HOMEWORK 5 - Fall 2020

HOMEWORK 5 - due Tuesday, November 3rd no later than 5:00PM

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may not use any Java API Data Structures to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

In this assignment, you will be modeling a file system using a ternary (3-child) tree data structure. In most operating systems, files are stored in directories. These directories may contain other directories along with files. For example, the following *file tree* is an abbreviated example of the Linux file system:

Sample Linux file system.

In addition to a file system, an OS typically also provides a *shell*, which is an interface allowing you to run programs and move through the file system. You are likely very familiar with a *Graphical User Interface*, or *GUI*, to interact with your host, although a much simpler interface is sometimes all that is required.

In Unix-based operating systems, the Bourne-again shell, or *bash*, is a terminal-based shell allowing you to run programs by typing commands into a terminal prompt. An example of a bash terminal is shown in the provided below, from a machine running Ubuntu 15.04, a linux distribution. In this assignment, you will implement a very simple bash shell providing a few very basic functions allowing you to interact with your file system.

Bash running on Ubuntu 15.04.

Required Classes

The following sections describe classes which are required for this assignment. Each section provides a description and the specifications necessary to complete each class. If you feel that additional methods would be useful, you may feel free to add them during your implementation as you see fit. However, all the variables and methods in the following specifications must be included in your project.

UML Diagram.

1. DirectoryNode

Write a fully-documented class named `DirectoryNode` which represents a node in the file tree. The `DirectoryNode` class should contain 3 `DirectoryNode` references, `left`, `middle`, and `right`. In addition, the class should contain a `String` member variable `name`, which indicates the name of the node in the tree.

NOTE: The `name` member variable should be a full string with no spaces, tabs, or any other whitespace.

Since `DirectoryNodes` can be either a *file* or a *folder*, include a `boolean` member variable named `isFile` to differentiate between the two. Note that if this value is set to `true`, then the node is not a directory, and therefore should NOT contain any children. That is, files are not allowed to have children.

- **`public DirectoryNode()` - constructor (you may include a constructor with parameters)**
- **Three `DirectoryNode` member variables:**
  - **`left`**
  - **`middle`**
  - **`right`**
- **A `String` member variable:**
  - **`name`**
- **A `boolean` member variable:**
  - **`isFile`**
- **`public void addChild(DirectoryNode newChild) throws FullDirectoryException, NotADirectoryException`**
  - ***Brief:***
    - **Adds `newChild` to any of the open child positions of this node (`left`, `middle`, or `right`).**
    - **NOTE: Children should be added to this node in left-to-right order, i.e. `left` is filled first, `middle` is filled second, and `right` is filled last**
  - ***Preconditions:***
    - **This node is not a *file*.**
    - **There is at least one empty position in the children of this node (`left`, `middle`, or `right`).**
  - ***Postconditions:***
    - **`newChild` has been added as a child of this node. If there is no room for a new node, throw a `FullDirectoryException`.**
  - ***Throws:***
    - **`NotADirectoryException`: Thrown if the current node is a *file*, as files cannot contain `DirectoryNode` references (i.e. all files are leaves).**
    - **`FullDirectoryException`: Thrown if all child references of this directory are occupied.**

2. DirectoryTree

Write a fully-documented class named `DirectoryTree` which implements a ternary (3-child) tree of `DirectoryNodes`. The class should contain a reference to the root of the tree, a cursor for the present working directory, and various methods for insertion and deletion.

The `DirectoryTree` class should provide an implementation for the operations defined for the system (*see list below and sample I/O for details)*. The class should contain methods for moving the cursor through the file system, printing the filepath of the present working directory (cursor location), listing the directories and files in the present working directory, printing the entire file system, and finding a file in the file system. For further information, see the UML diagram and sample I/O below.

- Two `DirectoryNode` member variables:
  - `root`
  - `cursor`

- `public DirectoryTree()` - constructor (you may include a constructor with parameters)
  - *Brief:*
    - Initializes a `DirectoryTree` object with a single `DirectoryNode` named `"root"`.
  - *Preconditions:*
    - None.
  - *Postconditions:*
    - The tree contains a single `DirectoryNode` named `"root"`, and both `cursor` and `root` reference this node.
    - NOTE: Do not confuse the name of the directory with the name of the reference variable. The `DirectoryNode` member variable of `DirectoryTree` named `root` should reference a `DirectoryNode` who's `name` is `"root"`, i.e. `root.getName().equals("root") is true`.

- `public void resetCursor()`
  - *Brief:*
    - Moves the cursor to the root node of the tree.
  - *Preconditions:*
    - None.
  - *Postconditions:*
    - The cursor now references the root node of the tree.

