A06, CISC474, 06S: Getting Started with JUnit and Ant

JUnit is a set of Java Classes that allows you to do *automated unit testing* of Java Software. Automated unit testing is a "big deal" in many real-world Java software development shops these days, as some of our guest speakers this semester will tell you.

What you need to do today

Finish up A03, A04, A05, and your code review

Last time, I asked the groups that had finished with A03, A04, A05 to "review" the submission of the "next" and "prev" groups (in the web ring), and post their comments to WebCT. If you haven't done that yet, that is step one for A06. Then you are ready for the next step.

If you like, you can divide up the work among the group members—have part of the group work on that, while the other part moves ahead with the items below.

Get familiar with JUnit (for today, on strauss) and Ant

In your group, gather 1 or 2 programmers to a laptop, and start familiarizing yourself with JUnit.

1. If you have two people, pick one person to "drive", and a second person to look on and watch, and guide the process.

2. First, read over the beginning of the article JUnit Test Infected: Programmers Love Writing Tests which is also available via a local link. Read just far enough that you start getting into code.

3. At that point, stop, back up a bit, and look at my article topics/java/junit/gettingStarted/gettingStartedWithJUnit.txt. This includes information on how to start working with JUnit on strauss (I'll try to get it installed on Porsche soon.) Do the steps needed to configure the strauss account of the person "driving" for JUnit.

4. Work through the "Money" example. The code is in topics/java/junit/gettingStarted/money on the course web site, which is also available directly on the strauss file system as the following directory:

   ```
   /www/htdocs/CIS/474/pconrad/06S/topics/java/junit/gettingStarted/money
   ```

   In that directory, among other things, you'll find a file called build.xml. If you are familiar with using Makefiles under C/C++, then this file is the same kind of thing. However, instead of using Make, it uses a system called Ant. That's a whole other subject, but one you might as well start taking a look at as well. Ant helps you automate the process of compiling your Java web app.

   So you should probably go ahead and read up on Ant, and configure your account for use with Ant also. Resources on Ant are in topics/java/deployment/ant

5. Once you are comfortable with how Junit and Ant work, you are ready for the next step.

By the way, if you haven't read Chapter 2 in Better Faster Lighter Java, tonight would be a good time to do so. It puts all this stuff on JUnit and Ant into a context.

Once you think you've gotten the hang of JUnit, test your understanding:

Start writing test cases for your Model class for the Scheduling app

Say *what?*!!?! We haven't even *written* these classes yet? And you want us to write *test cases?*

Yup. That's exactly what I'm saying.

I know it may seem crazy at first, but that's what "test-driven development" is all about. The idea is that you write the test cases first, and then write the class. This has certain implications:

- Writing the test cases forces you to think about how the class is going to be *used* before you start coding how it *works*.
- In order to compile and run the test cases, you have to at least "stub out" the class... i.e. write methods that usually end up, at least temporarily, having "fake bodies". They do nothing, and return some arbitrary value that just happens to have the correct datatype, e.g. 0, if the function returns an int, or "" if the function returns a String. This is fine because it means that first time you run the test, the test will *fail*. Which is perfectly fine. That way, you know the test can actually catch bad code.
- Then, you do just enough programming to make the test cases actually work. Typically, you do this with one person coding, and a second person looking over his/her shoulder to catch any typos, problems, etc. This is called pair-programming.

What test data should I use?

Start with your test data from your A05 web page.

Stories, unit testing, pair-programming... what is all this stuff? Does this have anything to do with the real world?

If you want to know more, try typing just that set of keywords (i.e. stories, unit testing, pair-programming) into a search engine and see what comes up. Also, check out these ads from a recent issue of a trade magazine.

Your deliverables for A06 (due one week from today, March 08, by 11:55pm).

On your group web site, a link to a directory called A06 that contains Java code. In this directory, include a build.xml file for use with Ant that compiles your code, and runs tests on it. It's ok if for now the tests fail (i.e. because the body of your methods is all in stub form.)

Next steps beyond that

Upcoming assignments will turn this all into a working web app—here's a preview:

- A07: Make a new directory (a copy of A06, once everything compiles and the tests all fail), in which you add code to the methods so that they all pass. (The reason we are having you do this in a separate directory is so that we can grade A06 while you work on A07.)
- A08: Once all your tests pass, then given that you already have a "view" (i.e. HTML code for the input, and HTML code for the output that could easily be turned into a JSP), and a "model" (your thoroughly tested Java classes from A08), writing a controller to stitch it all together into a working useful webapp should be a piece of cake.