COMP 210 - Lab 1 - Hello Java

Spring 2014

For your first assignments you will:

1. Write and run a Java Program in which you:
   (a) Do some basic terminal I/O.
   (b) Write some basic control flow statements in Java
   (c) Write and run some basic JUnit Tests.
2. Start to learn how to navigate and read Oracle's Java documentation

This should be a pretty low stress stuff so feel free to explore basic Java and have some fun with it. The important thing is that you get comfortable with putting some simple Java together.

The Assignments

For lab you need to complete parts 1 and 2. Use *handin* to submit your *source code folder* as assignment *lab1*[1]. For homework complete parts 3 and 4 and resubmit your source code[2] as *hwk1*.

[1] Seriously. Source code only. I'll get really grumpy if I get full project folders...
[2] with the stuff from parts 3 and 4 obviously

Part 1: Hello World

The first thing you should do is fire up a "Hello World" program like we did in class.[3]. The real value of such a simple program like "Hello World" isn't that you really learn much about the language you're using, it's that you learn the basics of how to use the tools you need to write code in that language. Don't work on the next stuff unless your "Hello World" is working and running and you can start and build a Java project in Eclipse.

Now, your first real programming task will be to modify the *main* method of your "Hello World" program so that it averages five numbers (doubles) given to us by the user. Assuming the user always inputs 1.1, then the I/O for such a program should look like this:

```
Enter number 1: 1.1
Enter number 2: 1.1
Enter number 3: 1.1
Enter number 4: 1.1
Enter number 5: 1.1
Average: 1.1
```

That I/O sequence for getting the user input seems really repetitive, right?[4] Once you can duplicate the above I/O[5], you're ready to move on.

[3] We went over the steps to get started coding in Java in class, but here's a reminder:

1. If you're not reusing an existing project, start a new Java Project
2. All source files (Classes, Interfaces, etc.) should go in a *package*, so create one or more packages. For this lab you'll need a *lab1*, or better named, package.
3. Create your source files in the appropriate package.

Be sure to pay attention to the Eclipse prompts about files names, etc.

[4] Use a loop to get it done!
[5] and repeat with different numbers of course

Part 2: From the Horses Mouth

Part of what we'll practice in this class is reading the official documentation for programming languages and libraries as opposed to text book descriptions. The style of the two forms of writing is generally very different. Oracle is the current maintainer of Java and they provide a very robust set of specifications and tutorials for Java programmers. The **A**pplication **P**rogramming **I**nterface specification for Java 7 can be found here [6].

[6] API

```
http://docs.oracle.com/javase/7/docs/api/overview-summary.html
```

We talked a bit about how this document is organized in class. So, now you need to go find the documentation for the *java.util.Formatter* class. As you can see, Formatter is found within the *java.util* package. The Formatter class documents the syntax for the format strings used by the *printf* which allows for classic formatted output and the kinds of things we'd use I/O manipulators for in C++.

After you get a sense for how formatted output with printf works. Modify your program so that all output is done with a *printf* statement and the last line is aligned with the input and so that the average is reported to exactly four decimal places. The new I/O should look like this:

```
Enter number 1: 1.1
Enter number 2: 1.1
Enter number 3: 1.1
Enter number 4: 1.1
Enter number 5: 1.1
       Average: 1.1000
```

If some of your inputs are greather than or equal to 10, then you might have trouble lining up the decimal place. Don't worry about that for now[7]. Once you have *printf* more or less figured out, why don't you go ahead and modify the program to ask the user how many numbers they want to enter and then get that many numbers and average them[8].

[7] It's a fun little puzzle for later though!

[8] should be review. snoozeville.

Part 3: Unit Testing with JUnit

Unit testing is an important part of our program design and development process. The *JUnit* framework provides the same unit testing framework as GTest did for C++. The syntax and usage is, however, different. The basics are very well document here:

`http://www.vogella.com/tutorials/JUnit/article.html`

Most of what you'll need today is covered in section three of Lars' guide. For this lab you need to write and run some unit tests that test silly things like the fact that $2 + 3$ is equal to 5. Put all your tests in a document separate from your code from parts 1 and 2. What you test is not important, however, *you should write at least one functional test and one behavioral test and a total of 10 tests using at least three different assertions.*[9]

[9] What is most important is that you get some JUnit testing code running.

Part 4: Break It and Play

Programmers spend more time fixing code than the do writing working code. Break your code from parts 1 and 2 by introducing some errors[10] and seeing what kind of error the compiler gives you. You should see a complete call stack with the top most item being the place where the error occurred. You should also note that clicking line numbers in the error message should take you to that exact line. If you want to poke around with some more Java code, then check out the *Trails Covering the Basics* found here:

[10] One at a time

`http://docs.oracle.com/javase/tutorial/`.

You should also play around with *printf* some more. What's required is that you *find a mix of five syntax errors and other basic Java features*[11] to play with and then, as comments in your code, describe the bug or feature you played with and why it was a worthwhile thing to explore[12]. If you happen to introduce errors while working on part 1, you're free to document those those for this part. If you didn't read this whole thing before getting started and have already completed the first 3 parts without document errors you encountered along the way, then shame on you for not reading the whole document before getting started.

[11] I really mean basic

[12] convince me you made a good choice