Do Not Hardcode Input/Output File Pathnames in Your Java Code

Note: the following discussion assumes you have not created a package for your project, i.e., that there is no **package *package-name*;** line in your source code files.

The purpose of this document is to discuss the following: for any homework exercise or programming project which reads from an input file and may or may not write to an output file, in your Java code **do not write hardcoded pathnames to the file(s)**.

For example, suppose your Java source code directory for Project 1 is *C:\Documents\CSE205\Project1\Source* (in Windows) or */home/hsimpson/cse205/project1/source* (in Mac/Linux) and the only source code file of your project is named *Main.java*, which would be located in *C:\Documents\CSE205\Project1\Source\Main.java* or */home/hsimpson /cse205/project1/source/Main.java*.

Assume your source code directory is your current directory, i.e., if you open a Command Prompt window in Windows or a Terminal window in Mac/Linux and navigate to your *Source* directory, then it becomes the current directory. Then,

```
C:\Documents\CSE205\Project1\Source> dir      -- In Windows
$ ls                                          -- in Mac/Linux
```

would display the files in your current directory, among which would include *Main.java*. Note in Windows that your current directory is listed in the prompt string that is displayed while waiting for you to type a command. In Linux, by default, Bash does not display the current directory in the prompt string although it can be configured to do so (I do not know if Linux is the same as Mac in this regard because I do not own a Mac).

You can build and run your program from the command line by using the **javac** (Java compiler) and **java** (loads the Java VM which runs the specified class) commands,

```
C:\Documents\CSE205\Project1\Source> javac Main.java       -- In Windows
$ javac Main.java                                          -- In Mac/Linux
```

This command will compile *Main.java*. Assuming there are no syntax errors, your current directory would now contain a new file named *Main.class*. The .class file is your executable program which you can now run,

```
C:\Documents\CSE205\Project1\Source> java Main      -- In Windows
$ java Main                                         -- In Mac/Linux
```

Now, suppose your program is to open an input file named *in.txt* for reading and an output file named *out.txt* for writing. Suppose *in.txt* is located in *C:\Downloads\CSE205\in.txt* or */home/hsimpson/downloads/cse205/in.txt*. Notice that the input file is not in the current directory. This complicates things.

To open *in.txt* for reading using the *Scanner* class then you would have to provide a pathname to *in.txt* by writing:

```
File inFile = new File("C:\\Downloads\\CSE205\\in.txt");              -- in Windows
File inFile = new File("/home/hsimpson/downloads/cse205/in.txt");    -- in Mac/Linux
```

We call this a **hardcoded pathname**. This will work just fine on your machine but **you cannot submit Main.java for grading containing hardcoded pathnames to the input file (or the output file)**. Why? Because when we run your program using the **java** command as shown above, then will not have a directory named *C:\Downloads\CSE205* or */home/hsimpson/downloads/cse205* and consequently, the attempt to open the file would throw a *FileNotFoundException*. Furthermore, since your program on our system would be unable to open the input file, it would likely be unable to open the output file if you used a hardcoded pathname to the directory where *out.txt* is to be written.

This would cause you to lose significant points due to a simple mistake and neither you nor we want that. You can use hardcoded pathnames in your IDE for testing, but what you need to do for the program you submit is to assume the input and output files are in the same current directory as *Main.java* and *Main.class*, i.e., *in.txt* would have to be copied to *C:\Documents\CSE205\Project1\Source\in.txt* or */home/hsimpson/cse205/project1/source/in.txt*. Then, modify the code that opens the input file,

```
File inFile = new File("in.txt");  // In Windows or Mac/Linux
```

Now the JVM will look for *in.txt* in the current directory which is exactly what we need for testing. The same will be true of the output file that your program writes,

```
File outFile = new File("out.txt");  // In Windows or Mac/Linux
```

and *out.txt* will be written to the current directory where *Main.java*, *Main.class*, and *in.txt* are.

Last thing: if you are using an IDE for development, then for testing within your IDE you need to place *in.txt* in the proper directory so the `new File()` statements above will find it. This varies from IDE to IDE but it will likely always be *some* directory within your project directory.

A simple way to determine what directory is the current directory when you run your program within the IDE is to run this small program which prints the *user.dir* system property that is initialized by the Java system before *main*() is called,

```
// PrintCurDir.java
public class PrintCurDir {
    public static void main(String[] pArgs) {
        System.out.println(System.getProperty("user.dir"));
    }
}
```

The internet says this directory is very likely your current directory. Your IDE may expect the current directory to be where *in.txt* needs to be placed so that you do not have to hardcode the pathname to *in.txt*. Then again, it might not.

Another way to try to figure out where *in.txt* needs to be placed is based on the fact that input and output files are usually located in the same directory. So, we can write a small program that outputs some text to a specially named file, e.g., "__LOOK_FOR_ME__.TXT". Then after the program runs, use the file explorer program of your OS and navigate around in the file system, starting at the folder where you are creating your IDE projects, until you find the file. Whatever directory you are in at that time may or may not be the current directory from your IDE's perspective, but this is most likely the directory where you need to place the input file. This program will create that special file for you,

```
// Class: FindCurDir (FindCurDir.java)
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class FindCurDir {
    public static void main(String[] pArgs) {
        try {
            PrintWriter out = new PrintWriter(new File("__LOOK_FOR_ME__.TXT"));
            out.println("Did you find me yet?");
            out.close();
        } catch (FileNotFoundException exc) {
            System.err.println("Could not open file for writing.");
            System.err.println("Do you have write permission to this directory?");
            System.err.println("Try moving FindCurDir.java to a different directory and then build
                               and run.");
        }
    }
}
```

What if I am using Packages?

The Java convention is to create subdirectories for packages. For example, suppose I am using the XYZ IDE (I made that up) and suppose we create a project named *Project1* in the folder *C:\Documents\CSE205* (this is the root folder for all of my projects). Then the *Project1* **project folder** will be: *C:\Documents\CSE205\Project1.*

Suppose we are going to organize our source and class files in a package named *proj1* causing our IDE to create the **package directory:** *C:\Documents\CSE205\Project1\proj1*.

Next, our hypothetical IDE creates a subdirectory named *Source* within the package directory for the *.java* source code files: *C:\Documents\CSE205\Project1\proj1\Source,* and a subdirectory named *Class* for the *.class* files: *C:\Documents\CSE205\Project1\proj1\Class.*

Now, suppose we add a new class named *Main* to the project; this will cause the IDE to create *Main.java* in: *C:\Documents\CSE205\Project1\proj1\Source\Main.java*

We then type a small program in *Main*, save it, and build it. This produces *Main.class* which is saved in: C*:\Documents\CSE205\Project1\proj1\Class\Main.class*

Then, you can use the trick discussed above for determining which directory should store the input and output files for your IDE.

For grading, it does not matter if you create a package for your source code file or do not create a package. We will use a method that will work for either situation.