Project: Spherocylinder with JUnit Tests – Part 3

Deliverables

Your project files should be submitted to the grading system by the due date and time specified. Note that there is also an optional Skeleton Code assignment (ungraded) which will ensure that you have classes and methods named correctly and also that you have the correct return types and parameter types. This ungraded assignment will also indicate level of coverage your tests have achieved. 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 files to the Completed Code assignment no later than 11:59 PM on the due date. Your grade will be determined, in part, by the tests that you pass or fail in your test file and by the level of coverage attained in your source file, as well as our usual correctness tests.

Files to submit to the grading system:

- Spherocylinder.java, SpherocylinderTest.java
- SpherocylinderList.java, SpherocylinderListTest.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) SpherocylinderList is a class representing a Spherocylinder list object; and (4) SpherocylinderListTest class is a JUnit test class which contains one or more test methods for each method in the SpherocylinderList class. *Note that there is no requirement for a class with a main method in this project.*

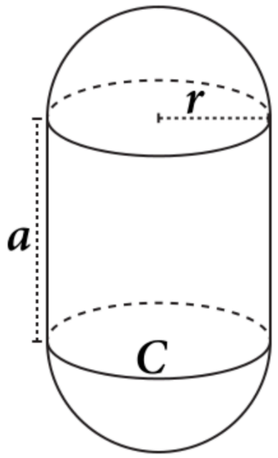
You should create a new folder to hold the files for this project and add your files from Part 2 (Spherocylinder.java file and SpherocylinderTest.java). You should create a new jGRASP project for Part 3 and add Spherocylinder.java file and SpherocylinderTest.java to the project; you should see the two files in their respective categories – Source Files and Test Files. If SpherocylinderTest.java appears in source File category, you should right-click on the file and select "Mark As Test" from the right-click menu. You will then be able to run the test file by clicking the JUnit run button on the Open Projects toolbar. After SpherocylinderList.java and SpherocylinderListTest.java are created as specified below, these should be added to your jGRASP project for Part 3 as well.

If you have successfully completed Spherocylinder.java and SpherocylinderTest.java in Part 2, you should go directly to SpherocylinderList.java on page 5.

- **Spherocylinder.java** (The specification of the Spherocylinder class is repeated below for your convenience from Part 2; there are no modifications for this class in Part 3)

  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)



The variables are abbreviated as follows:
- ***r** is radius*
- ***a** is cylinder height*
- ***C** is Circumference*
- ***SA** is Surface Area*
- ***V** is Volume*

$$C = 2\pi r$$

$$SA = 2\pi r(2r + a)$$

$$V = \pi r^2 \left(\frac{4}{3}r + a\right)$$

Design: The Spherocylinder class implements the Comparable interface for objects of type Spherocylinder and 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.

- 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 `getRadius`: Accepts no parameters and returns a `double` representing the radius field.
- o `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.
- o `getCylinderHeight`: Accepts no parameters and returns a `double` representing the cylinder height field.
- o `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.
- o `circumference`: Accepts no parameters and returns the `double` value for the circumference of the Spherocylinder.
- o `surfaceArea`: Accepts no parameters and returns the `double` value for the total surface area of the Spherocylinder.
- o `volume`: Accepts no parameters and returns the double value for the volume of the Spherocylinder. [*Be sure to avoid integer division in your expression*.]
- o `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 a 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.

```java
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.
- `compareTo`: Accepts a parameter of type Spherocylinder and returns an int as follows: a negative value if `this.volume()` is less than the parameter's volume; a positive value if `this.volume()` is greater than the parameter's volume; zero if the two volumes are essentially equal. *For a hint, see the activity for this module*.

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** (The specification of the SpherocylinderTest class is repeated below for your convenience from Part 2. If you completed this in Part 2, go on the SpherocylinderList class.

  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: 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 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 the grading system 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.

- **SpherocylinderList.java** (<u>new for Part 3</u>) – Consider implementing this file in parallel with its test file, SpherocylinderListTest.java, which is described after this class.

  Requirements: Create a SpherocylinderList 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 SpherocylinderList, total surface area, total volume, average surface area, and average volume for all Spherocylinder objects in the SpherocylinderList. The toString method returns summary information about the list (see below).

  Design: The SpherocylinderList class has <u>three fields</u>, 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 Spherocylinder objects in the array. <u>Note that the number of Spherocylinder objects may be less than the length of the array</u>. These are the only fields (or instance variables) that this class should have.

(2) Constructor: Your SpherocylinderList 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 Spherocylinder objects in the array. These parameters should be used to assign the fields described above (i.e., the instance variables).

(3) Methods: The methods for SpherocylinderList 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 SpherocylinderList. 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 (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. Below is an example of the formatted String returned by the `toString` method, where the name of the list (name field) is `Spherocylinder Test List` and the array of Spherocylinder objects contains the three examples described above (bottom of page 2).

```
----- Summary for Spherocylinder Test List -----
Number of Spherocylinders: 3
Total Surface Area: 184,790.426
Total Volume: 6,996,733.041
Average Surface Area: 61,596.809
Average Volume: 2,332,244.347
```

- `getList`: Returns the array of Spherocylinder objects (the second field above).
- `addSpherocylinder`: Returns nothing but takes three parameters (label, radius, and cylinder height), creates a new Spherocylinder object, and adds it to the SpherocylinderList object. Be sure to increment the int field containing the number of Spherocylinder objects in the SpherocylinderList object.
- `findSpherocylinder`: Takes a label of a Spherocylinder as the String parameter and returns the corresponding Spherocylinder object if found in the SpherocylinderList 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 SpherocylinderList object and deleted; otherwise returns null. Case should be ignored when attempting to match the label. 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.
- `findSpherocylinderWithLargestVolume`: Returns the Spherocylinder with the largest volume; if the list contains no Spherocylinder objects, returns null.

Code and Test: Some of the methods above require that you use a loop to go through the objects in the array. You should 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.

- **SpherocylinderListTest.java** (new for Part 3) – Consider implementing this file in parallel with its source file, SpherocylinderList.java, which is described above this class.

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

  Design: Typically, in each test method, you will need to create an instance of SpherocylinderList, 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 SpherocylinderList. 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 SpherocylinderList class.

  Code and Test: A good strategy would be to begin by writing test methods for those methods in SpherocylinderList 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 SpherocylinderList 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 SpherocylinderList. Be sure to call the SpherocylinderList toString method in one of your test cases so that the grading system will consider the toString method to be "covered" in its coverage analysis. Remember that when a test method fails, 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. You can also step-in to the method being called by the test method and then single-step through it, looking for the error.

Finally, 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 equals method.

The Grading System

When you submit your files (Spherocylinder.java, SpherocylinderTest.java, SpherocylinderList.java, and SpherocylinderListTest.java), the grading system 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. In this project, your test files should provide method, statement, and condition coverage. Each condition in your source file must be exercised both true and false.