Computer Science 62
Lab 6

Wednesday, March 5, 2014

`JUnit` is a modern testing tool designed by Eric Gamma and Kent Beck (promoter of "Extreme Programming") that is available from www.junit.org. `JUnit` is a tool that reflects the philosophy of test-driven development. Test-driven development involves writing "unit" tests first, and only then writing the actual code. Unit testing is easier and more effective (at least in the early stages of programming) than writing tests of the entire project. The goal is that you never integrate a class into a project until you are convinced that it works properly. Moreover, because we often modify classes, we keep the unit test so that we can rerun them after any changes are made. While anyone can do such testing, `JUnit` is a tool that helps you set up the tests and then runs them all automatically for you – a huge advantage!

You may work in pairs for this lab. Just make sure you haven't worked with the same person more than once.

Creating Test Cases in `JUnit`

We will only provide a brief overview here. You can learn more on your own. `JUnit` is integrated into Eclipse, so it is especially easy to use it when using Eclipse. Later you should read the tutorials that are available from the course web page for details. Because we will be using `JUnit` from within Eclipse, be sure to look at the `JUnit` documentation for Eclipse.

Here are general directions to create a test case for an existing project. Read them through quickly to get a sense as to what is happening. We'll use these instructions for an in lab demo that creates a unit test for your GridTest program from a couple of weeks ago. Then you will use them for the actual lab assignment, which will ask you to add unit tests to your calculator program of this past week.

1. Create a class extending `JUnit`.framework.TestCase. The best way to do this is to select New and then `JUnit` Test Case.

   (a) When the New `JUnit` Test Case window pops up, click on the choice button on top to select New `JUnit` 4 test.

   (b) Type in the name of your test class in the name field. If the class you will be testing already exists, enter it in the Class under Test field (you can also use the browse button to find it if you?d like). Click on the Next button.

   (c) You will get a window showing the methods of the class you will be testing (if you selected one). Click in the checkboxes to have method headers automatically generated for each selected method. Click on Finish and your class will be displayed. (The system may inform you that `JUnit` 4 is not on the build path. If so, select "Add `JUnit` 4 library to the build path" and select OK.)

2. The generated class will import org.`JUnit`.Assert.* and org.`JUnit`.Test. Also all of the methods that have stubs created will have an annotation @Test. The names of all of these methods begin with test, take no parameters and return void. They all start with a default body fail("Not yet implemented"). If you wish to add your own test methods make sure that they follow the same template. You may need some instance variables to be declared and initialized to run your tests. Declare the instance variables as usual, and initialize them in a method with header public void setUp() (which may have been added by the wizard). That method should start with the annotation @Before. (If the compiler objects to that annotation, add import org.`JUnit`.Before;.) If you need to put away resources (e.g., close files) after a test, add a method with no parameters (typically called cleanup) that has an @After tag before its declaration.

3. Write the body for each method. The tests should use methods like assertEquals(a,b) and assertTrue(msg,cond). The setUp method will be run before each test.

Running `JUnit`

To run a test case:

1. Select the test class in the Package Explorer pane on the left side of the Eclipse window.

2. Pull down the menu to the right of the triangle denoting program execution and select run as ... and then select `JUnit` test.

A `JUnit` panel should replace the PackageExplorer on the leftmost panel in the window. If the bar across the top is green, then all tests have succeeded. If one or more tests have failed then the bar will be red and each failed test will be listed. Select each to find out why the test failed.

The `JUnit` panel on the left side of the screen can be replaced by the usual Package Explorer panel if you just click on that tab on the top of the panel.

Continue to refine and add to your tests (just add a method whose name starts with test and code until you are convinced that all of the methods in your class work correctly. At that point you are ready to integrate your class with the other classes in your project and test them. Again, you can write test suites for these using `JUnit`.

The recommended way of working with `JUnit` is always to write your tests before writing the code that uses them. Thus when you start, all of your tests should fail. Your goal is to fix the failed tests one at a time until you get a green bar. If you had a good test suite then your class should be correct. (Though if you had a lousy test suit, you may still have errors!)

Today's Assignment

In today's lab, you will create a test class for the State class from your Calculator project. I have given you a partially written test class. You should include the bodies of methods that test multiplication, subtraction, division, and the clear method. Because you already have a project with your calculator, you need only add the file from /common/cs/cs062/labs/lab06 to your calculator project folder's calc package and refresh eclipse. If you are working with someone else, you may use either of your programs.