below is a version you are free to use.

```
public boolean equals(Object obj) {
    if (!(obj instanceof ConicalFrustum)) {
        return false;
    }
    else {
        ConicalFrustum d = (ConicalFrustum) obj;
        return (label.equalsIgnoreCase(d.getLabel())
            && Math.abs(radius1 - d.getRadius1()) < .000001
            && Math.abs(radius2 - d.getRadius2()) < .000001
            && Math.abs(height - d.getHeight()) < .000001);
    }
}
```

- hashCode() : Accepts no parameters and returns zero of type int. This method is required by Checkstyle if the equals method above is implemented.

Code and Test: As you implement the methods in your ConicalFrustum class, you should compile it and then create test methods as described below for the ConicalFrustumTest class.

- **ConicalFrustumTest.java**

Requirements: Create a ConicalFrustumTest class that contains a set of test methods to test each of the methods in ConicalFrustum.

Design: Typically, in each test method, you will need to create an instance of ConicalFrustum, call the method you are testing, and then make an assertion about the expected result and the actual result (note that the actual result is commonly the result of invoking the method unless it has a void return type). You can think of a test method as simply formalizing or codifying what you have been doing in interactions to make sure a method is working correctly. That is, the sequence of statements that you would enter in interactions to test a method should be entered into a single test method. You should have at least one test method for each method in ConicalFrustum, except for associated getters and setters which can be tested in the same method. However, if a method contains conditional statements (e.g., an if statement) that results in more than one distinct outcome, you need a test method for each outcome. For example, if the method returns boolean, you should have one test method where the expected return value is false and another test method that expects the return value to be true. Also, each condition in boolean expression must be exercised true and false. Collectively, these test methods are a set of test cases that can be invoked with a single click to test all of the methods in your ConicalFrustum class.

Code and Test: Since this is the first project requiring you to write JUnit test methods, a good strategy would be to begin by writing test methods for those methods in ConicalFrustum that you “know” are correct. By doing this, you will be able to concentrate on the getting the test methods correct. That is, if the test method *fails*, it is most likely due to a defect in the test method itself rather the ConicalFrustum method being testing. As you become more familiar with the process of writing test methods, you will be better prepared to write the test methods for the new methods in ConicalFrustum. Be sure to call the ConicalFrustum toString method in one of your test cases so that Web-CAT will consider the toString method to be “covered” in its coverage analysis. Remember that you can set a breakpoint in a JUnit test method and run the test file in Debug mode. Then, when you have an instance in the Debug tab, you can unfold it to see its values or you can open a canvas window and drag items from the Debug tab onto the canvas.

- **ConicalFrustumList2.java** (a modification of the **ConicalFrustumList2** class in the previous project; new requirements are underlined below)

Requirements: Create a ConicalFrustumList2 class that stores the name of the list and an array of ConicalFrustum objects. It also includes methods that return the name of the list, number of ConicalFrustum objects in the ConicalFrustumList2, total surface area, total volume, average

surface area, and average volume for all ConicalFrustum objects in the ConicalFrustumList2. The toString method returns a String containing the name of the list followed by each ConicalFrustum in the array, and a summaryInfo method returns summary information about the list (see below).

Design: The ConicalFrustumList2 class has fields, a constructor, and methods as outlined below. These instance variables should be private, and they should be the only instance variables (fields) in the class.

- (1) **Fields** (or instance variables): (1) a String representing the name of the list, (2) an array of ConicalFrustum objects, and (3) an int representing the number of elements in the array of ConicalFrustum objects. These are the only fields (or instance variables) that this class should have.
- (2) **Constructor:** Your ConicalFrustumList2 class must contain a constructor that accepts (1) a parameter of type String representing the name of the list, (2) a parameter of type ConicalFrustum[] representing the list of ConicalFrustum objects, and (3) a parameter of type int representing the number of elements in the ConicalFrustum array. These parameters should be used to assign the fields described above (i.e., the instance variables).
- (3) **Methods:** The methods for ConicalFrustumList2 are described below.
 - o getName: Returns a String representing the name of the list.
 - o numberOfConicalFrustums: Returns an int representing the number of ConicalFrustum objects in the ConicalFrustumList2. If there are zero ConicalFrustum objects in the list, zero should be returned.
 - o totalSurfaceArea: Returns a double representing the total surface areas for all ConicalFrustum objects in the list. If there are zero ConicalFrustum objects in the list, zero should be returned.
 - o totalVolume: Returns a double representing the total volumes for all ConicalFrustum objects in the list. If there are zero ConicalFrustum objects in the list, zero should be returned.
 - o averageSurfaceArea: Returns a double representing the average for the total surface area for all ConicalFrustum objects in the list. If there are zero ConicalFrustum objects in the list, zero should be returned.
 - o averageVolume: Returns a double representing the average volume for all ConicalFrustum objects in the list. If there are zero ConicalFrustum objects in the list, zero should be returned.
 - o toString: Returns a String (does not begin with \n) containing the name of the list followed by each ConicalFrustum 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 ConicalFrustum object in the list (adding a \n before and after each). Be sure to include appropriate newline escape sequences. For an example, see [lines 3 through 21](#) in the output from ConicalFrustumListApp (previous project) for the *ConicalFrustum_data_1.txt* input file. [Note that the toString result should **not** include

the summary items in lines 24 through 29 of the example. These lines represent the return value of the summaryInfo method below which will be called from main.]

