CS112 – Fall 2023
Project02
Instructor: Paul Haskell

INTRODUCTION

In this lab, you will work with recursion and trees to implement a model of a computer's file system. Your system will store and manipulate files and directories, just like the MacOS or Windows file system on your computer.

Setup

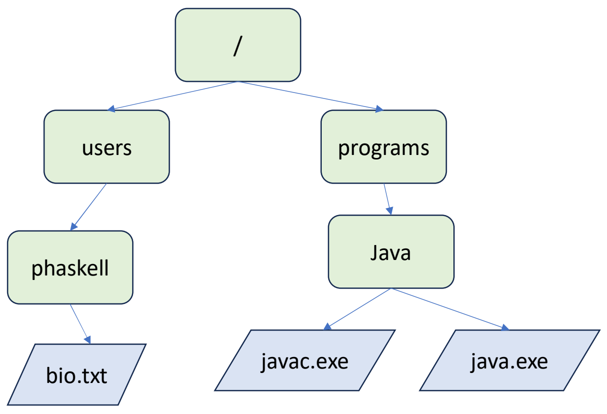
Your **FileSys.java** program will give the user a "command prompt", just like the "terminal" program on the Mac or the "command.exe" program on Windows. Your program's user can run a bunch of commands (explained below) to create, view, and remove files and directories.

A big part of the project will be design work: what classes do you need and what should they do? Another big part of the project will be testing: how do you make sure your program is robust (does not crash) and correct (does the right thing)?

Details

A file system is the software and structures on your computer that manages files and directories ("folder" is another name for directory). Every file system organizes directories as a tree. The directory at the top if the tree is called "root" and is often given the symbol "/" as a directory name. Every program running on a computer, including terminal command windows, has a "current" directory, which is the directory from which the program looks for files to read or write. Programs can change the current directory, either to the parent of the current directory or to any child directory inside the current directory.

Directories can store files as well as other directories. Files store data: text, executable programs, numerical data, etc. But files do not store other files or directories.



The "full directory path" of a file or directory is the listing of all of the directories, starting with root, that must be traversed to reach a particular file or directory. The components of the directory path are usually separated by the "/" character. For example, the full directory path of the **bio.txt** file is **/users/phaskell/bio.txt** . The first "/" refers to the "root" directory, and the other "/" characters are just used to separate the individual directory names and file name.

For this project, you will not be using the actual file system on your computer. You will not use the **File** and **Directory** classes in the **java.io** library. All the "files" and "directories" created inside your program will simply be objects inside your program.

Probably your first task for the project is to think about your design for files and directories. How to represent files and directories? What do they have in common? What is different about files and directories? What classes do you want to design to represent these capabilities? What are the methods and key class variables for each class?

Your **FileSys.java** program will implement a simple version of a terminal window, to be used by a human user. Your terminal window repeatedly will print a "command prompt" (please use "**prompt>** ", noting the space after the greater-than character) and will read a line of user input from the keyboard. Each line of user input should start with a command, and some commands take an additional argument. The commands you must implement are:

- **create fileName** This command creates a new file with the given filename. When this command is entered, your **FileSys.java** program should keep reading characters from the keyboard input and saving the characters to your "file". Keep reading the input until it contains a ~ character (tilde). The tilde indicates the end of the file. The tilde character itself should not be saved to the file.
  If there already is a file or directory in the current directory named "fileName" then report an error.
- **cat fileName** "cat" is short for "catenate". This command prints out the contents of the file named "filename" to the terminal output. If there is no file named "filename" in the current directory, or if there is a directory named "filename", then report an error.
- **rm filename** remove the file with the given name from the current directory. If there is no file named "filename" or if "filename" is a directory, then report an error.
- **mkdir dirName** make a new directory named "dirName" inside the current directory. If there already is a file or directory in the current directory named "dirName" then report an error.
- **rmdir dirName** remove the directory named "dirName" (and all of its contents) from the current directory. If there is no directory named "dirName" or if "dirName" is a file, report an error.
- **cd dirName** if "dirName" is a directory inside the current directory, change the current directory to be the directory named "dirName". If "dirName" is "/", then change the current directory to be the root directory of your file system. If "dirName" is ".." then change the current directory to be the parent of the current directory. If "dirName" contains several directory components e.g. "childDir/grandChildDir" or "/users" then change to the appropriate directory. If "dirName" is a file in the current directory or is not an existing directory path, report an error.

