CSC 207 Assignment 2A
Due Date: 5th November @ 11:59pm

Introduction:

In Assignment1, you and your partner used CRC cards to design a set of classes for maintaining a **mock file system** and interacting with it in a program that operates like a Unix shell. You also did some minimal programming in order to get used to sharing code using an SVN repository.

In Assignment2, you will be working in a team of 4. The design requirements for Assignment2A are quite similar to those of Assignment1, although we have modified the commands a bit. There will be two parts to Assignment2. This is Assignment2A or the first part of Assignment2. Read this document carefully.

Commands:

In all of the following commands, there may be any amount of whitespace (1 or more spaces and tabs) between the parts of a command.

For example (all the four variations are correct and considered valid):

```
/#: mv someFile1       someFile2
/#: mv          someFile1          someFile2
/#: mv someFile1 someFil
/#:                  mv someFile1 someFile2
```

Note: Furthermore, your program must not crash. Square brackets indicate an

optional argument. Some commands may have **...** which indicates a list of arguments. And every PATH, FILE, and DIR may either be a relative path or a full path (absolute path).

exit

Quit the program.

mkdir *DIR ...*

Create directories, each of which may be *relative* to the current directory or may be a *full* path.

cd *DIR*

Change directory to DIR, which may be relative to the current directory or may be a full path. As with Unix, `..` means a parent directory and a `.` means the current directory. The directory separator must be /, the forward slash. The root of the file system is a single slash: `/` .

ls *[PATH ...]*

If no paths are given, print the contents (file or directory) of the current directory, with a new line following each of the content (file or directory).

Otherwise, for each path p, the order listed:

- If p specifies a file, print p (i.e. the name of the file only)
- If p specifies a directory, print p, a colon, then the contents of that directory, then an extra new line.
- If p does not exist, print a suitable error message.

pwd

Print the current working directory path (i.e. the whole absolute path)

pushd *DIR*

Saves the current working directory by pushing onto the directory stack and then changes the new current working directory to DIR. The push must be consistent as per the LIFO behavior of a stack. The pushd command saves the old current working directory in directory of stack so that it can be returned to at any time (via the popd command). The size of the directory stack is dynamic and dependent on the pushd and popd commands.

popd

Remove the top entry from the directory stack, and cd into it. The removal must be consistent as per the LIFO behavior of a stack. The popd command removes the top most directory from the directory stack and makes it to the current working directory. If there is no directory onto the stack, then give appropriate error message.

history *[number]*

This command will print out recent commands, one command per line. i.e.

```
1 cd ..
2 mkdir textFolder
3 echo "Hello World"
4 fsjhdfks
5 history
```

The above output from history has two columns. The first column is numbered such that the line with the highest number is the most recent command. The most recent command is history. The second column contains the actual command. Note: Your output should also contain as output any syntactical errors typed by the user as seen on line 4.

We can truncate the output by specifying a number (>=0) after the command. For instance, if we want to only see the last 3 commands typed, we can type the following on the command line:

```
history 3
```

And the output will be as follows:

```
4 fsjhdfks
5 history
6 history 3
```

cat *FILE*

Display the contents of FILE in the shell.

echo *STRING [> OUTFILE]*

If OUTFILE is not provided, print STRING on the shell. Otherwise, put STRING into file OUTFILE. STRING is a string of characters surrounded by double quotation marks. This creates a new file if OUTFILE does not exists and erases the old contents if OUTFILE already exists. In either case, the only thing in OUTFILE should be STRING.

echo *STRING >> OUTFILE*
Like the previous command, but appends instead of overwrites.

man *CMD*
Print documentation for CMD

==========

Task0 (MUST BE COMPLETED BY EACH TEAM MEMBER individually): DUE ON 17TH October @ 11:59PM

1) Check whether your *UTORID Username* and *password* works correctly on the SVN repository.
2) Complete the honor code in *JShell.java* and commit it back to your SVN repo.
3) Read the readMe files that I have provided in each folder of your SVN repo to understand what is expected of you.

Failure to complete Task0 by due date will result in that team member losing one letter grade of his/her Assignment2A grade.

Task1:
Argue with your team until you have worked out a new design. Update your CRC cards and commit them or explain why no changes were necessary. Use the same format as last time (i.e. in Assignment1).

Task2:
Write the shell program implementing each of the above mentioned commands follow the SCRUM software development process. Make sure that, by the deadline of Assignment2A, the code you commit has been tested and fully documented. Remember that you should try to check in code that can be compiled and executed into your SVN repository.

In order to do well for this Assignment, you MUST strictly adhere to the SCRUM software development process. Follow the instructions carefully that are mentioned in the **grading scheme** and what I talked about in lecture regarding Assignment2.

On **October 29th and October 30th**, you will demo your Assignment2A to TA. More details on this later.