- summaryInfo: Returns a String (does not begin or end 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 ConicalFrustums, total surface area, total volume, average surface area, and average volume. Use "#,##0.0##" as the pattern to format the double values. For an example, see lines 24 through 29 in the output below from ConicalFrustumList2App for the *ConicalFrustum_data_1.txt* input file. The second example below shows the output from ConicalFrustumList2App for the *ConicalFrustum_data_0.txt* input file which contains a list name but no ConicalFrustum data.
- getList: Returns the array of ConicalFrustum 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 ConicalFrustum objects, uses the list name the array, and the number of ConicalFrustum objects in the array to create a new ConicalFrustumList2 object, and then returns the ConicalFrustumList2 object. See note #2 under Important Considerations for the ConicalFrustumList2MenuApp class to see how this method should be called.
- addConicalFrustum: Returns nothing but takes four parameters (label, radius1, radius2, and height), creates a new ConicalFrustum object, and adds it to the ConicalFrustumList2 object.
- findConicalFrustum: Takes a label of a ConicalFrustum as the String parameter and returns the corresponding ConicalFrustum object if found in the ConicalFrustumList2 object; otherwise returns null. Letter case should be ignored when attempting to match the label (e.g., "Small Example" and "sMaLL EXAMPLE" should be a match).
- deleteConicalFrustum: Takes a String as a parameter that represents the label of the ConicalFrustum. Returns the ConicalFrustum object if it is found in the ConicalFrustumList2 and deleted; otherwise returns null. Letter case should be ignored when attempting to match the label; consider calling/using findConicalFrustum 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 ConicalFrustum element in the array should be set to null. Finally, the number of elements field must be decremented.
- editConicalFrustum: Takes four parameters (label, radius1, radius2, and height), uses the label to find the corresponding the ConicalFrustum object. If found, sets the radius1, radius2, and height to the values passed in as parameters, and returns true. If not found, returns false.

New method for this Project

- findConicalFrustumWithLeastHeight: Returns the ConicalFrustum with the least height; if the list contains no ConicalFrustum objects, returns null.
- findConicalFrustumWithGreatestHeight: Returns the ConicalFrustum with the greatest height; if the list contains no ConicalFrustum objects, returns null.
- findConicalFrustumWithLeastVolume: Returns the ConicalFrustum with the least or smallest volume; if the list contains no ConicalFrustum objects, returns null.

- findConicalFrustumWithGreatestVolume: Returns the ConicalFrustum with the greatest volume; if the list contains no ConicalFrustum objects, returns null.

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 test method in the test file described below.

- **ConicalFrustumList2Test.java**

Requirements: Create a ConicalFrustumList2Test class that contains a set of *test* methods to test each of the methods in ConicalFrustumList2.

Design: Typically, in each test method, you will need to create an instance of ConicalFrustumList2, call the method you are testing, and then make an assertion about the expected result and the actual result (note that the actual result is usually the result of invoking the method unless it has a void return type). You can think of a test method as simply formalizing or codifying what you have been doing in interactions to make sure a method is working correctly. That is, the sequence of statements that you would enter in interactions to test a method should be entered into a single test method. You should have at least one test method for each method in ConicalFrustumList2. However, if a method contains conditional statements (e.g., an *if* statement) that results in more than one distinct outcome, you need a test method for each outcome. For example, if the method returns boolean, you should have one test method where the expected return value is false and another test method that expects the return value to be true. Also, each condition in boolean expression must be exercised true and false. Collectively, these test methods are a set of test cases that can be invoked with a single click to test all of the methods in your ConicalFrustumList2 class.

Code and Test: Since this is the first project requiring you to write JUnit test methods, a good strategy would be to begin by writing test methods for those methods in ConicalFrustumList2 that you “know” are correct. By doing this, you will be able to concentrate on the getting the test methods correct. That is, if the test method *fails*, it is most likely due to a defect in the test method itself rather the ConicalFrustumList2 method being testing. As you become more familiar with the process of writing test methods, you will be better prepared to write the test methods for the new methods in ConicalFrustumList2. Be sure to call the ConicalFrustumList2 toString method in one of your test cases so that Web-CAT will consider the toString method to be “covered” in its coverage analysis. Remember that you can set a breakpoint in a JUnit test method and run the test file in Debug mode. Then, when you have an instance in the Debug tab, you can unfold it to see its values or you can open a canvas window and drag items from the Debug tab onto the canvas.

Important: When comparing two arrays for equality in JUnit, be sure to use Assert.assertArrayEquals rather than Assert.assertEquals. Assert.assertArrayEquals will return true only if the two arrays are the same length and the elements are equal based on an element by element comparison using the appropriate equals method.

Web-CAT

Assignment Part A – submit: ConicalFrustum.java, ConicalFrustumTest.java

Assignment Part B – submit: ConicalFrustum.java, ConicalFrustumTest.java, ConicalFrustumList2.java, and ConicalFrustumList2Test.java.

Note that data files ConicalFrustum_data_1.txt and ConicalFrustum_data_0.txt are available in Web-CAT for you to use in your test methods. If you want to use your own data files, they should have a .txt extension, and they should be included with submission to Web-CAT (i.e., just add the .txt data file to your jGRASP project in the Source Files category).

Web-CAT will use the results of your test methods and their level of coverage of your source files as well as the results of our reference correctness tests to determine your grade.