- `public void changeDirectory(String name) throws NotADirectoryException`
  - *Brief:*
    - Moves the cursor to the directory with the name indicated by `name`.
  - *Preconditions:*
    - 'name' references a valid directory ('name' cannot reference a file).
  - *Postconditions:*
    - The cursor now references the directory with the name indicated by `name`. If a child could not be found with that name, then the user is prompted to enter a different directory name. If the name was not a directory, a NotADirectoryException hs been thrown
  - *Throws:*
    - `NotADirectoryException`: Thrown if the node with the indicated name is a *file*, as files cannot be selected by the cursor, or cannot be found.
  - NOTE: In modern operating systems, the change directory command (`cd {path}`) allows the user to jump from a current directory to any other directory in the file system given an absolute or relative path. In this assignment, you will only be required to change directory to *direct* children of the cursor (`cd {dir}`). However, you may implement the more general command for absolute paths for extra credit.

- `public String presentWorkingDirectory()`
  - *Brief:*
    - Returns a String containing the path of directory names from the root node of the tree to the cursor, with each name separated by a forward slash "/".
    - e.g. `root/home/user/Documents` if the cursor is at `Documents` in the example above.
  - *Preconditions:*
    - None.
  - *Postconditions:*
    - The cursor remains at the same `DirectoryNode`.

- `public String listDirectory()`
  - *Brief:*
    - Returns a String containing a space-separated list of names of all the child directories or files of the cursor.
    - e.g. `dev home bin` if the cursor is at `root` in the example above.
  - *Preconditions:*
    - None.
  - *Postconditions:*
    - The cursor remains at the same `DirectoryNode`.
  - *Returns:*
    - A formatted String of `DirectoryNode` names.

- `public void printDirectoryTree()`
  - *Brief:*
    - Prints a formatted nested list of names of all the nodes in the directory tree, starting from the cursor.
    - See sample I/O for an example.
  - *Preconditions:*
    - None.
  - *Postconditions:*
    - The cursor remains at the same `DirectoryNode`.

- `public void makeDirectory(String name) throws IllegalArgumentException, FullDirectoryException`
  - *Brief:*
    - Creates a directory with the indicated name and adds it to the children of the cursor node. Remember that children of a node are added in left-to-right order.

- *Parameters:*
  - `name` The name of the directory to add.
- *Preconditions:*
  - 'name' is a legal argument (does not contain spaces " " or forward slashes "/").
- *Postconditions:*
  - A new `DirectoryNode` has been added to the children of the cursor, or an exception has been thrown.
- *Throws:*
  - `IllegalArgumentException`: Thrown if the 'name' argument is invalid.
  - `FullDirectoryException`: Thrown if all child references of this directory are occupied.

- `public void makeFile(String name) throws IllegalArgumentException, FullDirectoryException`
  - *Brief:*
    - Creates a file with the indicated name and adds it to the children of the cursor node. Remember that children of a node are added in left-to-right order.
  - *Parameters:*
    - `name` The name of the file to add.
  - *Preconditions:*
    - 'name' is a legal argument (does not contain spaces " " or forward slashes "/").
  - *Postconditions:*
    - A new `DirectoryNode` has been added to the children of the cursor, or an exception has been thrown.
  - *Throws:*
    - `IllegalArgumentException`: Thrown if the 'name' argument is invalid.
    - `FullDirectoryException`: Thrown if all child references of this directory are occupied.

3. BashTerminal

Write a fully-documented class named `BashTerminal`. The class should contain a single `main` method which allows a user to interact with a file system implemented by an instance of `DirectoryTree` using the following commands (note that commands are case-sensitive and will always be lower-case):

Command	Description
`pwd`	Print the "present working directory" of the cursor node (e.g `root/home/user/Documents`).
`ls`	List the names of all the child directories or files of the cursor.
`ls -R`	Recursive traversal of the directory tree. Prints the entire tree starting from the cursor in *pre-order* traversal.
`cd {dir}`	Moves the cursor to the child directory with the indicated name (Only consider the *direct* children of the cursor).
`cd /`	Moves the cursor to the root of the tree.
`mkdir {name}`	Creates a new directory with the indicated name as a child of the cursor, as long as there is room.
`touch {name}`	Creates a new file with the indicated name as a child of the cursor, as long as there is room.
`exit`	Terminates the program.

It should be noted that these commands should all have the same effect as if you were to execute them in a live bash shell on any Unix-based operating system (linux, mac, etc.). However, the command `ls -R` has been modified to make the assignment easier, and you are not expected to move up directories or change directories for absolute and relative paths with the `cd` command. You are encouraged to try these commands on a live bash terminal if you have access to get a feel for how they should work.

- `public static void main(String[] args)`
  - *Brief:*
    - Runs a program which takes user input and builds a `DirectoryTree` using the commands indicated above.

EXTRA CREDIT - OPTIONAL

For extra credit, you may include the following features in your submission. These parts are NOT REQUIRED to recieve full credit on your assignment; however, they may be included in your submission if you wish to attempt them.

- Include the following additional commands in your `BashTerminal`:

Command	Description
`find {name}`	Finds the node in the tree with the indicated name and prints the path.
`cd ..`	Moves the cursor up to its parent directory (does nothing at root). (e.g. `cd root/home/user/Documents`)
`cd {path}`	Moves the cursor to the directory with the indicated **path**. (e.g. `cd root/home/user/Documents`)
`mv {src} {dst}`	Moves a file or directory specified by src to dst, including all children. (Note that `src` and `dst` are **absolute** paths).

