Computer Science 507
Software Engineering
The College of Saint Rose
Spring 2013

Lab 6: Unit Testing with JUnit

Due: 6:00 PM, Monday, March 25, 2013

This week, we will work with the basics of the JUnit unit testing system.

You may work alone or in groups of 2 or 3 for these exercises.

JUnit Introduction and Setup

JUnit is a framework to manage test cases for Java classes. The idea is that for a Java class you wish to test, you write an additional class to perform the unit tests.

JUnit is often used with the Eclipse IDE, but can be used on its own or with other IDEs as well. You are welcome to complete this lab with any IDE you wish, but the instructions will assume you are running Java from the command line on `mogul.strose.edu`. If you wish to use JUnit in a different environment, you will need to set it up as appropriate.

To set up the Java environment on mogul to include the JUnit jar files. There is a script you can run to do this:

```
. /home/cs507/junit/junit.bashrc
```

Note: in the above, the period and space are part of the command and are important!

You can either enter this each time you log into mogul and wish to use JUnit, or you can add it to the end of the `.bashrc` file in your home directory.

Please copy the files `SuperSimple.java` and `SuperSimpleTest.java` from `/home/cs507/junit` into a directory for this lab. These are an incredibly simple Java class and its corresponding JUnit test class. Take a look at these files and make sure everything in them makes sense (there's not much).

To compile these, since we are using a bit of a non-standard Java installation on mogul, we need to add the `-1.5` flag to get a version of Java compatible with this version of JUnit. These command should work:

```
javac -1.5 SuperSimple.java
javac -1.5 SuperSimpleTest.java
```

You can then run the unit test with the following:

```
java org.junit.runner.JUnitCore SuperSimpleTest
```

Question 1: What output do you get when you run the "SuperSimple" unit test? (1 point)

Now, change the `isItSimple` method so it no longer "works" (*i.e.*, it doesn't return `true`) and recompile and rerun the test.

Question 2: What output do you get now? (1 point)

This example uses one kind of JUnit assertion, but there are many more.

Question 3: Write a new unit test class that creates a Java `ArrayList`, adds two items to the array list with the `add` method. It should then have assertions to verify that the two items are in the appropriate locations and that the `size` method returns 2. Include this test class and the output when you run the test in your submission. (5 points)

These kinds of tests work for many simple situations.

More Substantial Tests

Thinking about that `ArrayList` example, you will quickly realize that many tests will require some "set up" before and/or "tear down" after each test case. JUnit provides this capability with the `@Before` and `@After` annotations. The test method annotated with `@Before` will run before *each* test method (*i.e.*, those annotated with `@Test`) and the `@After` method will run after each (whether the test succeeds or fails).

Note that you will need an additional `import` for each of these:

```
import org.junit.After;
import org.junit.Before;
```

Question 4: Make a copy of your `ArrayList` test from the previous section and modify it so that the construction of the `ArrayList` and the addition of the two items are in a method annotated with `@Before` (note that you will likely now need to declare your `ArrayList` as an instance variable), and separate out your `ArrayList` tests into 3 separate tests: one to check that the first element is correct, a second to check that the second element is correct, and a third to check that the `size` method returns 2. Include this test class and the output when you run the test in your submission. (5 points)

Question 5: Develop a Java class that implements a non-trivial data structure and/or algorithm, and a JUnit test class that tests the important features. Possibilities include a sorting algorithm, a list or tree data structure, or a class that performs some mathematical calculations that you can check easily. Submit your Java code for both the implementation and JUnit test class. (8 points)

Submission and Grading

To submit the assignment, send your source files and responses to the questions above to *teres-coj@strose.edu* by 6:00 PM, Monday, March 25, 2013. Please include a meaningful subject line (something like "CS507 Lab 6 Submission").

This lab will be graded out of 20 points.