Computer Science 62
Lab 8

Wednesday, October 28, 2015

Today's lab has two purposes: it is a continuation of the binary tree experiments from last lab and an introduction to some command-line tools. The Java portion of the exercise is easy except that we will not use Eclipse. You may work with a partner for today's lab.

You will find a pdf on the "Documents and Handouts" page entitled "Our own highly-abbreviated introduction to command-line tools". You should read through the first two sections (terminal and emacs) of this handout before coming to lab.

Getting started

Open a terminal window and create a directory `~/Documents/cs062/lab08`. Note that this may require you to do many intermediary steps that I have not listed (e.g. navigate to the `cs062` directory you created in the beginning of the semester). Next create a directory `bst` inside the `lab08` directory but do not change into this directory.

Open Aquamacs, type the following code, and save it as `Lab08.java` inside the `cs062/lab08/bst` directory.

```
package bst;

public class Lab08{
   public static void main(String[] args){
      System.out.println("Running my first command-line program!");
      System.out.println("The command-line arguments are:");

      for( int i = 0; i < args.length; i++ ){
         System.out.println(i + ": " + args[i]);
      }
   }
}
```

Make sure you understand the code above – what output would you expect if you ran this code? After saving, return to the `Terminal` and compile the file by typing:

```
javac bst/*.java
```

which tells the Java compiler to compile all of the .java files in your current directory (which should just be Lab08.java). If you now type `ls bst` you'll see two files: your original Lab08.java file and the compiled Java byte code Lab08.class.

Finally, to run your program type:

```
java bst.Lab08
```

Notice that the package name is incorporated into the name of the class to execute. Alternatively, you can also run your code as follows:

```
java bst/Lab08
```

which looks more like the native directory structure. You cannot run your code from the `bst` directory by just typing `java Lab08`. This is because the package name is part of the executable name, so for Java the class `Lab08` does not exist. Its "real" name is `bst.Lab08`.

You should see the message printed out without any command-line arguments. If you want to pass in command-line arguments, you can add them after the java command:

```
java bst.Lab08 argument1 argument2 argument3
```

The arguments are determined by whitespace. If you want to have an argument with spaces in it, you need to surround it by quotes. Play with your program a bit until you're comfortable with command-line arguments.

Try making some changes to your program and run it again. Notice that to make a change you change the file in Emacs, save it and then you need to recompile with the javac command (otherwise, you'll still be using the old version).

More Experiments with Tree heights

Now that you have the hang of compiling at the command-line, let's experiment with binary search trees and red-black trees. We did not cover red-black trees in class but we listed them as an example of a balanced binary tree. That is, a red-black tree is a binary tree with extra mechanisms for ensuring the binary tree stays balanced.

In Emacs, delete everything in your `Lab08.java` file and replace it with:

```
package bst;

import structure5.*;

public class Lab08 {
    public static void main(String[] args) {
         BinarySearchTree<Integer> bstree = new BinarySearchTree<Integer>();
         for (int i = 0; i < 128; i++){
             bstree.add(i);
         }

         System.out.println(bstree.height());
     }
}
```

To compile this you'll need to add one extra thing to your compile command:

```
javac -classpath /common/cs/cs062/bailey.jar:.  bst/*.java
```

The `-classpath` flag tells the compiler where to find all of the classes that we need for our program. To run our program we need to tell the compiler where to find:

- The `BinarySearchTree` class
- The `bst.Lab08` class

The class definition for the `BinarySearchTree` class is contained in the structure5 package which is in the Java archive `bailey.jar`. The class definition for the `bst.Lab08` class is in the current directory.

The command-line argument that follows the `-classpath` flag contains two class paths separated by a `:` (colon). The first class path (`/common/cs/cs062/bailey.jar`) is the location of the `BinarySearchTree` class. The second class path (`.`) is the location of the `bst.Lab08` class[1].

When we run our program, we similarly need to specify the classpath:

```
java -classpath /common/cs/cs062/bailey.jar:.  bst.Lab08
```

Now that you have this working, let's play around with different types of trees:

- Construct a `RedBlackSearchTree` alongside the `BinarySearchTree` and compare their heights.
- For a more realistic comparison, calculate the heights of the two kinds of trees with randomly-generated entries (you may find your code from last time useful to look at).

[1] On the command-line, the period (`.`) almost always stands for "this current directory". For example, if you type `ls .` at the command line then you will get a listing of all the files in this current directory

Using Git to manage your code

Git is a distributed version control system for software development, i.e., it allows you to maintain multiple versions of your code and revert to prior states when bugs arise. Additionally, the distributed part means, it provides tools that allow multiple developers to work on a project simultaneously with minimal conflicts. Below we will give you some Git commands that you can follow blindly if you like, but which should be simple and straightforward enough to understand how they work.

Before we can use Git we need to tell it who we are with the following commands:

```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL ADDRESS"
```

with `"YOUR NAME"` and `"YOUR EMAIL ADDRESS"` replaces by your name and (one of) your Pomona College email address. Your name and email address will be associated with changes you make to source code managed by Git.

Let's begin by converting the directory you have been working in to a Git repository. Such a repository (also called a "repo") is called a local repository, as it resides on your local machine. Typically, you would also have a remote repository on a separate server so that the files could be shared among other developers on the project. We will save working with remote repositories for a future lab.

We begin by creating the Git repository itself. This needs to be done once per project.
From the command line, change into your `cs062` directory:

```
cd ~
cd Documents
cd cs062
```

The `~` is a special character that indicates a user's home directory. Alternately, you could have just typed `cd ~/Documents/cs062/lab08` which has the same effect as the commands above.

Now, we need to tell Git that we would like to use this directory as a repository by typing `git init`, where `init` stands for initialization. Next, we need to tell Git which of our files it should keep track of by typing `git add bst/*.java`, which tells Git to add all of our source files. Finally, we tell Git we would like to save a version of our project, this is called committing, with the command `git commit -am "My first commit!"`. Now our repository is fully setup and ready to use.

Now, make some changes to the Lab08.java file. For example, you can change the number of values to insert into the tree from 128 to 256. Save your changes, compile, and run your code as we discussed above.

Your changes now exist in your working directory, but not in the repository. To save your changes to the repository, you need to "commit" them. Make sure you are in your `cs062/lab08` directory, and use the following commit command:

```
git commit -am "increased number of values to 256"
```

All commits require a message that will, hopefully, describe the changes being made. The "-m" option tells Git to use the commit message that will follow in quotes. Failure to include the "-m" option will cause a terminal text editor to be opened for you to enter the message.

If all has gone well, your changes are now also saved in the repository. Make a few more changes to your `Lab08.java` file, remembering to commit your changes each time. We'll see why we did this in a minute.

This may seem like a lot of work just to do the same thing we've been doing all along without Git. There is a point to this, however. Git is handy for keeping a master version of the project that multiple people can checkout, change, and work on. More important, though, is that Git (in fact, all version control systems) don't just save the current version of a file, it saves all versions of a file.

By now we have committed several changes to our project. Let's use the log command to see what we have done, type: `git log`.
Here we have a log of all of the versions of our project. In future labs we will learn how to work with previous versions of the project.

What to hand in

You should turn in the folder `lab08` by dragging it to the dropbox folder, as we have done with previous projects.