CS1110   Lab 03. Writing and testing classes   Fall 2008
Playing with static variables

Name _____   NetId _____

Section day/time _____   Section instructor _____

In this lab, we give you practice with writing a class and using the JUnit tester to test its methods.

Task 1. Download. Download file ThreeDimPoint.java from here:

http://www.cs.cornell.edu/courses/cs100j/2008sp/handouts/labs/lab03program/ThreeDimPoint.java.

Save it in a new directory. Each instance of class `ThreeDimPoint` is a three-dimensional point (x, y, z). When you are done with this lab, you may want to copy this file —and a JUnit file that you will construct— to your memory stick —or email the files to yourself.

Task 2. Examine the beginning of class ThreeDimPoint. It has three variables, `x`, `y`, and `z`, a constructor, and three getter functions. Understand what those four methods are supposed to do FROM THEIR SPECIFICATIONS. Don't look at the bodies yet.

Task 3. Create a test class and test the constructor and three getter functions. We'll tell you now: the methods have errors in them. DO NOT LOOK FOR THEM AND FIX THEM IN THE BEGINNING. Instead, create a JUnit test class and test the four methods; use the tests to discover bugs and then fix the bugs. Continue testing until your test cases don't give error messages. Here's what to do:

1. In DrJava, select menu `File` item `New JUnit Test Case...`. This will create a second file; call it `DimTester.java`. Save it.
2. Change the name of method `testX` in class `DimTester` to `testGetters`.
3. In the body of method `testGetters`, write Java statements that:

   (a) Create an instance of `ThreeDimPoint` and save its name in a variable. Note that the constructor call in the new expression requires three arguments --the x, y, and z coordinates of the point. (b) Test the three getter methods. Refer to the slides for the lecture of 5 February for help. Remember, method `assertEquals(x,y)` stops all testing, with an error message, if `x` not equal `y`.

4. Compile the two classes, by clicking your mouse on button `Compile All`.
5. Click button `Test`.
6. If an error message results, study the message and the constructor and the relevant getter function to determine what is wrong and fix the program.

Repeat steps 4, 5, and 6 until no error message results.

Congratulations! You have used a JUnit tester to debug your first program.

Task 4. Test function hasAZero. This function is supposed to return true if at least one of the x-coordinate, y-coordinate, and z-coordinate is 0. If none of them are zero, it returns false. That's what the specification says. Do not make any changes to the function before testing it. In class `DimTester`, make up another test

function, `testHasAZero`, that will test function `hasAZero`. A possible test case is any set of values (x, y, z). Think about it: how many test cases do you need in order to be sure that the method is correct? Perhaps 6 or 7 or 8? Below, write down a list of test cases that you think will suffice to provide some assurance that the function is correct:

In test function `testHasAZero`, implement all the test cases that you think you need. The test function may have to create more than one instance of `ThreeDimPoint` in order to implement all your test cases.

Now run the test program (click on button `Test`). If you get error messages, look at the program and fix it. Continue testing and debugging in this fashion until running the test program does not produce an error.

Test 5. Trying to make a method static. Consider the line from point $(0, 0, 0)$ to point (x, y, x). The length of this line is the square root of $x*x + y*y + z*z$. In Java, function call `Math.sqrt(n)` finds the square root of `n`, as you can see from method `length` in class `ThreeDimPoint`. Method length computes the length of the line given by the instance in which it appears. Try it out using these two statements in the interactions pane:

> d= **new** ThreeDimPoint(3, 4, 5);
> d.length()   // Write the answer here:

Now do the same thing using this single statement.

> (**new** ThreeDimPoint(3,4,5)).length()   // Write the answer here:

The second way, just above, contains a new-expression that is NOT assigned to a variable. This is legal. There is nothing wrong with it. The new folder (object) is created and stored in file-drawer `ThreeDimPoint`, and the name of the folder is the result of the new-expression; then, function `length()` of that folder is called.

Now, make the function static by placing the word `static` right after `public` and compile. What happens? Write here the error message that is printed:

What is the problem? Explain in your own words what the problem is:

Then remove the word `static` and compile again.

Test 6. Making a method static. Now consider function `length1`. It has the heading

> **public double** length1(ThreeDimPoint c)

so when one calls it, one has to give it an argument that is the name of a `ThreeDimPoint` folder. We'll show you this in a minute.

Make this method `static`, by placing keyword `static` after `public`, and compile. It works? Can you explain why?

Yes, the method does not refer directly to any fields of the folder in which the method occurs, so it does not have to be in each folder of the class, so it can be static. There is ONE copy of it, and the copy is in file-drawer `ThreeDimPoint`.

Now do these two lines in the interactions pane, one at a time. The first creates a `ThreeDimPoint` folder and stores its name in folder f. The second gets its length by calling function length in object `f`. The third gets it length by calling static function `ThreeDimPoint.length1`.

> f= **new** ThreeDimPoint(3,4,5);
> f.length()
> ThreeDimPoint.length1(f)

This illustrates how one can use a static method. If a method does not refer to any fields, place it in the file-drawer by making it static. If you still have questions about this, ASK THE TA OR CONSULTANT!

Task 7. Class Math. Class `Math` has lots of static functions that are mathematical in nature. Class `Math`, which is always available (can always be used), is used mainly as a file-drawer for static functions and variables. It has no fields that you can change. It has no methods. Everything in it is static. Here are some function calls and what they mean:

Math.abs(x)	absolute value of x	Math.abs(-4) is 4
Math.sqrt(x)	square root of x	Math.sqrt(25) is 5.0
Math.min(b,c)	minimum of b and c	Math.min(5, 4) is 4
Math.max(b,c)	maximum of b and c	Math.max(–6,4) is 4
Math.floor(x)	largest integer that is not larger than x	Math.floor(–3.2) is –4.0 Math.floor(3.2) is 3.0
Math.ceil(x)	smallest integer that is not smaller than x	Math.ceil(–3.2) is –3.0 Math.ceil(3.2) is 4.0

In addition, static variable `Math.PI` gives the best **double** approximation to pi, the ratio of the diameter of a circle to its circumference. Class `Math` has other static functions. Later, we will tell you where to find their descriptions.

Fill in the following table by placing the calls in the interactions pane and seeing what value DrJava gives:

call	value
Math.min(-7, 4)	
Math.min(Math.min(3,4), 5)	

Math.sqrt(5)	
Math.sqrt(-5)	
Math.sqrt(25)	
Math.floor(–3.7)	
Math.ceil(3.7)	
Math.ceil(–3.7)	
Math.abs(3.7)	
Math.abs(–3.7)	
Math.abs(–3)	
Math.PI	