CSC B07 Assignment 2B
Due date: 26th November @ 11:59pm

1 Introduction

In Assignment2B, you will continue to work on the mock file system and JShell. We have modified the commands a bit, hence read this document very carefully. Note that at end of Assignment2B, all commands and functionality that were mentioned in Assignment2A and now mentioned in Assignment2B must be completed. If you had any left over work from Assignment2A, your team must ensure that it is completed in Assignment2B. If you create any new Exception classes, make sure to update your CRC Cards accordingly with CRC Cards. You MUST also use Generics.

As in Assignment2A, you will continue to follow the Scrum software development process in 2B as well. At the end of Assignment2B, just like in Assignment2A, each one of you will be asked to evaluate your team members. Also just like Assignment2A, completing this evaluation is mandatory and part of your assignment and failure to do this will result in an individual penalty. And finally, just like assignment 2A, do not work create any new functionality or commands that the customer has not asked for. Creating new functionality or changing the requirements of the customer will result in a penalty on your grade.

2 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 three variations are correct and valid):

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

Furthermore, your program must not crash. Square brackets indicate an optional argument.

- Clarification: ... indicates a list.
- Clarification: every PATH, FILE, and DIR may either be a relative path or a full path (absolute path).
- **Addition:** redirection, either with > OUTFILE or >> OUTFILE, can be applied to every command except for exit. This captures the output of the command and redirects it to OUTFILE, and no output is shown to the user. MAKE SURE THAT YOU NOT REDIRECT ANY ERRORS. ALL ERRORS MUST STILL BE DISPLAYED ON THE CONSOLE. YOU ONLY REDIRECT STD OUTPUT AND NOT STDERROR. IF YOU ARE NOT FAMILIAR WITH STD OUTPUT AND STDERROR, PLEASE REFER TO YOUR LABS OR TALK TO YOUR INSTRUCTOR. If a command does not give any output (such as the cp command), then you must not create OUTFILE.

2.1 List of commands

1. **exit**

   Quit the program

2. **mkdir DIR ...**

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

3. **cd DIR**

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

4. **ls [-R] [PATH . . .]**

   Addition: if –R is present, recursively list all subdirectories.
   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
   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 message.

5. **pwd**

   Print the current working directory (including the whole path).

6. **mv OLDPATH NEWPATH**

   Addition: This is a new command in Assignment2B Move item OLDPATH to NEWPATH. Both OLDPATH and NEWPATH may be relative to the current directory or may be full paths. If NEWPATH is a directory, move the item into the directory.

7. **cp OLDPATH NEWPATH**

   Addition: This is a new command in Assignment2B Like mv, but don't remove OLDPATH. If OLDPATH is a directory, recursively copy the contents.

8. **cat FILE . . .**

   Addition: if there are more than one file, you must display all their contents on the console. (assuming all are a valid path). For any file that contains an invalid path, display an appropriate error for that path only. All other valid paths must still be shown on the console.

   Display the contents of FILE and other files on the console in the shell.

9. **get URL**

   Addition: This is a new command in Assignment2B URL is a web address. Retrieve the file at that URL and add it to the current working directory.
   Example1:
   get http://www.cs.cmu.edu/ spok/grimmtmp/073.txt
   Will get the contents of the file, i.e. 073.txt and create a file called 073.txt with the contents in the current working directory.
   Example2:
   get http://www.ub.edu/gilcub/SIMPLE/simple.html
   Will get the contents of the file, i.e. simple.html (raw HTML) and create a file called simple.html with the contents in the current working directory.

10. **echo String**

    Print String

11. **man CMD**

    Print documentation for CMD.

12. **pushd DIR**

    Saves the current working directory by pushing onto directory stack and then changes the new current working directory to DIR. The push must be consistent as per the LIFO behaviour of a stack. The pushd command saves the old current working directory in directory 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 the popd commands.

