CMPS 11
Intermediate Programming
Lab Assignment 1

The purpose of this assignment is threefold: get a basic introduction to the Unix operating system, to learn how to create and edit text files using either the Vi or Emacs text editors, and to learn to compile and run a java program.

Preparation
Before attempting this assignment, begin reading one of the Unix tutorials linked on the course website. You need not complete the tutorial, but find one that you like, and bookmark it for future reference. Vi and Emacs are two very popular text editors available on all Unix systems. Start reading one of the Vi tutorials, or the Emacs tutorial that are linked on the course website.

Unix
Logon to your Unix Timeshare account (unix.ucsc.edu). If you don't know how to do this, you'll need to go to a lab and get some help. For those running Windows, you need to download a program called PuTTy (Google that name to get a download link) then use it to log on to the Timeshare. If you are running Mac OSX, you need to open a terminal window and do the command:

```
% ssh cruzid@unix.ucsc.edu
```

where "cruzid" is your CruzID, then respond with your Blue password. The Unix command line prompt will be represented here as `%`, although it may look different in your session. From within your home directory, use the `mkdir` command to create a directory called `cs11`, in which you will place all your work for this class. Type `ls` to list the contents of your home directory. You will see the new `cs11` directory. Make `cs11` your current working directory by typing `cd cs11` at the command prompt.

```
% mkdir cs11
% ls
% cd cs11
```

Remember that you can learn about any Unix command by typing `man` at the command prompt. Try:

```
% man mkdir
% man ls
% man cd
% man man
```

Man pages are notorious for being cryptic and even impenetrable, especially for beginners. Typically they assume a great deal of background knowledge. Nevertheless, you must get used to reading them since they are an invaluable resource. Use the man pages in conjunction with the tutorial to build up your vocabulary of Unix commands. Also try using Google to find Unix commands. For instance a Google search on the phrase "Unix copy" brings up a reference to the `cp` command. Research the following Unix commands, either through the tutorial, or man pages, or Google: `man`, `ls`, `pwd`, `cd`, `mkdir`, `more`, `less`, `cp`, `cat`, `rm`, `rmdir`, `mv`, `echo`, `date`, `time`, `alias`, `history`. You can also try just typing the command and see what happens.

Editors
Using either Vi or Emacs, create a text file in your `cs11` directory called `HelloWorld.java` containing the following lines. (Note this file can be found on the course website under the Examples link.)

1

```
/*
*   HelloWorld.java
*/
class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello, world!");
    }
}
```

This is a java *source file*. Type `more HelloWorld.java` at the command prompt to view the contents of the file.

Java
In order to run the program we must first compile it. A *compiler* is a program which translates *source code* into *executable code*, which is what the computer understands. To compile the above program type

```
% javac HelloWorld.java
```

You should see the unix prompt (`%`) disappear for a few seconds, while it works, then reappear. List the contents of `cs11` again to see the new file `HelloWorld.class`. This is a java *object file*, also called an *executable file*. You can now run the program by typing

```
% java HelloWorld
```

This command should cause the words

```
Hello, world!
```

to be printed to the screen, followed by a new command prompt on the command line. We will have a lot more to say about the proper use and syntax of the java programming language, but for now just note that what is printed to the screen is exactly what appears between quotes in the line

```
        System.out.println("Hello, world!");
```

in the source file `HelloWorld.java`. Also note that everything that appears between `/*` and `*/` constitutes a comment and is ignored by the compiler. Every program you write in this class must begin with a comment block of the following form.

```
/*  filename
 *   your Name
 *   your CruzID
 *   the assignment name (like lab1 or pa1)
 *   a very short description of what the program does
 */
```

Open up your editor and change the comment block in HelloWorld.java to conform to the above format. Also change the body of the program so that it prints out

```
Hello, my name is Foo Bar.
```
Where `Foo Bar` is your name. Compile the new program and run it. If it does not compile, i.e. if you get error messages when you run `javac`, look for some stray character that you might have inserted into the file inadvertently, or perhaps a required character you failed to type.

2

Now create a new text file called HelloWorld2.java containing the lines

```
/*  HelloWorld2.java
 *  your Name
 *  your CruzID
 *  lab1
 *  Prints greeting and some system information.
 */

class HelloWorld2{
   public static void main( String[] args ){
       String os = System.getProperty("os.name");
       String osVer = System.getProperty("os.version");
       String jre = System.getProperty("java.runtime.name");
       String jreVer = System.getProperty("java.runtime.version");
       String jvm = System.getProperty("java.vm.name");
       String jvmVer = System.getProperty("java.vm.version");
       String home = System.getProperty("java.home");
       double freemem = Runtime.getRuntime().freeMemory();
       long time = System.currentTimeMillis();

       System.out.println("Hello, World!");
       System.out.println("Operating system: "+os+" "+osVer);
       System.out.println("Runtime environment: "+jre+" "+jreVer);
       System.out.println("Virtual machine: "+jvm+" "+jvmVer);
       System.out.println("Java home directory: "+home);
       System.out.println("Free memory: "+freemem+" bytes");
       System.out.printf("Time: %tc.%n", time);
   }
}
```

Compile this program and run it. Do

```
% javac HelloWorld2.java
```

Then

```
% java HelloWorld2
```

You will see that it prints something like

```
Hello, World!
Operating system: Linux 2.6.32-504.3.3.el6.x86_64
Runtime environment: OpenJDK Runtime Environment 1.7.0_71-mockbuild_2014_12_18_14_23-b00
Virtual machine: OpenJDK 64-Bit Server VM 24.65-b04
Java home directory: /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64/jre
Free memory: 1.22431496E8 bytes
Time: Sun Jan 04 14:47:24 PST 2015.
```

The exact output you get will depend on the date and time you run it, as well as the platform you are working on. You can see that the extra lines in this version of the program have the effect of collecting and printing certain platform specific information. The meaning of these lines may be discussed in class at some point.

Now edit this file once more so that the comment block contains your name and userid, and alter the greeting so that it prints

```
Hello, my name is Foo Bar
```

where `Foo Bar` is your name, as before. Recompile your program, wring out any typographical errors you might find, then test it.

What to turn in
Read the instructions on the website concerning the use of the submit command. Briefly, the syntax of the submit command is

```
% submit class_name assignment_name  file1 file2 file3 …
```

Here `class_name` will be `cmps011-pt.w15` for all Lab and Programming assignments you turn in. The `assignment_name` in this case is `lab1`. Submit the two souce files `HelloWorld.java` and `HelloWorld2.java`. Thus your submit command will be

```
% submit cmps011-pt.w15 lab1 HelloWorld.java HelloWorld2.java
```

Start early and ask questions in the lab sessions and office hours if anything is unclear.