Project 4: Cone App

Due: Skeleton Code (ungraded - checks class name, method names, return type, etc.)
Completed Code – Thursday, September 22, 2016

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your files to the skeleton code assignment until the project due date but should try to do this by Friday, September 16 (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 of the example in the Class Notes or they may be essentially completed files. For an example of skeleton code, see loan.java in the Class Notes for this week (04_Writing_Classes\examples\method_stubs\loan.java). 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. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one-day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline.

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

- Cone.java
- ConeApp.java

Specifications

Overview: You will write a program this week that is composed of two classes: (1) one named Cone that defines right circular Cone objects (base of cone is a circle and the axis is perpendicular to the base) where height and radius are positive, and (2) the other, ConeApp, which has a main method that reads in data, creates a Cone object, and then prints the object.

- **Cone.java**

  Requirements: Create a Cone class that stores the label, height, and radius (height and radius each must be greater than zero). The Cone class also includes methods to set and get each of these fields, as well as methods to calculate the base perimeter, base area, slant height, side area, surface area, and volume of a Cone object, and a method to provide a String value of a Cone object (i.e., a class instance).

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

  (1) **Fields** (instance variables): label of type String, height of type double, and radius of type double. These instance variables should be private so that they are not directly accessible from outside of the Cone class, and these should be the only instance variables in the class.

  (2) **Constructor**: Your Cone class must contain a constructor that accepts three parameters (see types of above) representing the label, height, and radius. Instead of assigning the parameters directly to the fields, the respective set method for each field (described below) should be called. For example, instead of the statement `label = labelIn;` use the statement `setLabel(labelIn);` Below are examples of how the constructor could be used to

create Cone 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.

```
Cone example1 = new Cone("Short Example", 3.0, 4.0);

Cone example2 = new Cone(" Wide Example ", 10.6, 22.1);

Cone example3 = new Cone("Tall Example", 100, 20);
```

(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 Cone are described below. See formulas in Code and Test below.

- 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 `getHeight`: Accepts no parameters and returns a double representing the height field.
- o `setHeight`: Accepts a double parameter and returns a boolean. If the height is greater than zero, sets height field and returns true. Otherwise, the method returns false and the height is not set.
- o `getRadius`: Accepts no parameters and returns a double representing the radius field.
- o `setRadius`: Accepts a double parameter and returns a boolean. If the radius is greater than zero, sets radius field and returns true. Otherwise, the method returns false and the radius is not set.
- o `basePerimeter`: Accepts no parameters and returns the double value for the perimeter of the base circle of the cone calculated using radius.
- o `baseArea`: Accepts no parameters and returns the double value for the base area calculated using radius.
- o `slantHeight`: Accepts no parameters and returns the double value for the slant height calculated using height and radius.
- o `sideArea`: Accepts no parameters and returns the double value for the side area calculated using radius and slant height.
- o `surfaceArea`: Accepts no parameters and returns the double value for the total surface area calculated using the base area and side area.
- o `volume`: Accepts no parameters and returns the double value for the volume calculated using height and radius.
- o `toString`: Returns a String containing the information about the Cone object formatted as shown below, including decimal formatting ("#,##0.0##") for the double values. Newline 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: basePerimeter(), baseArea(), slantHeight(), sideArea(), surfaceArea(), and volume(). Each line should have no leading and 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).

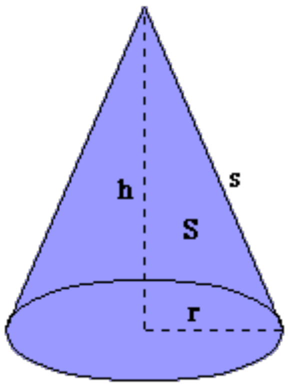
```
"Short Example" is a cone with height = 3.0 units and radius = 4.0 units,
which has base perimeter = 25.133 units, base area = 50.265 square units,
slant height = 5.0 units, side area = 62.832 square units,
surface area = 113.097 square units, and volume = 50.265 cubic units.

"Wide Example" is a cone with height = 10.6 units and radius = 22.1 units,
which has base perimeter = 138.858 units, base area = 1,534.385 square units,
slant height = 24.511 units, side area = 1,701.752 square units,
surface area = 3,236.137 square units, and volume = 5,421.495 cubic units.

"Tall Example" is a cone with height = 100.0 units and radius = 20.0 units,
which has base perimeter = 125.664 units, base area = 1,256.637 square units,
slant height = 101.98 units, side area = 6,407.617 square units,
surface area = 7,664.254 square units, and volume = 41,887.902 cubic units.
```

Code and Test: As you implement your Cone 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 Cone in interactions (see the examples above). 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 Cone 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 Cone then prints is out. This would be similar to the class you will create in Part 2, except that in in Part 2 you will read in the values and then create the object.

The formulas below are provided to assist you in computing return values for the respective cone methods described above (adapted from http://mathforum.org/dr.math/faq/formulas/faq.cone.html). For Pi, `Math.PI` should be used.

height: h
radius of base: r
perimeter of base: P
base area: B
slant height: s
side area: S
total surface area: T
Volume: V

$P = \text{Pi } 2r$
$B = \text{Pi } r^2$
$s = \text{sqrt}(r^2+h^2)$
$S = \text{Pi } rs$
$T = \text{Pi } r(r+s)$
$V = \text{Pi } r^2h/3$



- **ConeApp.java**

  Requirements: Create ConeApp class with a main method that reads in values for label, height, and radius. After the values have been read in, main creates a Cone object and then prints the object.

  Design: The main method should prompt the user to enter the label, height, and radius. After a value is read in for height, if the value is less than or equal to zero, an appropriate message (see examples below) should be printed followed by a *return* from main. Do the same after a value for radius is read in. Assuming that both height and radius are positive, a Cone object should be created and printed. Below are examples where the user has entered non-positive values for height and radius followed by an example using the values from the first example above for label, side, and height. Your program input/output should be **exactly** as follows.

Line # Program input/output

```
Enter label, height, and radius for a cone.
   label: Example with bad height
   height: 0
Error: height must be greater than 0.

```

Line # Program input/output

```
Enter label, height, and radius for a cone.
   label: Example with bad radius
   height: 2
   radius: -1
Error: radius must be greater than 0.

```

Line # Program input/output

```
Enter label, height, and radius for a cone.
   label: Short Example
   height: 3.0
   radius: 4.0
"Short Example" is a cone with height = 3.0 units and radius = 4.0 units,
which has base perimeter = 25.133 units, base area = 50.265 square units,
slant height = 5.0 units, side area = 62.832 square units,
surface area = 113.097 square units, and volume = 50.265 cubic units.

```

Code: Your program should use the nextLine method of the Scanner class to read user input. Note that this method returns the input as a String. If you need to convert the String to a double, 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 height and radius, use a tab "\t" rather than three spaces.

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 Cone, 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 Cone object. Web-CAT will test all of the methods specified above for Cone 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 Cone 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 returns true to let the caller know the radius field was set; whereas setRadius(-3.5) will return false since the radius field is not set. So if the caller knows that x is positive, then the return value of setRadius(x) can safely be ignored since it can be assumed to be true.

3. 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.