  - **ls** print out ("list") all the files and directories inside the current directory, in alphabetical order. Print "(*)" after the directories. For example:
    ```
    prompt> ls
    hereIsADirectory (*)
    hereIsAnotherDirectory (*)
    iAmAFile
    iTooAmAFile
    ```

- **du** "disk usage". This command uses recursion! Find and print the total size (in bytes) of all the files in this directory and all the files in all subdirectories.
- **pwd** "print working directory". Print the full directory path to the current directory, starting from root.
- **find name** Find <u>all</u> files or directories named "name" in the current directory <u>or any child directory</u> and print the full directory path of all such files or directories. For example:
  ```
  prompt> cd /
  prompt> find myStats.txt
  /users/steph/myStats.txt
  /users/klay/myStats.txt
  /data/network/myStats.txt
  ```
  This probably uses recursion also.
- **exit** exit the FileSys program.

Upper/lowercase distinctions matter. For example, it is permitted to have two different files named **file1.txt** and **fiLe1.txt**.

<u>Error reporting</u>:

- if the user enters a command other than one of the commands listed above,
- if the user does not enter a file or directory name when one is required,
- if the user commits some other error as described above,

then your FileSys program should print some useful error message (that starts with the word "ERROR" in all capitals) and should print the prompt and await the next user command.

Part 1 – Design and Testing

Proper design and testing are really important for this project. In Part 1, you will focus on design and testing.

- Please prepare a 1-2 page document **design.txt** that describes your planned design for the project. What classes will you create? What public methods will they have? Which classes are derived from other classes? What key data structures will you build?
- Please prepare a test file **testInput.txt** that contains input commands that you plan to feed to your program to test it. Your input commands should test every possible command and should test error handling also.

Since your **FileSys.java** program will read user inputs from the keyboard, in Part 2 you can use "input file redirection" to feed your **testInput.txt** to your program for testing. Then you can inspect the outputs to be sure they are correct. (You also can use "output file redirection" to capture the output from your program.)

Part 2 – the Program

For Part 2, your **FileSys.java** must implement all the actual commands, creating and deleting files and directories. All commands should do the right thing and produce the correct outputs. Errors should be detected and an appropriate message should be printed out.

With your test files from Part 1 available, you should be in good position to test. But as you test, you probably will think of changes to your design and also to your test instructions. For Part 2, in addition to your FileSys.java (and any other Java files you write), Please prepare an **updates.txt** doc, explaining what you changed with your design and your testing since Part 1, and why.

Reminder

Put all your files in your **Project02** directory and push to GitHub before the deadlines.

- Part 1 (**design.txt** and **testInput.txt**) are due before 11:59pm on November 29th
- Part 2 (**FileSys.java** and **updates.txt**) are due before 11:59pm on December 6th. **You many not use late-day-credits to turn in Part 2 late! December 6th is the last day of the semester.**

Conclusion

This project combines several of the trickiest technologies we learned about this semester: input processing, exception handling, trees, and recursion. And testing! Nice work!

Grading Rubric

Part 1's **design.txt** is worth 0-25 points, depending on the subjective opinion of the grader

Part 1's **testInput.txt** is worth 0-25 points, depending on the subjective opinion of the grader

Part 2's **FileSys.java** is worth 100 points:

- 7 points for each of 10 test cases
- 0-30 points for software quality, as judged subjectively by the grader

Part 2's **updates.txt** is worth 0-25 points. It should list and explain changes both to the design and to your list of tests. The description need not be super-long, but it should list the final test inputs you used.