

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

- Spherocylinder.java, SpherocylinderTest.java

Part B

- Spherocylinder.java, SpherocylinderTest.java
- SpherocylinderList2.java, SpherocylinderList2Test.java

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

Overview: This project consists of four classes: (1) Spherocylinder is a class representing a Spherocylinder object; (2) SpherocylinderTest class is a JUnit test class which contains one or more test methods for each method in the Spherocylinder class; (3) SpherocylinderList2 is a class representing a Spherocylinder list object; and (4) SpherocylinderList2Test class is a JUnit test class which contains one or more test methods for each method in the SpherocylinderList2 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 Spherocylinder.java file and SpherocylinderList2.java file from the previous project.

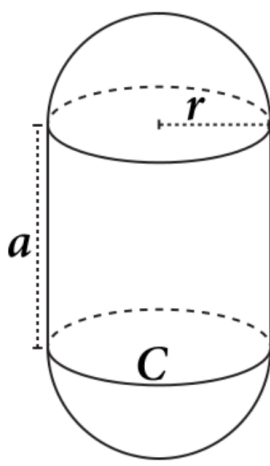
You must create a jGRASP project and add your Spherocylinder.java file and SpherocylinderList2.java file. With this project is open, your test files will be automatically added to the project when they are created. You will be able to run all test files by clicking the JUnit run button on the Open Projects toolbar.

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

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

Requirements: Create a Spherocylinder class that stores the label, radius, and cylinder height where both radius and cylinder height are non-negative. The Spherocylinder class also includes methods to set and get each of these fields, as well as methods to calculate the circumference, surface area, and volume of a Spherocylinder object, and a method to provide a String value that describes a Spherocylinder object.

A spherocylinder (or capsule) is a 3-dimensional object made up of two hemispheres connected by a cylinder as shown below. The formulas are provided to assist you in computing return values for the respective Spherocylinder methods described in this project. Source for figures and formulas: [https://en.wikipedia.org/wiki/Capsule_\(geometry\)](https://en.wikipedia.org/wiki/Capsule_(geometry))

	<p>The variables are abbreviated as follows:</p> <p><i>r</i> is radius <i>a</i> is cylinder height <i>C</i> is Circumference <i>SA</i> is Surface Area <i>V</i> is Volume</p>	$C = 2 \pi r$ $SA = 2\pi r(2r + a)$ $V = \pi r^2 \left(\frac{4}{3}r + a \right)$
--	---	---

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

- (1) **Fields:** Instance Variables - label of type `String`, radius of type `double`, and cylinder 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 Spherocylinder 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 Spherocylinder class must contain a public constructor that accepts three parameters (see types of above) representing the label, radius, and cylinder 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);`

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

Below are examples of how the constructor could be used to create Spherocylinder 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.

```
Spherocylinder example1 = new Spherocylinder("Small Example", 0.5, 0.25);
Spherocylinder example2 = new Spherocylinder(" Medium Example ", 10.8, 10.1);
Spherocylinder example3 = new Spherocylinder("Large Example", 98.32, 99.0);
```

(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 Spherocylinder, 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.
- `getRadius`: Accepts no parameters and returns a double representing the radius field.
- `setRadius`: Takes a double parameter and returns a boolean. If the double parameter is non-negative, then the parameter is set to the radius field and the method returns true. Otherwise, the method returns false and the radius field is not set.
- `getCylinderHeight`: Accepts no parameters and returns a double representing the cylinder height field.
- `setCylinderHeight`: Accepts a double parameter and returns a boolean as follows. If the double parameter is non-negative, then the parameter is set to the cylinder height field and the method returns true. Otherwise, the method returns false and the cylinder height field is not set.
- `circumference`: Accepts no parameters and returns the double value for the circumference of the Spherocylinder.
- `surfaceArea`: Accepts no parameters and returns the double value for the total surface area of the Spherocylinder.
- `volume`: Accepts no parameters and returns the double value for the volume of the Spherocylinder. *[Be sure to avoid integer division in your expression.]*
- `toString`: Returns a String containing the information about the Spherocylinder 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 within the String but it should not begin or end with a newline. In addition to the field values (or corresponding “get” methods), the following methods should be used to compute appropriate values in the `toString` method: `circumference()`,

`surfaceArea()`, and `volume()`. 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).

Spherocylinder "Small Example" with radius 0.5 and cylinder height 0.25 has:

```
circumference = 3.142 units
surface area = 3.927 square units
volume = 0.72 cubic units
```

Spherocylinder "Medium Example" with radius 10.8 and cylinder height 10.1 has:

```
circumference = 67.858 units
surface area = 2,151.111 square units
volume = 8,977.666 cubic units
```

Spherocylinder "Large Example" with radius 98.32 and cylinder height 99.0 has:

```
circumference = 617.763 units
surface area = 182,635.388 square units
volume = 6,987,754.655 cubic 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 not a `Spherocylinder`; otherwise, when cast to a `Spherocylinder`, if it has the same field values as the `Spherocylinder` upon which the method was called. Otherwise, it returns `false`. Note that this `equals` method with parameter type `Object` will be called by the JUnit `Assert.assertEquals` method when two `Spherocylinder` objects are checked for equality.

Below is a version you are free to use.

