Project # 2
Simulation of UNIX Directory Navigation
Due: Friday, Mar 8 by 9:00AM

Suggested Reading:
Section 4.1 of Weiss

Overview

In this project you will create an interactive program which simulates navigation and modification of the UNIX directory system.

A directory system in most operating systems can be viewed as a *tree* of nodes where each node is either a directory (equivalently a "folder") or a file. There is a root directory which may have files in it (these can be considered *leaves*), and possibly subdirectories. Subdirectories may be either *leaves* (i.e., if a subdirectory is empty) or *internal nodes* if they have contain one or more file or subdirectory (children).

Your program will start with a directory system with just an empty root directory `/` ("slash") your. It will then process user input corresponding to UNIX commands to build, modify and navigate a data structure representing such a directory tree.

Paths and System State

Figure gives an example of a directory system. The properties of a directory system and assumptions we will make are summarized as follows.

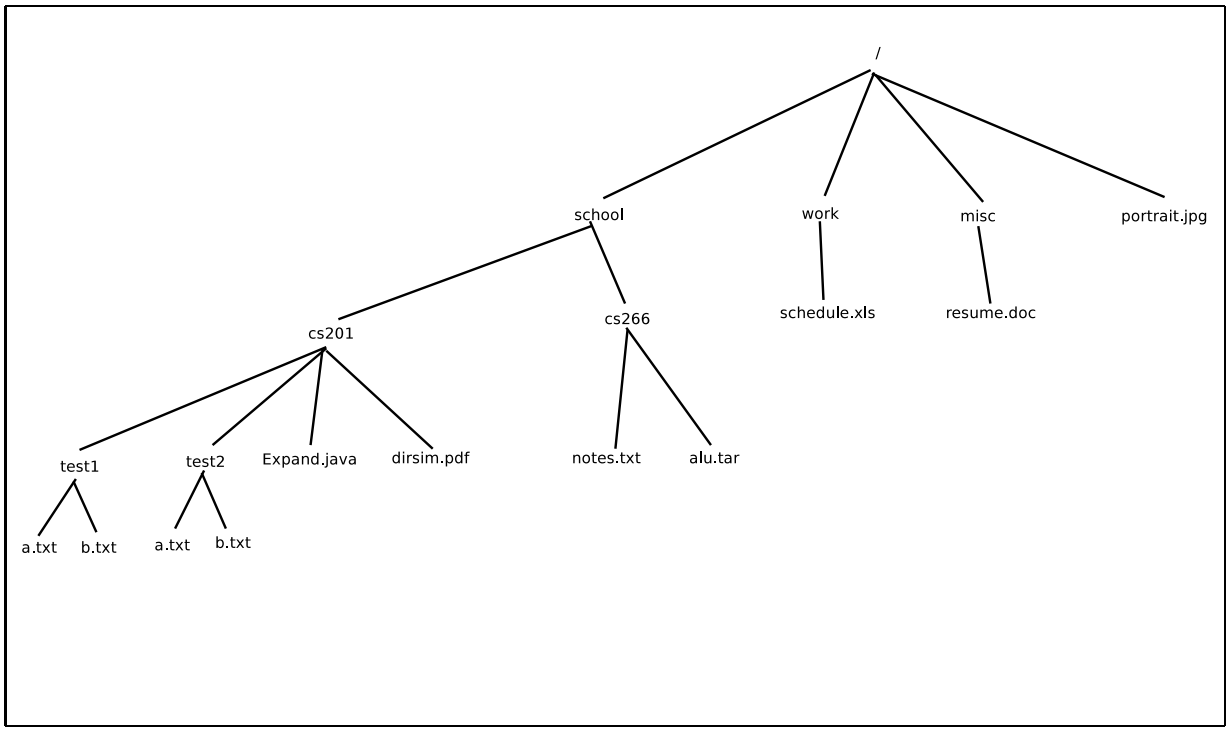

Figure 1: Example Directory System

- As in the figure, the root directory will always be `/`. When your program starts, the entire directory system is just the root directory containing no subdirectories or files. The root directory can never be removed or renamed.

- Every file and directory in the system can be referred to by unique *path* starting at the root `/`. We call such a path starting at the root an **absolute path**. Paths are specified as strings with the `/` symbol separating components of the path (this means that we will also *not* allow any file or directory other than the root to contain a `/`).

  In the figure, the path to the `cs201` directory is `/school/cs201`. Other paths are `/work/schedule.xls`, `/school/cs201/test1/a.txt` and `/school/cs201/test2/a.txt`

- At any point in time, the user is at his/her **present working directory**. Moving around the directory system corresponds ‘ to changing the working directory (using commands like `cd`, `pushd` and `popd` described below). The `pwd` (“present working directory”) reports the absolute path to the current directory.

- All directories **except** the root directory have a parent directory. The parent directory of the present working directory is specified as `..` (“dot-dot”).

- In addition to absolute paths, there are **relative paths**. These paths are **relative** to the current working directory.

