**Software Testing Notebook Worksheet #2**
**Structural Testing**
**Due: Friday, November 7, 2003**

Name: _____

Lab Time: _____

Grade: _____/50

On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work.

Signature: _____

## jcoverage

This week you will be using a tool called `jcoverage` to create a structural test set. A structural test set is complete when it achieves complete coverage of the program's control flow graph. `jcoverage` records coverage information for test cases executed by your program under test.

To get started using `jcoverage`, we need to configure your environment.

First, you need to add the following directory to your command path:

```
~csci3308/ant/bin
```

.

Second, you need to add the following lines to your .cshrc file:

```
setenv JAVA_HOME /tools/cs/j2sdk

setenv ANT_HOME ~csci3308/ant
```

Third, you need to add the j2sdk package to the list of packages in your .cshrc file.

Fourth, logout and login and verify that your new environment reflects these changes. You should see the two new environment variables and the new directory in your PATH environment variable. The output of the command `which javac` should be `/tools/cs/j2sdk/bin/javac` and the output of the command `which java` should be `/tools/cs/j2sdk/bin/java`.

**ezpay source code**

Copy `~csci3308/src/ezpay2.tar` to your `src` directory. This tar file is designed to be unpacked into your existing `ezpay` directory:

```
tar xvf ezpay2.tar
```

Your `ezpay` directory now contains the following new items:

- A `build.xml` file
- An `ezpay` shell script
- A `run-tests` shell script
- A `src` directory
- A `test/ts2/tc01` directory

Take a look at the `src` directory and examine the source code for ezpay. You can also take a look at the `run-tests` script. This script can serve as a basis for writing a script to automate your test cases for this week. It is slightly different from the script shown in worksheet 1, due to a requirement imposed by the jcoverage tool (explained in more detail below). Feel free to change the `run-tests` script as much as you like.

IMPORTANT: You can look at the `build.xml` file and the `ezpay` script if you want, but please do not change them.

**Building the ezpay program**

The `build.xml` script is designed to make it easy to build the `ezpay` program and to make it easy to generate `jcoverage` reports. To build ezpay, type the following:

```
ant
```

...and that's it! (If you are curious, `ant` is a build management system similar to `make`. The `build.xml` file contains all of the instructions needed to build `ezpay` and integrate it with the `jcoverage` tool.) Note that the build process created a file called `jcoverage.ser` that keeps track of the lines of code that have been executed by your test cases. You are now ready to generate your first `jcoverage` report. (Hint: since we haven't run any test cases yet, our report is not going to have much to show!) Type the following:

```
ant report
```

This command has created a `report` directory. You can view the report by loading `report/index.html` into a Web browser.

**Running a Test Case**

Now, lets execute a test case and update our report. Type:

```
./run-tests
```

You should see a message that `tc01` of `ts2` passed. Now, type:

```
ant report
```

and reload the coverage report in your Web browser. You will see that this single test case has executed a good chunk of the EZPay program. (Running the `jcoverage`-instrumented version of `ezpay` caused `ezpay` to update the `jcoverage.ser` file with information about which lines of code the test case executed.

NOTE: It is important that you invoke `ezpay` in the directory containing the `jcoverage.ser` file. Otherwise, `ezpay` will not be able to update the `jcoverage.ser` file and you will therefore not make progress in trying to achieve complete coverage of `ezpay`'s source code. This is why the `run-tests` script for this week is different from last week's script in two major ways. First, we are not making use of the architecture-independent build directory that we created last week. Second, we are not cd'ing into the test case directory. Instead we are executing ezpay in the current directory and reaching down into the testcase directory to access the input and output.expected files.

You are now ready to start creating your own test cases for `ts2`. Your goal is to achieve at least 99% line and branch coverage for `ezpay`. Feel free to reuse test cases from `ts1`. I recommend adding one test case at a time, running ezpay on that test case and updating the coverage report. This will allow you to track your coverage results incrementally. The report will help guide your testing process since you will naturally start to write test cases to hit unvisted lines in the program.

**What to Turn in**

This worksheet is worth 50 points. You must turn in the following:

- A list of the test cases you had to add to `ts2` in order to achieve 99% line and branch coverage. This description should include the test case's number, its input file, and its documentation file. If you could not achieve 99% line and branch coverage, then be sure to include a paragraph or two explaining why it was not possible. (25 points).

- A tar file that contains `ts2` and your final coverage report. Your tar file should be named `lastname-ts2.tar`. You can create this tar file with the command:

  ```
  tar cvf lastname-ts2.tar report test/ts2
  ```

  Remember to use your own last name in the command above. (10 points).

- A paragraph or two describing your thoughts on using the `jcoverage` tool. (5 points).

- The output of your structural test run, including a summary, e.g. how many test cases passed and how many failed, and the details, e.g. which test cases passed and which ones failed. Remember that each test case should produce output that can be compared with the expected output to indicate whether or not the program passed or failed the test case according to the ezpay specification. Note: this output should match the format requested in section 4 of the functional test plan developed for worksheet 1. (10 points).