## JUnit Extra Credit

Write at least 4 JUnit test cases for each the following methods in your Grades class from Lab 2:

* median
* average
* minimum

Therefore, you must have at least 12 JUnit test cases included with your project (i.e., included in the project export that you turn in). Please place all test cases in a separate package in your lab project folder (suggested names for the test package are "test" or "testing"). All test cases must pass and the implementation they are testing (i.e., your Grades class methods) must work correctly (exception handling technique is up to you).

JUnit is already part of your Eclipse installation on the CS workstations. You create a new JUnit test class (or set of test cases) by right-clicking on your project and selecting New->JUnit Test Case and select JUnit 4 as the testing framework. You can read more about using JUnit here: [Unit Testing with JUnit - Tutorial](http://www.vogella.com/tutorials/JUnit/article.html).

### Comments

You must create javadoc comments for your JUnit test class and test cases (methods).

### Rubric

This extra lab is worth 100 points. An incorrect submission will possibly get zero points.

* (90 pts.) The test cases pass, coverage is adequate, and your implementation works correctly.
* (10 pts.) There are javadoc comments: at least one for the class and another for each test case (method).

### Regarding writing test cases

The hard part about writing test cases is knowing which test cases to write (i.e., what do you test?). Some general test case creation guidelines are:

* test parameter extremes (e.g., a few elements and a lot of elements, but keep "lot" reasonable). These should test correctness of implementation.
* test exceptional parameter values (e.g., empty and/or null). These should test the robustness of the implementation.
* test statement coverage. These tests will execute every line of code in the implementation. Loop tests should focus on correctness when a loop iterates 0, 1, and many times.

Note that for simple or even moderately-sized methods, the above guidelines will overlap (e.g., by testing parameter extremes and exceptional values, you also test every statement in the method).

### JUnit test class example using the Chapter 7 GradeBook class

package test;

import static org.junit.Assert.\*;

import org.junit.Test;

import core.GradeBook;

public class TestGradeBook {

/\*\*

\* average method test case using null grade array. The correct behavior is

\* a runtime error.

\*/

@Test

public void testAverageNull() {

GradeBook gb = new GradeBook("test", null);

boolean nullError = false;

try {

gb.getAverage();

} catch (NullPointerException e) {

nullError = true;

}

assertTrue("should be a NullPointerException", nullError);

}

/\*\*

\* average method test case using 10 elements. The correct behavior is the

\* average of the numbers plus or minus roundoff.

\*/

@Test

public void testAverage10Element() {

int[] grades = { 87, 99, 96, 99, 86, 96, 77, 95, 70, 88 };

GradeBook gb = new GradeBook("test", grades);

double average = 0;

for (int grade : grades) {

average += grade;

}

average /= grades.length;

assertEquals("10 element test failed", average, gb.getAverage(),

0.000001);

}

}

To make this example work on the [chapter7 project](http://cs.utsa.edu/~cs3443/notes/chapter07/chapter7.zip):

* When you create the JUnit Test Case, use test as the package and TestGradeBook as the name.
* The GradeBook.java and GradeBookTest.java files need to be moved to a package. Select File->Package, use core as the package name. Then in the Package Explorer view, drag and drop the java files into the core package.
* The try-catch statement is used to handle the runtime error (a NullPointerException). Section 7.4 introduces the try-catch statement, and more details can be found in Chapter 11.