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. 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

Specifications

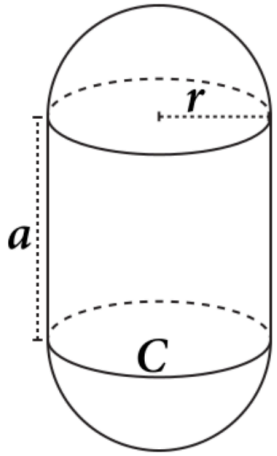
Overview: This project consists of two classes: (1) Spherocylinder is a class representing a Spherocylinder object and (2) SpherocylinderTest class is a JUnit test class which contains one or more test methods for each method in the Spherocylinder class. Note that there is no requirement for a class with a main method in this project.

You should create a new folder to hold the files for this project. After you have created your Spherocylinder.java file, you should create a jGRASP project and add your Spherocylinder.java file; you should see it in the Source Files category of the Project section of the Browse tab. With this project is open, your test file, SpherocylinderTest.java, will be automatically added to the project when it is created; you should see it in the Test Files category. 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.

- **Spherocylinder.java**

  Requirements: Create a Spherocylinder class that stores the label, radius, and cylinder height where both radius and cylinder height are non-negative (>= 0). 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. The Spherocylinder class includes a one static field (or class variable) to track the number of Spherocylinder objects that have been created, as well appropriate static methods to access and reset this field. And finally, this class provides a method that JUnit will use to test Spherocylinder objects for equality as well as a method required by Checkstyle.

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 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 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.

```
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 could be doing in jGRASP 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. 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 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 as new methods are developed. Be sure to call the Spherocylinder `toString` method in one of your test methods and assert something about the return value. If you do not want to use assertEquals, which would require the return value match the expected value exactly, you could use assertTrue and check that the return value contains the expected value. For example, for Spherocylinder example3:

```
Assert.assertTrue(example3.toString().contains("\"Large Example\""));
```

Also, 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.

The Grading System

When you submit Spherocylinder.java and SpherocylinderTest.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 file should provide at least method coverage. That is, each method must be called at least once in a test method.