Computer Science 62
Lab 8

Wednesday, March 26, 2014

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 "Handouts" page entitled "Introduction to terminal, emacs, subversion, and make". **Read through this handout before coming to lab.**

Getting started

Open a terminal window and create a directory `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 zipped up in the `bailey.jar` file. 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 out 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

SVN to the rescue

Below we will give you some SVN commands that you can follow blindly if you like, but which should be simple and straightforward enough to understand how they work.

Let's begin by creating a new SVN repository for our `Lab08` project. Typically, the repository (or "repo" for short) would be on a separate server so that the files could be shared among many users. Here, we will create the repo in our own directory, just to see how it works.

We begin by creating the SVN repository itself. This needs to be done once, and a single repo can hold many projects.

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" which has the same effect as the commands above.

Next, create a directory that will serve as our SVN repository, and tell SVN that we are using it.

```
mkdir svnroot
svnadmin create /home/achambers/Documents/cs062/svnroot
```

You will want to change "achambers" to your own home directory name (should be the same as your login name) here and in all later commands. With SVN, it is usually best to always use absolute pathnames (the path starting from root "/"). In some cases, including this one, the relative pathname (the path to the repo directory from the current directory) is also acceptable. However, this is not always the case for SVN commands, so absolute pathnames are usually recommended.

Next, we need to import our existing project into our new repo. First, change the current directory to be the `Lab08` project directory. Then, import all of the files into our new repository:

```
cd ~/Documents/cs062/lab08
svn import -m "create new import in repo" file:///home/achambers/Documents/cs062/svnroot/lab08
```

Here, the import command takes all the files and folders in the current directory and adds them to the repository. Notice that we have named the project in the repository as "lab08." This is the same as our project name. It does not have to be, but it is good practice to cut down on confusion.

Now, our repository is setup, and our lab08 project is safely saved to it. Let's test this. Change into your cs062 directory, and rename the lab08 directory to something else.

```
cd ..
mv lab08 old_project
```

The ".." in the change directory command means "go up one level" so we should now be in the cs062 directory. Notice that we used the move command "mv" to do the renaming. You may want to use the "ls" command to verify that the project directory has a new name.

Now, we can try to check out our project from the repository. Type the following at the command line:

```
svn co file:///home/achambers/Documents/cs062/svnroot/lab08
```

The "co" stands for checkout. The current version of the lab08 directory should now exist in your cs062 folder. Change into the lab08 directory and use the commands you know to make sure the correct files exist.

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 (cs062/lab08), 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:

```
svn commit -m "increased number of values to 256"
```

All commits require a message that will, hopefully, describe the changes being made. The "-m" option tells SVN to use the commit message that will follow in quotes. Failure to include the "-m" option will cause a text editor to be opened for you to enter the message. Since the default text editor on most linux systems is a program called "vi," it is easier to just use the "-m" option.

If all has gone well, your changes are now also saved in the repository, and available to be checked out by other users. 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 SVN. There is a point to this, however. SVN is handy for keeping a master version of the project that multiple people can checkout, change, and work on. More important, though, is that SVN (in fact, all version control systems) don't just save the current version of a file, it saves all versions of a file. Let's go back to our sample SVN repository.

By now we have committed several changes to our project. Let's use the status command to see what we have.

```
svn stat -v
```

I get output that looks something like this:

```
achambers@project:~/Documents/cs062/lab08$ svn stat -v
                1        1 achambers    .
                1        1 achambers    bst
                4        4 achambers    bst/Lab08.java
                2        2 achambers    bst/Lab08.class
```

The "-v" option tells svn to give verbose output. This allows us to see the current working version of a file (the first column), as well as the last version of the file that changed (the second column). Above, I can see that I am working on version 4 of my Lab08.java file, and I know that version 4 changed from version 3. You can see any version of the file by simply dumping it's text to the screen as follows.

```
svn cat -r 4 bst/Lab08.java
```

The "-r" option gives the revision number. In this case, I am asking to look at revision 4 of Lab08.java. The "cat" command dumps the text of the file to the screen. There is also a "cat" command in linux, which is where the SVN command gets its name. Use the above command to look at all of the previous versions of your Lab08.java

What to hand in

You should copy and paste your terminal screen after some of the "svn stat" and "svn cat" commands you performed in the previous section into a text file. We want to verify that you were able to check a file into a repository, change it, then recover the text of an old revision of the file. Rename the text file to **Lab8_LastNameFirstName.txt** and submit it to the dropbox.