  For example, suppose my current working directory is `/school`; then I could refer to `Expand.java` using the relative path `cs201/Expand.java`

  I could also refer to the file `alu.tar` by the relative path `../cs266/alu.tar`

  Note: Given a string for a path, you can **always** determine if it is absolute or relative:

  paths beginning with `/` are absolute

  all other paths are relative

Program Description

Your program will be called `DirSim` (it is a full program – i.e., your class `DirSim` will have a main method). It takes no command line arguments – it will just immediately enter a command interpreter.

As stated earlier, initially there will just be a root directory called `/`. After this the user will be able to enter commands to build, change and navigate the directory structure.

The commands are described below. Angle brackets are used to indicate that a user-specified string is expected.

`pwd`

prints the absolute path of the present working directory. Initially, of course, it is just `/`.

`mkdir <dirname>`

creates a new subdirectory in the current working directory with the given name.

Example: `mkdir cs335`

If there is already a subdirectory or a file of that name, the command fails.

Note: unlike in UNIX, the specified name cannot be a path; you do not need to do any path parsing for this command (e.g., if there is a `/` in the name, it is an error).

`touch <filename>`

creates a file with the given name.

Example: `touch spam.jar`

If there is already a subdirectory or a file of that name, the command does nothing.

Note: unlike in UNIX, the specified name cannot be a path; you do not need to do any path parsing for this command (e.g., if there is a `/` in the name, it is an error).

`cd <directory>`

changes the current working directory to that of the given directory (if it exists).

The target directory is given as a path (either relative or absolute).

Examples: suppose my present working directory is `/school/cs201`; the following commands would take me to the same place.

```
cd ../266
cd /school/cs266
```

Also assuming the present working directory is `/school/cs201`, I could do this:

```
cd ../cs266/../cs201
```

This is a valid command that just keeps me in place.

Note: this command requires path parsing!

`rm <pathname>`

This isn't quite the same as the UNIX `rm` command in that we will have it apply to both files and directories. In the case of a file, the files is removed from the current directory (if it exists – if not a message is printed). In the case of a directory the directory is removed, *but only after all of its subdirectories have been removed (recursively)* (this is like a `rm -r` command in UNIX).

Note: any attempt to remove the current working directory should fail! For example, `rm ..`

Note: this command requires path parsing!

`ls`

The `ls` command lists all of the files and subdirectories contained in the current working directory (*not* recursively). It does not take a parameter as in UNIX. It also does not matter which order the files and subdirectories are listed in.

To indicate to the user which entries in the directory are files and which are subdirectories, you will put a `/` after all subdirectories. For example, if your current working directory is `/school/cs201`, the `ls` command would produce something like:

```
test1/
test2/
Expand.java
dirsim.pdf
```

This is a simplified version of the UNIX `ls` command.

`find <name>`

finds all files and directories with the given name either at the current working directory or in some subdirectory. The absolute paths to all matches are printed.

For example, suppose the current working directory is `/school/cs201`; the command `find a.txt` would produce output like:

```
/school/cs201/test1/a.txt
/school/cs201/test2/a.txt
```

Note: the name is **not** a path; if the user specified a path (i.e., a string containing `/`) an error is reported.

This is a simplified version of the UNIX `find` command.

`exit`

Just exits the `DirSim` program.

Sample Session

The user prompt is `>`; all other lines are output. This example builds part of the example in Fig 1.

```
% java DirSim
> pwd
  /
> ls
> mkdir school
> ls
  school/
> mkdir work
> ls
  school
  work
> cd school
> pwd
  /school
> mkdir cs201
> cd CS201
  no such directory
> cd cs201
> pwd
  /school/cs201
> touch dirsim.pdf
> mkdir test1
> ls
  dirsim.pdf
  test1/
> cd foo
  not a directory
> cd test1
> pwd
  /school/cs201/test1
> mkdir
> cd ../..
> pwd
  /school
```

```
> find a.txt
  /school/cs201/test1/a.txt
  /school/cs201/test2/a.txt
> mkdir 266
> cd cs266
> rm ..
  cannot remove ..
> rm ../cs201
> cd ..
> ls
  cs266
```

Writing Your Program

You should design a class providing the appropriate primitives manipulating tree data structures. Isolate the class from the outside world as much as possible via `private` data members. An instance of this class encapsulates the *state* of the directory system including current working directory. The wrapper program should do the command parsing and interface with the class.

Tips and Rules of the Road:

- Parsing pathnames will be important. The `split()` method in the `String` class should be handy.
- Note that some commands to not require or expect path names; you might start with these.
- There are no runtime requirements for this project.
- You are free to use the Java `ArrayList`, `LinkedList` and `Stack` (and of course `String`, `Scanner`, etc.). But you don't have to use them if you don't want (except `String` and `Scanner`). A solution which starts from scratch might be cleaner anyhow...
- If there is some class you'd like to use, but aren't sure it's allowed, just ask!

Turning in Your Program

You will submit all of your Java files including one named `DirSim.java` which must contain your `main` method. Submission will be through blackboard.