JUnit Testing using the Command line

Preface

An important part of writing code is to test it for correctness. One way to avoid bug-ridden code is to write **JUnit tests** which you will learn how to do in a future lab. In the meantime, we have provided you a sample JUnit test to test a `Calculator` object for correctness. This is a short guide on how to run JUnit tests we provide you using the command line. This will be particularly useful if you decide not to use an IDE for upcoming projects.

Requirements

What to Download

Every time you want to run JUnit tests using the command line, you will need the associated `.jar` files which are essentially a bunch of compiled and compressed java files. We have provided you the necessary `.jar` files to run JUnit tests here; you should have three `.jar`s in total.

Organization

Since running JUnit tests using the command line can get quite messy, we recommend you set up your lab/project file structure as follows:

1. Create a directory to host your lab/project files, e.g. `project1`
2. Under that directory, create a `lib` directory. This is where all of your `.jar` files will go.

How to run (with an Example)

Showing you how to run JUnit using the command line is easier with a running example As mentioned above, we have provided you a sample project for you to play around with. Once you have downloaded the sample project, open your command line and `cd` into the project directory; you should see a `lib` directory, a `Calculator.java` file, anda `TestCalculator.java` file which is our JUnit test.

Refer to the Command Line Primer if you do not know how to use the command line.

Compiling your code 2.0

In lab 1, we introduced one way of compiling java code:

```
javac [class name].java [another class].java ...
```

JUnit tests are like any other java files, so we also have to compile them. Unfortunately, using the command above won't do the trick since JUnit tests have some external dependencies. In other words, Java doesn't automatically provide us the required files to just run the tests. In order to resolve those dependencies and compile the tests, you will have to use the following command:

```
javac -cp .:lib/* [JUnit].java
```

Example: `javac -cp .:lib/* TestCalculator.java`

Caution: You may have to compile your non-test files first!

Windows: If you are using Windows, run the following:

```
javac -cp .;lib/* TestCalculator.java
```

Notice the replacement of the colon before `lib` with a semi-colon.

Exploration: As always, check the man pages*if you are curious on the capabilities of `javac` Otherwise, you will notice a few new things in our compiling command. `-cp .:lib/*` allows you to add external libraries during compilation so that java can link up your code with any dependencies when you run your program.

Running the unit tests

After you have compiled the test files, the next step is to give the JUnit test a run. Running a junit test requires a bit more elbow grease:

```
java -cp .:lib/* org.junit.runner.JUnitCore [name of test]
```

Example: `java -cp .:lib/* org.junit.runner.JUnitCore TestCalculator`

Windows: If you are using Windows, run the following:

```
java -cp .;lib/* org.junit.runner.JUnitCore TestCalculator
```

Notice the replacement of the colon before `lib` with a semi-colon.

*Unless you are on Windows, in this case, Google is your friend

If all of the tests pass, then you will see `SUCCESS` printed to the terminal.

If one or more fail, you will see a bunch of text printed to your terminal. Focus on the text that is not indented, it tells you the name of the test that failed, what your test expected, and what the test actually got. This will allow you to determine which methods are not working the way they are supposed to.

That's really all there is to it! Why don't you go ahead and fix our `Calculator` class and retest it, make sure all of the tests pass this time!

> **Caution:** Remember to compile every java file that you changed

> **Exploration:** Feel free to add more to our `Calculator` class. See if you can figure out how unit tests are written and write some yourself! This is a vital skill to develop if you plan on developing software in the future.