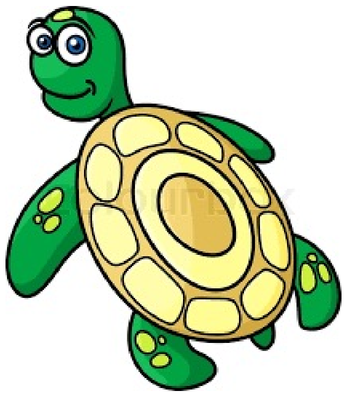
# CPSC 331 — Java Exercise #6

# ***What Do You Mean,*** My Code isn't Flawless?!?!?!?

# Part Two: Using Shell Scripts to Automate Testing



## About This Exercise

The previous exercise introduced the use of the `JUnit` test harness to test methods provided by classes. Unfortunately `JUnit` does not provide a straightforward way to test programs that are to be executed from the command line.

In this exercise you will learn that a Unix command shell script can be used for this — so you should be using a Macintosh, or a Unix or Linux server to carry out this exercise, rather than Windows.

A shell script can also be used to automate the use of other tests. In particular, it is possible to write a script resembling the one included with this exercise so that it simplifies the use of `JUnit` for testing.

***Note:*** When downloading text files, file conversion problems sometimes arise because Windows and UNIX use different formats. This can generally be fixed by using the `dos2unix` command, using the name of the downloaded file to be fixed as the argument for this command.

## 1 Executing a Shell Script

1. Like the exercise before it, this exercise concerns tests of the program `Fibonacci3b.java`. Please download and compile this program if you have not already done so.

2. Download the file `test_Fibonacci3b_main.sh`. Make sure that this "executable" by executing the command

   ```
   chmod u+x test_Fibonacci3b_main.sh
   ```

   in the directory where this file has been stored, and then execute the command

   ```
   ./test_Fibonacci3b_main.sh
   ```

   in this directory. This should produce a rather long screen output that is identical to the contents of the file `test_main_output.txt`, which is also now available.

   If you receive an error message, instead, then execute the command

   ```
   dos2unix test_Fibonacci3b_main.sh
   ```

   and try again.

3. Once you have got that to work, list the contents of the `test_Fibonacci3b_main.sh` file. It begins with the line

   ```
   #!/bin/bash
   ```

   which indicates that this is a Unix shell script that makes use of the `bash` shell. Various other shells are available, but this is the shell used, by default, by undergraduate students in the Department of Computer Science.

   The file continues with three lines, using the `echo` command, to display a bit of white space and a title explaining that dynamic tests of the main method in the Java class `cpsc331.tutorials.Fibonacci3b` are now being performed.

   The file continues with the following lines.

   ```
   echo 'Executing java cpsc331.tutorials.Fibonacci3b'
   echo ' '
   echo 'Expected Output:'
   echo 'No input found!  A nonnegative integer is required.'
   echo ' '
   echo 'Actual Output:'
   ```

```
java cpsc331.tutorials.Fibonacci3b
echo ' '
echo 'Rationale:  No command-line inputs have been provided.'
echo ' '
echo ' '
```

Most of this just consists of a sequence of applications of the `echo` command to display text, describing the test being performed, the expected output, and the rationale for the test — which is an execution that fails to include the required integer input. The seventh line, here, is different:

```
java cpsc331.tutorials.Fibonacci3b
```

— the program being tested really *is* being executed, without any inputs, in order to allow a visual comparison of the expected output for this test and what is generated when it is carried out.

The file continues with several more sequences of lines that resemble the following, but implement different tests (in much the same way):

- The program is executed from the command line with a pair of inputs — $1$ and $3$ — instead of a single one. An error message is used to explain that only a single input is allowed.
- The program is executed from the command line with a single input, "`zero`". An error message is used to explain that the input must be an integer (rather than, in this case, a string that is not readily converted into an integer).
- The program is executed from the command line with the single input, "$-1$". An error message is used to explain that the input must be a non-negative integer.

Remaining tests include non-negative integers, and the corresponding Fibonacci number should be returned as output. A message explaining this is given, using the "`echo`" command, so that "rationales" do not need to be given for the remaining tests.

In particular, the same inputs are considered as in the tests implemented for the `fibLoop` method, using `JUnit`, and considered in the previous Java development exercise: The inputs $0$, $1$, $2$, $3$, $4$, $5$, $10$, $46$ and $47$ are considered.

Please note that the "expected output" is the same as the "actual output" in every case: All tests have been passed.

4. Next, please download the file `test_Fibonacci3b.sh`, make that this is executable, and run it — essentially as you did for the "`test_Fibonacci3b_main.sh`" file, above.

   An examination of either the instructions in this file, or the output its execution generates, should confirm that an execution of this shell script includes both an execution of the “`test_Fibonacci3b_main.sh`” script, and an execution of the tests that were implemented using `JUnit` and considered during the previous development exercise.

Yes: One shell script can be executed by another, and `JUnit` tests can be executed too. It is, therefore, possible to write shell scripts carrying out tests for individual programs, and then write one or more “master scripts” that carry out all the tests needed for a large development project.

5. The shell scripts considered in this exercise have been extremely basic — and the output generated for successful tests have probably been considerably longer than is necessary or helpful. It is possible to write considerably more sophisticated and powerful scripts than the ones shown here.

   Numerous `bash` tutorials are available, and it might be best if you search the internet to find one that suites you best. That noted, the tutorial at

   `http://mywiki.wooledge.org/BashGuide`

   seems to be well-regarded.