CS61B, Fall 2004 **Lab #0** **P. N. Hilfinger**

Due: Fri., 3 September 2004, midnight

The intention of these laboratory exercises is simply orientation. You don't have to finish them during the scheduled lab, but you should take advantage of the availability of a TA during the lab. *Don't be afraid to ask questions!* To get credit for this lab, you must complete step 6 at the end (submitting your work).

Reading: *Before* starting the lab, you should complete the reading given in Monday's lecture, and look over the sections of the reader on Emacs and `gjdb`.

1. Get an account form, login, and answer the litany of questions that you'll see in your Console window. It's very important that you do this; otherwise, we'll assume you are not in the course. If you are using your personal named account (as opposed to a class account), you won't need an account form, but you will need to work out how to use the standard CS61B configuration.

2. Find the CS61B class home page with a browser (such as `netscape`).

3. Find out how to run `Emacs`. Create a shell window in Emacs `M-x shell` and use it for Unix commands in the rest of this problem set. Look at the man page for `cp` under Emacs, using the `M-x man` command (this is not the only way to look at on-line manual pages, but it has its points). Figure out how to copy an entire directory of files (that is, given a directory `Foo`, create a new directory `Bar`, such that `Bar/a.cc` is a copy of `Foo/a.cc`, `Bar/Baz/xyzzy` is a copy of `Foo/Baz/xyzzy`, etc.).

4. The directory `$master/hw/lab0` contains files `Lab0.java`, `List.java`, and `Makefile` (plus a skeleton `Addition.java`, which you'll use later). Do the following steps *in Emacs*. (Yes, I know you are familiar with another text editor, but at least I'd like everyone to be familiar with this interface.)

(a) Create a new subdirectory and copy all the files from `$master/hw/lab0` into it. The most convenient place to do this is in a Unix shell window in Emacs, using an appropriate Unix `cp` command, or a combination of a `mkdir` (make directory) and `cp` command. Change the current directory in this shell window to this new directory (the Unix `cd` command).

(b) Load the file `Lab0.java` into an Emacs buffer (the `C-xC-f` or `C-x4C-f` command, or use the menu at the top of your Emacs screen).

(c) Try to compile the files in your directory using the "Compile" entry in the Emacs Tools menu. Try this again with the Emacs `M-x compile` command, and then the `C-xC-e` command.

(d) Use `C-x‘` (control-x backquote) to sequence through the errors the compiler produces.

(e) Correct the compile-time errors in the program and repeat step (c) to get it to compile. I suggest that you make one change at a time and then re-compile immediately thereafter, because a single error often causes many messages, most of which are useless. (No, you have not missed any secret lectures; I really am asking you to correct errors in a language you hardly know at all! In fact, there are even things in this program that aren't in your documentation at all. Yes, it isn't "fair." One of the things you have to learn is how to make progress without perfect knowledge of what's going on. Use your native intelligence; read the program over for clues. The errors are sufficiently simple that you should be able to understand them without detailed knowledge of Java syntax.)

(f) After you have corrected the compile-time errors, run the program by typing `java Lab0` in your shell. The program still won't work properly (it is supposed to read and merge two ordered lists of integers into a single ordered list of integers, and selectively print it).

(g) Within Emacs, use the command `M-x gjdb` and answer the prompt it gives you by typing `Lab0`, so that the whole prompt line reads "`gjdb Lab0`," followed by Return. This should put you into a buffer called `*gud-Lab0*`, running our class Java debugger. Use the `Run` entry in the Debug menu or the `run` command at the prompt to run the program. Type in the following input and see what happens (the italics indicate your input; non-italics are prompts printed by the program):

```
Enter number of elements in first list: 3
Enter 3 values:
1 7 10
Enter number of elements in second list: 5
Enter 5 values:
-2 0 3 6 19
Enter range of elements desired (two integers): 0 10
```

What should it print after this?

(h) Instead, you will probably see a message about a "NullPointerException" that is "uncaught." This means that the program did something wrong. Use the View Caller menu item in the Debug menu (or more conveniently, the `f3` key) and you should find the line in the program that called the method (function) that just failed (the call `L0.toString()`).

(i) At this point, you can print the value of `L0` with the command

```
main[1] p L0
```

(or "`print L0`" if you like verbosity). Actually, you don't type "`main[1]`;" it's a prompt. Try also the commands

```
p/1 L0
p/2 L0
p/3 L0
```

and try to make sense of the output.

(j) Let's try to see what happened. Position the Emacs cursor at the line with the call to `toString` (which, if you've followed directions so far, has an '`=>`' pointing at it). Use the "Set Breakpoint" menu item in the Debug menu or `C-x SPACE` to set a breakpoint on that line. Now go back to the Emacs buffer labeled `*gud-Lab0*`, and run the program again. Enter the same input as before. You should see the program stop at the line you marked. The stopping point is known as a *breakpoint*.

(k) At this point, the `f5` key or the `step` command in the `*gud-Lab0*` buffer should cause your program to stop on the first line of the `toString` function. Now use the command `next`, or the `f6` key to step your program through the `toString` function one statement at a time. You should see it loop.

(l) Try to figure out what is going on by stepping the program (with "`next`") and using the "p" command to print `L`, its head ("`p L.head`"), and its tail as needed (actually, you can print the value of just about any Java expression).

(m) Use your ingenuity (and the debugger) to fix `toString` to run correctly.

(n) The program will print an empty list for the result, which is wrong. Try to figure out why and fix it. Hint: this is just the sort of error you CS61A graduates might have encountered.

5. [This one may take some thinking and reading.] Create a program to read two integers from the terminal (the standard input) and print their sum. Your program (in a file `Addition.java`) should have the following form:

```
import java.util.Scanner;
import static java.lang.System.in;
import static java.lang.System.out;

public class Addition {
    /** Read two integers from System.in and print their results. */
    public static void main(String[] ignored)
    {
        int x, y;
        // REPLACE THE FOLLOWING COMMENTS WITH JAVA CODE THAT DOES
        // THE RIGHT THING.
        // Prompt for and read an integer and put it into x
        // Prompt for and read second integer and put it into y
        // Print x, y, and x+y, followed by a newline

    }
}
```

It should print the prompt "`Enter integer and press Enter:`," read an integer, repeat the process for a second integer, and then print "N+M=S," where M and N are the two inputs and S is their sum. Of course, I've said nothing whatever about how to read input and about how

to convert strings of digits into integers and vice-versa. However, there are enough clues in this assignment for you to figure it out.

6. When you have finished all of the above, use the `submit` command to submit it electronically. This command should be available to you if you are using the usual class setup. If you work from home, transfer your files to the instructional machines first, make sure they work there, and then use `submit`. To use the `submit` command, simply type

```
submit lab0
```

from within a directory that contains all the things you intend to hand in.