13. **popd**

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

14. **history [number]**

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

```
cd ..
mkdir textFolder
echo \Hello World"
fsjhdfks
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 only to see the last three commands typed, we can type the following on the command line:

    history 3

    And the output will be as follows:

```
fsjhdfks
history
history 3
```

15. **save FileName**

    Addition: The above command will interact with your real file system on your computer. When the above command is typed, you must ensure that the entire state of the program is written to the file FileName. The file FileName is some file that is stored on the actual filesystem of your computer. The purpose of this command is to save the session of the JShell before the user closes it down. You must ensure that the entire mock filesystem including any state of any of the commands is written to FileName so that the next time the JShell is started, the user can type in the command `load FileName` to reinitialize the last saved session and begin from where they left off. You are free to use any format (XML or JSON or something else) for the creation of the above file. The path for FileName must be a valid path on your computer and not on JShell. For instance, if the user types in the command `save /Users/User1/Desktop/save.txt`, then you will create a file `save.txt` on your computer that will save the session of the JShell. If the above file exists on your computer, then you must overwrite the file completely.

16. **load FileName**

    Addition: When the user types in the above command, your JShell must load the contents of the FileName and reinitialize everything that was saved previously into the FileName. This allows the user to start a new JShell session, type in `load FileName` and resume activity from where they left off previously. We hope that this command is the very first thing that the user types in when starting a new JShell, however, if the user were to type in the `load` command at any point after any of the other commands have been executed, you must ensure that the `load` command is disabled. This is because, if the user were to type in this command after any of the other commands have been typed, then you already have a new mock filesystem running inside JShell. This may cause potential confusion whether the current running mock file system must be discarded before loading the previous session of some other mock filesystem that was saved in FileName.

17. **Addition:** You are now asked to implement the `find` command. The syntax of the `find` command is as follows: `find path ... -type [f|d] -name expression`. So here are some examples of how the `find` command may be used:

    - `find /users/Desktop -type f -name "xyz"`. This command will search the directory `Desktop` and find all files (indicated by `type f`) that have the name exactly `xyz`.
    - `find /users/Desktop -type d -name "abc"`. This command will search the directory `Desktop` and find all directories (indicated by `type d`) that have the name exactly `abc`.
    - `find /users/Desktop`. This command will result in an error because it has missing arguments. Note: For any missing arguments or incorrect argument parameters, your program must not crash and result in error.
    - `find /users/Desktop /users/Desktop1 -type d -name "abc"`. This command will search the directory `Desktop` and `Desktop1` and find all directories (indicated by `type d`) that have the name exactly `abc`.
    - `find /users/Desktop /users/Desktop1 -type f -name "abc"`. This command will search the directory `Desktop` and `Desktop1` and find all directories (indicated by `type f`) that have the name exactly `abc`.
    - If at any point one of the path in this command is invalid, you must print out an error for that path, however, you must continue searching for any other files or directories that may exist in other valid paths.
    - The `find` command must accept the path just as other commands in your assignment handout, as relative or in absolute path.
    - You can display the output of the `find` command in any format that you wish.

18. **Addition:** You are now asked to implement the `tree` command. The the `tree` command takes in no input parameter.

    - When the user types in the `tree` you must starting, from the root directory ('\') display the entire file system as a tree. For every level of the tree, you must indent by a tab character.
        - For instance if the root directory contains two subdirectories as 'A' and 'B,' then you will display the following:

```
\
  A
  B
```

        - For instance if the root directory contains two sub directories as 'A', 'B', 'C' and 'A' in turn contains 'A1' and 'A2', then you will display the following:

```
\
  A
   A1
   A2
  B
  C
```

        - When the user types in `tree` and the only directory present is the root directory, then you simply show the root directory.

```
\
```

In order to do well for this Assignment, you MUST strictly adhere to the SCRUM software development process just as Assignment2A. Follow the instructions carefully that are mentioned in the grading scheme (same grading scheme applies for Assignment2B and Assignment2A).