# Project: Spherocylinder App

## 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 much earlier. The skeleton code assignment is ungraded, but it checks that your classes and methods are named correctly and that methods and parameters are correctly typed. 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. If you are unable to submit via Web-CAT, you should e-mail your files in a zip file to your TA before the deadline.

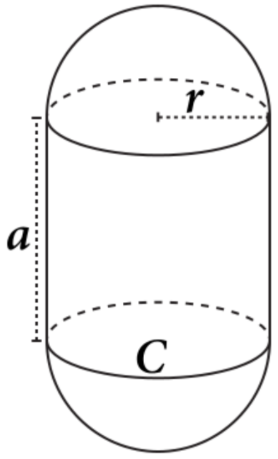
Files to submit to Web-CAT (both files must be submitted together):

- Spherocylinder.java
- SpherocylinderApp.java

## Specifications

**Overview:** You will write a program this week that is composed of two classes: (1) Spherocylinder, which defines Spherocylinder objects, and (2) SpherocylinderApp, which has a main method that reads in data, creates a Spherocylinder object, and then prints the 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.

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

Source for figures and formulas: https://en.wikipedia.org/wiki/Capsule_(geometry)

- **Spherocylinder.java**

  **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.

**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.

(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);` 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.

- `surfaceArea`: Accepts no parameters and returns the `double` value for the total surface area of the Spherocylinder. [*Be sure to avoid integer division in your expression*.]
- `volume`: Accepts no parameters and returns the double value for the volume of the Spherocylinder.
- `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. 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
```

**Code and Test**: As you implement your Spherocylinder 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 Spherocylinder in interactions (e.g., copy/paste the examples on page 2). 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 Spherocylinder 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 Spherocylinder then prints it out. This would be similar to the SpherocylinderApp class you will below, except that in the SpherocylinderApp class you will read in the values and then create and print the object.

- **SpherocylinderApp.java**
  - **Requirements**: Create a SpherocylinderApp class with a `main` method that reads in values for label, radius, and cylinder height. After the values have been read in, `main` creates a Spherocylinder object and then prints a new line and the object.
  - **Design**: The `main` method should prompt the user to enter the label, radius, and cylinder height. After a value is read in for radius, if the value is less than zero, an appropriate message (see examples below) should be printed followed by a *return* from `main`. After a value is read in for cylinder height, if the value is less than zero, an appropriate message

(see examples below) should be printed followed by a *return* from `main`. Assuming that both radius and cylinder height are non-negative, a Spherocylinder object should be created and printed. Below are three examples: (1) the user has entered a negative value for radius, (2) the user has entered a negative value for cylinder height, and (3) the user has entered the values from the first example above for label, radius, and cylinder height. Your program input/output should be **exactly** as follows.

| Line # | Program input/output |
|---|---|
| 1 | `Enter label, radius, and cylinder height for a spherocylinder.` |
| 2 | `   label: One with a bad radius` |
| 3 | `   radius: -10.6` |
| 4 | `Error: radius must be non-negative.` |
| 5 | |

| Line # | Program input/output |
|---|---|
| 1 | `Enter label, radius, and cylinder height for a spherocylinder.` |
| 2 | `   label: One with a bad cylinder height` |
| 3 | `   radius: 12.5` |
| 4 | `   cylinder height: -3.5` |
| 5 | `Error: cylinder height must be non-negative.` |

| Line # | Program input/output |
|---|---|
| 1 | `Enter label, radius, and cylinder height for a spherocylinder.` |
| 2 | `   label: Small Example` |
| 3 | `   radius: 0.5` |
| 4 | `   cylinder height: 0.25` |
| 5 | |
| 6 | `Spherocylinder "Small Example" with radius 0.5 and cylinder height 0.25 has:` |
| 7 | `   circumference = 3.142 units` |
| 8 | `   surface area = 3.927 square units` |
| 9 | `   volume = 0.72 cubic units` |

**Code**: To read user input, your program should use the `nextLine` method of the `Scanner` class. Note that this method returns the input as a `String`. Whenever necessary, you can use the `Double.parseDouble` method to convert the input `String` to a `double`. For example: `Double.parseDouble(s1)` will return the `double` value represented by `String s1`. For the printed lines requesting input for label, radius, and cylinder height, use a tab `"\t"` rather than three spaces. The <u>object</u> can be printed simply by using its variable name; i.e., when the object variable name is evaluated in the print statement, its `toString()` method is automatically called. So printing the object reference variable `myObj` is equivalent to printing the return value of the `myObj.toString()` method call.

**Test**: You should test several sets of data to make sure that your program is working correctly. Although your main method may not use all the methods in Spherocylinder, you should ensure that all of your methods work according to the specification. You can use interactions in jGRASP or you can write another class and main method to exercise the methods. The viewer canvas should also be helpful, especially using the "Basic" viewer and the "toString" viewer for a Spherocylinder object. Web-CAT will test all of the methods specified above for Spherocylinder to determine your project grade.

**General Notes**

1. All input from the keyboard and all output to the screen should done in the `main` method. Only one `Scanner` object on `System.in` should be created and this should be done in the `main` method. All printing (i.e., using the `System.out.print and System.out.println` methods) should be in the `main` method. Hence, none of your methods in the Spherocylinder class should do any input/output (I/O).

2. When a method has a return value, you can ignore the return value if it is no interest in the current context. For example, when `setRadius(3.5)` is invoked, it sets the `radius` field and returns `true`; whereas `setRadius(-3.5)` will return `false` without setting the `radius` field. If the caller knows that the value of `x` is non-negative in the method call `setRadius(x)`, it can be assumed that the field was set appropriately and the return value can be safely ignored. Although caller can always ignore the return value; however, it is important to provide the return value so the method to indicate whether or not the `setRadius` method did in fact set the field.

3. If the caller knows that the value of `x` is non-negative in the method call `setRadius(x)`, it can be assumed that the field was set appropriately and the return value can be ignored safely.

4. Even though your `main` method may not be using the return type of a method, you can ensure that the return type is correct using interactions.