- Instead of using a ternary (3-child) tree structure, implement `DirectoryTree` so that any node may contain up to 10 child references. If you attempt this, try to avoid having 10 member variables for each `DirectoryNode` (i.e. `child1, child2, child3` etc.). What would be a better way to handle an arbitrary number of child references?

Sample Input/Output:

NOTE: When prompting user input, you should indicate the name of the user and the host being accessed in the following format:

```
[user@host]: $ // Waiting for command.
```

To assist with grading, please replace 'user' with your own netID. You may feel free to name your host whatever you would like, as long as it is appropriate. Computer Science puns and obscure references will be appreciated.

// Comment in green, input in red, output in black

```
// General use.
Starting bash terminal.
[user@host]: $ pwd
root
[user@host]: $ mkdir dev
[user@host]: $ mkdir home
[user@host]: $ mkdir bin
[user@host]: $ ls
dev home bin
[user@host]: $ cd dev
[user@host]: $ pwd
root/dev
[user@host]: $ touch ttys0
[user@host]: $ touch ttys1
[user@host]: $ ls
ttsy0 ttsy1
[user@host]: $ cd /
[user@host]: $ pwd
root
[user@host]: $ cd bin
[user@host]: $ touch sublime
[user@host]: $ touch gcc
[user@host]: $ cd /
[user@host]: $ ls -R

|- root             // Note directories begin with '|-'
    |- dev
        - ttys0     // Note files begin with '-'
        - ttys1
    |- home
    |- bin
        - sublime
        - gcc

[user@host]: $ cd home
[user@host]: $ mkdir user
[user@host]: $ cd user
[user@host]: $ pwd
root/home/user
[user@host]: $ mkdir Documents
[user@host]: $ mkdir Pictures
[user@host]: $ mkdir Downloads
[user@host]: $ cd Documents
[user@host]: $ touch hw5.java
[user@host]: $ touch resume.pdf
[user@host]: $ ls
hw5.java resume.pdf
[user@host]: $ cd /
[user@host]: $ cd home
[user@host]: $ cd user
[user@host]: $ cd Pictures
[user@host]: $ touch puppies.jpg
[user@host]: $ cd /
[user@host]: $ ls -R

|- root
    |- dev
        - ttys0
        - ttys1
    |- home
        |- user
            |- Documents
                - hw5.java
                - resume.pdf
            |- Pictures
                - puppies.jpg
            |- Downloads
    |- bin
        - sublime
        - gcc

[user@host]: $ cd home
[user@host]: $ cd user
[user@host]: $ pwd
root/home/user
[user@host]: $ ls -R

|- user
    |- Documents
        - hw5.java
        - resume.pdf
    |- Pictures
        - puppies.jpg
    |- Downloads

[user@host]: $ exit
Program terminating normally


// Special cases.
```

```
...
[user@host]: $ pwd
root/dev
[user@host]: $ ls -R

|- dev
     - ttys0
     - ttys1
     - ttys2

[user@host]: $ touch ttys3
ERROR: Present directory is full.
[user@host]: $ cd ttys2
ERROR: Cannot change directory into a file.
[user@host]: $ cd nonexistantDirectory
ERROR: No such directory named 'nonexistantDirectory'.
[user@host]: $ exit
Program terminating normally


// EXTRA CREDIT - NOT REQUIRED.
...
[user@host]: $ pwd
root
[user@host]: $ ls -R

|- root
    |- dev
         - ttys0
         - ttys1
    |- home
        |- user
            |- Documents
                - hw5.java
                |- myFolder
                     - file1.txt
                     - file2.txt
                     - file3.txt
            |- Pictures
                 - puppies.jpg
    |- tmp
         - puppies.jpg

[user@host]: $ find puppies.jpg
root/home/user/Pictures/puppies.jpg
root/tmp/puppies.jpg
[user@host]: $ find kittens.jpg
ERROR: No such file exits.
[user@host]: $ cd home/user // Note cd with path.
[user@host]: $ pwd
root/home/user
[user@host]: $ cd .. // Move up to parent.
[user@host]: $ pwd
root/home
[user@host]: $ cd ..
[user@host]: $ pwd
root
[user@host]: $ cd ..
ERROR: Already at root directory.
[user@host]: $ pwd
root
[user@host]: $ mv root/home/user/Documents/myFolder root/tmp // Note absolute paths.
[user@host]: $ ls -R

|- root
    |- dev
         - ttys0
         - ttys1
    |- home
        |- user
            |- Documents
                 - hw5.java
            |- Pictures
                 - puppies.jpg
    |- tmp
        - puppies.jpg
        |- myFolder
             - file1.txt
             - file2.txt
             - file3.txt

[user@host]: $ exit
Program terminating normally
```