```
public boolean equals(Object obj) {

    if (!(obj instanceof Spherocylinder)) {
        return false;
    }
    else {
        Spherocylinder d = (Spherocylinder) obj;
        return (label.equalsIgnoreCase(d.getLabel())
                && Math.abs(radius - d.getRadius()) < .000001
                && Math.abs(cylinderHeight - d.getCylinderHeight())
                < .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 `Spherocylinder` class, you should compile it and then create test methods as described below for the `SpherocylinderTest` class.

- **SpherocylinderTest.java**

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

Design: Typically, in each test method, you will need to create an instance of Spherocylinder, 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 Spherocylinder, 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 Spherocylinder 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 Spherocylinder 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 than the Spherocylinder 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 Spherocylinder. Be sure to call the Spherocylinder 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.

- **SpherocylinderList2.java** (a modification of the **SpherocylinderList2** class in the previous project; new requirements are underlined below)

Requirements: Create a SpherocylinderList2 class that stores the name of the list and an array of Spherocylinder objects. It also includes methods that return the name of the list, number of Spherocylinder objects in the SpherocylinderList2, total surface area, total volume, average surface area, and average volume for all Spherocylinder objects in the SpherocylinderList2. The toString method returns a String containing the name of the list followed by each Spherocylinder in the array, and a summaryInfo method returns summary information about the list (see below).

Design: The SpherocylinderList2 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 Spherocylinder objects, and (3) an `int` representing the number of elements in the array of Spherocylinder objects. These are the only fields (or instance variables) that this class should have.
- (2) **Constructor:** Your SpherocylinderList2 class must contain a constructor that accepts three parameters: (1) a parameter of type String representing the name of the list, (2) a parameter of type `Spherocylinder[]`, representing the list of Spherocylinder objects, and (3) a parameter of type `int` representing the number of elements in the Spherocylinder array. These parameters should be used to assign the fields described above (i.e., the instance variables).
- (3) **Methods:** The methods for SpherocylinderList2 are described below.
 - `getName`: Returns a String representing the name of the list.
 - `numberOfSpherocylinders`: Returns an `int` representing the number of Spherocylinder objects in the SpherocylinderList2. If there are zero Spherocylinder objects in the list, zero should be returned.
 - `totalSurfaceArea`: Returns a double representing the total surface areas for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
 - `totalVolume`: Returns a double representing the total volumes for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
 - `averageSurfaceArea`: Returns a double representing the average surface area for all Spherocylinder objects in the list. If there are zero Spherocylinder objects in the list, zero should be returned.
 - `averageVolume`: Returns a double representing the average volume for all Spherocylinder objects in the list. If there are zero Spherocylinder 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 Spherocylinder 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 Spherocylinder 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 19 in the output below for option P in SpherocylinderList2App after *Spherocylinder_data_1.txt* has been read in with option R. [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 Spherocylinders, total surface area, total volume, average surface area, and average volume. Use `"#,##0.0###"` as the pattern to format the double values.

- `getList`: Returns the array of Spherocylinder 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 Spherocylinder objects, uses the list name, the array, and number of Spherocylinder objects in the array to create a SpherocylinderList2 object, and then returns the SpherocylinderList2 object. See note #1 under Important Considerations for the SpherocylinderList2MenuApp class (last page) to see how this method should be called.
- `addSpherocylinder`: Returns nothing but takes three parameters (label, radius, and cylinder height), creates a new Spherocylinder object, and adds it to the SpherocylinderList2 object.
- `findSpherocylinder`: Takes a label of a Spherocylinder as the String parameter and returns the corresponding Spherocylinder object if found in the SpherocylinderList2 object; otherwise returns null. Case should be ignored when attempting to match the label.
- `deleteSpherocylinder`: Takes a String as a parameter that represents the label of the Spherocylinder and returns the Spherocylinder if it is found in the SpherocylinderList2 object and deleted; otherwise returns null. Case should be ignored when attempting to match the label; consider calling/using findSpherocylinder 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 Spherocylinder element in the array should be set to null. Finally, the number of elements field must be decremented.
- `editSpherocylinder`: Takes three parameters (label, radius, and cylinder height), uses the label to find the corresponding the Spherocylinder object in the list. If found, sets the radius and cylinder height to the values passed in as parameters, and returns true. If not found, returns false.

New method for this Project

- `findSpherocylinderWithShortestRadius`: Returns the Spherocylinder with the shortest radius; if the list contains no Spherocylinder objects, returns null.
- `findSpherocylinderWithLongestRadius`: Returns the Spherocylinder with the longest radius; if the list contains no Spherocylinder objects, returns null.
- `findSpherocylinderWithSmallestVolume`: Returns the Spherocylinder with the smallest volume; if the list contains no Spherocylinder objects, returns null.
- `findSpherocylinderWithLargestVolume`: Returns the Spherocylinder with the largest volume; if the list contains no Spherocylinder 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.

- **SpherocylinderList2Test.java**

Requirements: Create a SpherocylinderList2Test class that contains a set of *test* methods to test each of the methods in SpherocylinderList2.

Design: Typically, in each test method, you will need to create an instance of SpherocylinderList2, 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 SpherocylinderList2. 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 SpherocylinderList2 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 SpherocylinderList2 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 SpherocylinderList2 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 SpherocylinderList2. Be sure to call the SpherocylinderList2 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.

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: Spherocylinder.java, SpherocylinderTest.java

Assignment Part B – submit: Spherocylinder.java, SpherocylinderTest.java, SpherocylinderList2.java, and SpherocylinderList2Test.java.

Note that data files Spherocylinder_data_1.txt and Spherocylinder_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.