CS 61B Lab 1
January 27-28, 2004

Goal: This lab will give you practice editing and running Java programs, using Emacs, and accessing online Java documentation.

There are several things you will need to have your TA check off so you can get credit for this lab. You can have them checked off all at once when you are finished, but don't wait until the last minute--another lab may start as soon as yours ends, so ask to be checked off when there is still plenty of time. Checkoffs will not be done outside of lab.

PART I: Getting started (1 point)
---

1. If you are officially registered for a lab--not on the waiting list--you must attend that lab. Otherwise, find a lab that has space.

2. Pick up an account form from your Lab TA.

3. Login to a computer using the temporary password on your account form. Answer the questions the computer asks. Use whatever the registrar uses for your given name and surname. Your registered name is used for grading. If you mess up and need to correct your answers, run "register" from the Unix prompt.

4. Warning: do not, at this time or any point in the future, delete the .login, .cshrc, and .emacs files in your directory. If you would like to customize your account, do so by adding new commands to the end of these files, not replacing the files.

5. The rest of this lab is a TEAM project, with two-person teams. Introduce yourself to the people to your left and right (there is one point associated with knowing the names of two other people in the class). Choose a partner. If you're the odd person out in the lab, go ahead and begin, but if someone new arrives, partner up with them and help them catch up.

PART II: Compile-time errors (1 point)

After logging in, you and your partner may find it easier to work on one computer for a while. However, you should each be separately capable of demonstrating your knowledge of the lab to the TA when it's time to check off.

1. Make a lab1 subdirectory:
```
mkdir lab1
```

2. Change into your lab1 directory:
```
cd lab1
```

3. Copy the three lab files by typing:
```
cp $master/lab/lab1/* .
```

IMPORTANT: You must type the space and the period for this to work!

(Note: master is a shell variable that refers to the cs61b home directory, which is /home/ff/cs61b. Typing $master evaluates that variable. If the cp command above doesn't work, it probably means that something is messed up in your login scripts, so talk to your TA or lab assistant.)

4. Load the program file Names.java into Emacs using C-x C-f. "C-x" is read "control x," and is typed by holding down the control (Ctrl) key while typing "x". You may need to type the full pathname lab1/Names.java, then hit Enter.

Names.java is a simple program for performing various string operations on a name. It almost works, but you need to make some changes so that it compiles and runs correctly.

5. Compile the program within Emacs using:
```
C-x C-e
```

This will build a command line to run the java compiler:
```
javac -g
```

"javac" is the name of the Java compiler. "-g" is a switch to tell the compiler to generate the information the debugger will need. You need to finish the command line by supplying the name of the file to be compiled:
```
javac -g Names.java
```

You may instead type this command in an xterm, assuming you're in (your copy of) the lab1 directory.

6. You will find that javac cannot compile the program. The code contains two syntax errors which you are quite likely to make when you will write programs. Use the control sequence
```
C-x `
```
to jump directly to errors found by javac. (Be sure to use the single opening quote, not the apostrophe.) Figure out what's wrong, correct the errors, save the file using "C-x C-s", and compile the code.

PART III: run-time error (1 point)
---

1. Once javac is able to compile the code, it will create a file called Names.class in the same directory. You can run this program using the java interpreter. In a shell (an xterm or an Emacs buffer called *shell*), type the following:

```
java Names
```

In this case, the ".class" file extension on Names.class is assumed. If you accidentally type "java Names.class", java will look for a file called "Names.class.class".

The program has an error. How can you tell there is something wrong? This type of error, which occurs at run-time, tends to be significantly more difficult to correct than compile-time errors. (It's still somewhat easier than discovering an error in which the program appears to finish without problems, but is doing some computation incorrectly.) The error message may be hard to read at first, but it will allow you to answer certain questions: What is the method (i.e., procedure) that generated an error? What is the line number within the file Names.java?

The error is in one of the methods in the String class, which is a standard Java library. Your textbooks contain some documentation of the Java library, but the best source is the online documentation. To find this, start Netscape by typing:

```
netscape &
```

Double-click on the URL window and type the URL for the class Web page:

```
http://www.cs.berkeley.edu/~jrs/61b
```

From there, you can find a pointer to the "Java 2 standard libraries API." There are two libraries that you will be using early in the semester--the java.lang package and the java.io package. Documentation on String and other standard data types is found in java.lang, so go there and find the String class. There is a lot of information, not all of which will make sense right now, but you should be able to find a description of the problematic String method. Be prepared to show your TA or lab assistant how you found this information and how you figured out from it what was wrong with the program.

When you think you have found the error, correct it, save the file, recompile and execute it to see if the problem is solved.

Aside: You may think that the file produced by javac is named Names.class because the input file is named Names.java. Not so--the name of the .class file is based on the class name IN Names.java. To experiment with this, change the line "class Names {" to "class NAMES {" and recompile using javac. What new file is created?

PART IV: Emacs help (1 point)
---

1. Load the file 'roster.txt' in Emacs. You will see an unsorted list of names, with each line in the form <surname, given names>.

You should use the help facilities of Emacs (apropos and info) to find out how to sort all these names by last name. Then sort the lines of the file. If you are unfamiliar with the help facilities type "C-h ?", and Emacs will guide you through them.

Save the result using "C-x C-s".

If you have the course reader, take a few minutes to look through it now so you know exactly what information you can look up in it. Note that the Emacs Quick Reference Guide is in the very front of the reader. It will be handy to keep your reader by you whenever you're in the lab.

PART V: Change your password (no points)

Change your account's password using the "passwd" command. Note that it may take a while for your password change to be stored in the system files, so you may need to use your old password if you login in the interim.

Check-off

When you're done, show your TA or lab assistant your work, and explain how you accessed the Java String documentation and found the bug. They may ask some questions, such as what you discovered about bugs in the original version of the Names program.

1 point: Show that your Names program works.
1 point: Briefly explain how you discovered and fixed the bugs, show how you accessed the Java String documentation, and/or demonstrate what you did to figure out the bugs, as your TA or lab assistant directs.
1 point: Show your sorted roster.txt.
1 point: Tell the TA the names of the students next to you.

If you are inexperienced with either Unix or Emacs, you are strongly encouraged to attend one of the help sessions on these topics held by the Computer Science Undergraduate Associaton (CSUA).