

## 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:

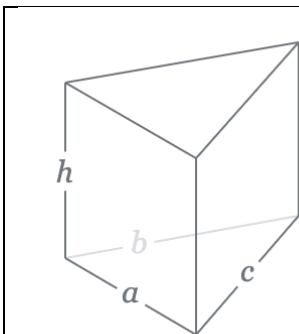
- TriangularPrism.java, TriangularPrismTest.java

## Specifications

**Overview:** This project consists of two classes: (1) TriangularPrism is a class representing a TriangularPrism object and (2) TriangularPrismTest class is a JUnit test class which contains one or more test methods for each method in the TriangularPrism 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 TriangularPrism.java file, you should create a jGRASP project and add your TriangularPrism.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, TriangularPrismTest.java, will be automatically added to the project when it is created; you should see it in the Test Files category. If TriangularPrismTest.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.

A **uniform TriangularPrism** is a TriangularPrism in which the faces (bottom and top) are equilateral triangles ( $a = b = c$ ) with side edge length  $a$ . When lying on a triangle face, the prism has height  $h$ . The sides of the prism are three rectangles of the same size. ([https://en.wikipedia.org/wiki/Triangular\\_prism](https://en.wikipedia.org/wiki/Triangular_prism))



*The variables are abbreviated as follows:*

$a$  is triangle edge length  
 $h$  is height of prism  
 $A_t$  is triangle area  
 $A_r$  is rectangle area  
 $A$  is total surface area  
 $V$  is volume

$$A_t = 0.25\sqrt{3}a^2$$

$$A_r = ah$$

$$A = 2A_t + 3A_r$$

$$V = A_t h$$

- **TriangularPrism.java**

**Requirements:** Create a TriangularPrism class that stores the label, triangle edge, and prism height (edge and height are non-negative,  $\geq 0$ ). The TriangularPrism class also includes methods to set and get each of these fields, as well as methods to calculate the triangle area, rectangle area, surface area, and volume of a TriangularPrism object, and a method to provide a String value that describes a TriangularPrism object. The TriangularPrism class includes a one static field (or class variable) to track the number of TriangularPrism 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 TriangularPrism objects for equality as well as a method required by Checkstyle.

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

- (1) **Fields:** Instance Variables - label of type String, edge of type double, and 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 TriangularPrism 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 TriangularPrism class must contain a public constructor that accepts three parameters (see types of above) representing the label, edge, and 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 TriangularPrism is constructed.

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

```
TriangularPrism ex1 = new TriangularPrism("Small Example", 1.8, 3.25);
```

```
TriangularPrism ex2 = new TriangularPrism(" Medium Example ", 10.7, 25.4);
```

```
TriangularPrism ex3 = new TriangularPrism("Large Example", 45.47, 105.0);
```

- (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 TriangularPrism, 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.
- `getEdge`: Accepts no parameters and returns a `double` representing the edge field.
- `setEdge`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is non-negative, then the parameter is set to the edge field and the method returns `true`. Otherwise, the method returns `false` and the edge field is not set.
- `getHeight`: Accepts no parameters and returns a `double` representing the height field.
- `setHeight`: Takes a `double` parameter and returns a `boolean`. If the `double` parameter is non-negative, then the parameter is set to the height field and the method returns `true`. Otherwise, the method returns `false` and the height field is not set.
- `triangleArea`: Accepts no parameters and returns the `double` value for the area of one of the triangular faces of the prism.
- `rectangleArea`: Accepts no parameters and returns the `double` value for area of one of the rectangle sides of the prism.
- `surfaceArea`: Accepts no parameters and returns the `double` value for the total surface area of the `TriangularPrism`.
- `volume`: Accepts no parameters and returns the `double` value for the volume of the `TriangularPrism`.
- `toString`: Returns a `String` containing the information about the `TriangularPrism` 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: `rectangleArea()`, `triangleArea()`, and `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 `ex1`, `ex2`, and `ex3` respectively are shown below (the blank lines are not part of the `toString` values).

```
TriangularPrism "Small Example" with triangle edge of 1.8 units
and prism height of 3.25 units has:
    triangle area = 1.403 square units
    rectangle area = 5.85 square units
    surface area = 20.356 square units
    volume = 4.56 cubic units
```

```
TriangularPrism "Medium Example" with triangle edge of 10.7 units
and prism height of 25.4 units has:
    triangle area = 49.576 square units
    rectangle area = 271.78 square units
    surface area = 914.491 square units
    volume = 1,259.221 cubic units
```

```
TriangularPrism "Large Example" with triangle edge of 45.47 units
and prism height of 105.0 units has:
  triangle area = 895.263 square units
  rectangle area = 4,774.35 square units
  surface area = 16,113.576 square units
  volume = 94,002.595 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 `TriangularPrism`; otherwise, when cast to a `TriangularPrism`, if it has the same field values as the `TriangularPrism` upon which the method was called, it returns true. Otherwise, it returns false. Note that this `equals` method with parameter type `Object` will be called by the JUnit `Assert.assertEquals` method when two `TriangularPrism` objects are checked for equality.

Below is a version you are free to use.

```
public boolean equals(Object obj) {

    if (!(obj instanceof TriangularPrism)) {
        return false;
    }
    else {
        TriangularPrism d = (TriangularPrism) obj;
        return (label.equalsIgnoreCase(d.getLabel())
            && (Math.abs(edge - d.getEdge()) < .000001)
            && (Math.abs(height - d.getHeight()) < .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 `TriangularPrism` class, you should compile it and then create test methods as described below for the `TriangularPrismTest` class.

- **TriangularPrismTest.java**

**Requirements:** Create a `TriangularPrismTest` class that contains a set of *test* methods to test each of the methods in `TriangularPrism`.

**Design:** Typically, in each test method, you will need to create an instance of `TriangularPrism`, 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 `TriangularPrism`, 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 the methods in your `TriangularPrism` 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 `TriangularPrism` 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 `TriangularPrism` method being tested. 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 `TriangularPrism 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 `TriangularPrism example3`:

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

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 `TriangularPrism.java` and `TriangularPrismTest.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.