Programming Assignment I (3rd week)

Creating a Shell Interface Using Java (A modified version from [www.utc.edu](http://www.utc.edu) )

We will complete the project described in page 149 to 152. It gives a good starting point in Figure 3.37. (Please refer to the attached image files.)

1. Read the project description thoroughly before you do anything else. Make sure you compile and test your program repeatedly as you are implementing each step described below.
2. Type in the **OSProcess** program in Figure 3.13 on page 118 (Check the attached image files). Compile it and run it. Make sure you understand how it works, since you will need to use similar code for you shell.
3. Type the **SimpleShell** program in Figure 3.37, compile it and run it.
4. Begin by adding code that splits the String into a String array. Look up the String class in the Java API and read about the **split()** method. You will be splitting the user input based on a single ‘space’ character and saving the result in a String array.
5. The **ProcessorBuilder** constructor that is described page 150 accepts a List of **String** objects. You will need to find a class that implements the List interface and then transform your String array into a List of Strings. You can use **ArrayList** or List class in java.util package. You will need to read the java API to learn how to add String objects to your **ArrayList** or List. One you figure out your need to write code that iterates through your String array and adds each **String** to the **ArrayList** or List.
6. One you correctly have all of your String in your List, all you need to do is to create your **ProcessBuilder** object.
7. Next, you will need to create a **Scanner** or **BufferedReader** to read the output string of the process builder object. This is almost identical to how it is handled in the **OSProcess** program in page 118. Your **OSProcess** will print out the result of the command. Try it with the **cat** command, the **ls** command, and the **ps** command.
8. Congratulations! Your project is now working! Back up your program to a separate directory in case you mess it up by adding the stuff below.
9. Add code to your project so that if the command entered is “**exit**” or “**quit**” the shell outputs “Goodbye.” And exits the program. You can use **System.exit(0)** to exit your running program. It may be better if the code is in front of the codes that handle **ProcessBuilder** object.
10. Next, get your shell to successfully change directories using the cd command. To do this, you will need to use the directory (File directory) method of the **ProcessBuilder** project, and you will also need to have a File object that keeps track of the current directory. For example, if you are in your home directory and it contains a directory called Project1, and you type:

cd /home/**user[[1]](#footnote-1)**/ Project1

to get into your Project1 directory. If you just type cd

cd Project1

you will get an exception. If you get this far,

cd ..

will not work as expected.

1. Get your program working with relative paths. For instance, make it so typing:

cd Project 1

actually takes you to the Project1 directory instead of throwing an exception. Also

cd ..

should work as expected in this case.

1. Add some error/exception handling and do it gracefully. For instance, if a user types in

cd fakeDirectory

and it does not exist, you should tell the user instead of crashing with an exception. If I can crash your program by trying to use your shell by doing normal things, you won’t get these final points.

1. Adding a History Feature: Many UNIX shells provide a history feature that allows users to see the history of commands they have entered and to return a command from that history. The history includes all commands that have been entered by the user since the shell was invoked. For example, if the user entered the *history* command and saw as output:

0 pwd

1 ls –l

2 cat Prog.java

The history would list pwd as the first command entered, ls –l as the second command, and so on. Modify your shell program so that commands are entered into a history

(Hint: The java.util.ArrayList provides a useful data structure for storing these commands)

Your program must allow users to return commands from their history by supporting the following three techniques:

1. When the user enters the command history, you will print out the contents of the history of commands that have been entered into the shell, along with the command numbers.
2. When the user enters !!, run the previous command in the history. If there is no previous command, output an appropriate error message.
3. When the user enters !<integer value *i*> (for example !5), run the *i*th command in the history. For example, entering !4 would run the fourth command in the command history. Make sure you perform proper error checking to ensure that the interger value is a valid number in the command library.

**Table 1. testing cases of your shell project**

|  |  |  |
| --- | --- | --- |
| **Input test cases** | **Results** | **Points** |
| Compile and runs the program | The shell program can compile and run successfully | 10 |
| ProcessBuilder object | ProcessBuilder object is correctly created, ps, ls, and cat command works | 5 |
| Exit and quit | Exit and quit works | 5 |
| Change directory to /home/user | **cd** followed by **pwd**  You are able to change directory to /home/user directory and display it. (absolute path is working) | 5 |
| Handle absolute path | **cd /home**  You are able to change directory to /home | 10 |
| Relative path | then **cd user**  You are able to change directory to /home/user (relative path is working) | 10 |
| Go back to /home then  **cd ./user/Project1 , cd user/Project1,** or any existing folder  You are able to change directory to /home/user/Project1 | 10 |
| cd ..  You are able to change directory to /home/user | 5 |
| Error handling | **cd fakeDirectory**  Error message for invalid path | 5 |
| history | ‘history <number>’ works well | 5 |
| ‘history !!’ works well | 5 |
| ‘history !<number>’ works well | 10 |
| documentation |  | 10 |
| 1-5 questions |  | 5 |

If you made it this far and everything is working, you’ve done a really great job and you’ve passed all testing cases.

So you are finished! Unless

**Bonus**

Implement the history feature as described at the end of the project description. If you get it working flawlessly and eventually earn an A in the class, then I will think you are awesome and get 20 bonus points on this project as well. If you are only able to get it to work partially, then explain, in detail, what is working and tell me how many bonus points you think it’s worth between 1 and 19. Please make sure the rest of your project is working first.

**What did you learn?**

1. How far did you get in this project? What grade are you expecting? Did you find the milestones in the description helpful? Do you think you could have gotten the project working as well as you did without the milestones?
2. Do you feel like a better programmer now that you’ve completed this project? How does this compare to programming project that you’ve had in prior class?
3. Describe 3 problems (relating to this assignment) that arose while you were working on this project and explain how you solved them.
4. How long did you spend on this assignment? Give me specifics. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Time** | **Activities** | **Outcomes** |
| 03/26 | 2:13 ~ 5:29 | ~step 9 | Successful I think, but only act window command such as ipconfig. So I have to solve this problem after some time sleep… |
| 3/29~30 | 23:00~07:21 | Finish making code | Well operating at window environment. |
| 3/30 | 13:14~18:00 | Test code in Linux system | Something is not work as expected. I mention it detail at the README file. |

1. Total hours spent. Try to be honest and do not exaggerate here.

**Turn in:**

1. Your source code & **Readme** file which includes how to compile and run your source code.
2. Documentation including test results of test cases, you can use appropriate screen shots here to show successful cases. (10 points)
3. The answers to above 1-5 questions. (5 points)

Note

* Make frequent backups of your program so that you can revert to a working copy if you mess something up.
* Do NOT turn a program that will no compile. We will not be able to test it, and we can’t spend time trying to get it working, so you will not get any credit.

1. **user** is assumed to be your username. [↑](#footnote-ref-1)