/\*

\* jshell.java

\* CS575\_japarker\_program1

\* Jefferson Parker

\* 20180220

\*/

**import** java.io.\*;

**import** java.util.ArrayList;

**import** java.util.ListIterator;

**import** java.util.Scanner;

**public** **class** jshell {

// Create private class variables.

**private** **static** ArrayList<String> *history* = **new** ArrayList<String>(); // Command history list.

**private** **static** String *nextArg*; // User input argument.

// Integer values to track command 'Types' as constants.

**private** **static** **final** **int** ***historyCommand*** = 1;

**private** **static** **final** **int** ***changeDirCommand*** = 2;

**private** **static** **final** **int** ***backCommand*** = 3;

**private** **static** **final** **int** ***multiCommand*** = 4;

**private** **static** **final** **int** ***pipeCommand*** = 5;

**private** **static** **final** **int** ***regCommand*** = 6;

// Method to implement an instance of jshell program from command line.

**public** **static** **void** main(String[] args) {

// Creat Scanner object to read keyboard input.

**final** Scanner keyboard = **new** Scanner (System.***in***);

// Prompt the user for input.

System.***out***.println("Welcome to jshell.");

System.***out***.println("jshell can manage all of your shell command needs.");

System.***out***.println("Type HISTORY to see last 10 commands.");

System.***out***.println("Type EXIT to exit.");

//Loop through input until EXIT command is given.

**do** {

// Print the shell prompt, get user input and process the argument.

System.***out***.print("jshell:>");

*nextArg* = keyboard.nextLine().trim();

**int** commType = *getCommandType*(*nextArg*);

**switch**(commType) {

**case** ***historyCommand***: *printHistory*(); **break**;

**case** ***changeDirCommand***: *changeDirectory*(*nextArg*); **break**;

**case** ***backCommand***: *runBackgroundProcess*(*nextArg*); **break**;

**case** ***multiCommand***: *runMultipleProcess*(*nextArg*); **break**;

**case** ***pipeCommand***: *runPipeCommand*(*nextArg*); **break**;

**case** ***regCommand***: *runProcess*(*nextArg*); **break**;

}

} **while** (!*nextArg*.equalsIgnoreCase("EXIT"));

// Cleanup on exit.

keyboard.close();

System.***out***.println("Good Bye.");

}

// Catchall exceptions

**private** **static** **void** catchExceptions (Exception e) {

e.notify();

System.***out***.println("oops, so sorry");

}

/\*

\* The changeDirectory method does not work, and it is not advised to try it in Java.

\*/

// Change current working directory.

// From www.javacodex.com/Files/Set-The-Current-Working-Directory

**private** **static** **void** changeDirectory(String arg) {

// Parse the target directory.

String targetDir = **null**;

// Handle different case options.

// From https://stackoverflow.com/questions/5054995/how-to-replace-case-insensitive-literal-substrings-in-java

**if**(arg.toUpperCase().contains("CD ")) {targetDir = arg.replaceAll("(?i)cd ","");}

// Test if targetDir exists. If not, send a message.

// From https:\\stackoverflow.com/questions/12780446/check-if-a-path-represents-a-file-or-a-folder/12780471

File file = **new** File(targetDir);

**if**(file.isDirectory()) {

**try**{Process proc = Runtime.*getRuntime*().exec(**new** String[] {"cd", targetDir});}

**catch**(Exception e) {*catchExceptions*(e);}

}

// Add the command to the shell history.

*history*.add(arg);

}

// Parse the input argument to a class specific type.

**private** **static** **int** getCommandType (String arg) {

// System.out.println("Getting command type for " + arg);

**if** (arg.equalsIgnoreCase("EXIT")) {**return** (0);}

**else** **if**(arg.equalsIgnoreCase("HISTORY")) {**return** (***historyCommand***);}

**else** **if**(arg.toUpperCase().contains("CD")) {**return** (***changeDirCommand***);}

**else** **if**(arg.contains(";")) {**return** (***multiCommand***);}

**else** **if**(arg.contains("|")) {**return** (***pipeCommand***);}

**else** **if**(arg.contains("&")) {**return** (***backCommand***);}

**else** {**return** (***regCommand***);}

}

// Print last ten, or fewer, shell commands.

**private** **static** **void** printHistory() {

// Print the last ten items (or less) from the history list.

**if**(*history*.size() >= 10) {

ListIterator<String> histIterator = *history*.listIterator(*history*.size() - 10);

**while** (histIterator.hasNext()) {

System.***out***.println(histIterator.next());

}

}

**else** {

ListIterator<String> histIterator = *history*.listIterator();

**while** (histIterator.hasNext()) {

System.***out***.println(histIterator.next());

}

}

}

// Execute the process, don't wait for it to complete.

**private** **static** **void** runBackgroundProcess (String arg) {

**try**{Process proc = Runtime.*getRuntime*().exec(**new** String[] {"/bin/sh", "-c", arg});}

**catch**(Exception e) {*catchExceptions*(e);}

// System.out.println("In background process " + arg);

// Add the command to the shell history.

*history*.add(arg);

}

**private** **static** **void** runMultipleProcess (String arg) {

// Split the method input into a String array of commands.

String [] argumentList = arg.split(";");

// Execute each command.

**for** (String nextMultiArg : argumentList) {

nextMultiArg = nextMultiArg.trim();

// System.out.println("In multi command process " + nextMultiArg);

**int** commType = *getCommandType*(nextMultiArg);

**switch**(commType) {

**case** ***historyCommand***: *printHistory*(); **break**;

**case** ***changeDirCommand***: *changeDirectory*(nextMultiArg); **break**;

**case** ***backCommand***: *runBackgroundProcess*(nextMultiArg); **break**;

**case** ***pipeCommand***: *runPipeCommand*(nextMultiArg); **break**;

**case** ***regCommand***: *runProcess*(nextMultiArg); **break**;

}

}

}

/\*

\* The runPipeCommand method is not functional.

\* The problem is in the BufferedReader that gets the output of Process 2.

\* The Reader is not reading, and it is not stopping, when their is null output.

\*

\* NOTE, the pipe command works properly when passed to runProcess().

// From https://www.developer.com/java/data/understanding-java-process-and-java-processbuilder.html

// From https://stackoverflow.com/questions/5987970/socket-bufferedreader-hangs-at-readline

private static void runPipeCommand (String arg) {

// Parse arg into pre and post pipe commands, trimming any white space.

String arg1 = arg.substring(0, arg.indexOf("|")).trim();

String arg2 = arg.substring(arg.indexOf("|") +1, arg.length()).trim();

// Create a temporary file to transfer the output of process 1 to input of process 2.

// If an existing copy of the file is present, delete it.

File pipeTempFile = new File("pipeTemp");

OutputStream pipeTempOut = null;

try {

if(!pipeTempFile.exists()) {pipeTempFile.createNewFile();}

else{

pipeTempFile.delete();

pipeTempFile.createNewFile();

}

pipeTempOut = new FileOutputStream(pipeTempFile);

}

catch(Exception e) {catchExceptions(e);}

// Execute Process 1 (the Pre-pipe process) and write the output to a file.

try{

System.out.println("In pipe command1 " + arg1);

Process proc1 = Runtime.getRuntime().exec(new String[] {"sh", "-c", arg1});

BufferedReader procRead1 = new BufferedReader (new InputStreamReader (proc1.getInputStream()));

while(procRead1.readLine() != null) {pipeTempOut.write((procRead1.readLine() + "\n").getBytes());}

proc1.destroy();

pipeTempOut.flush();

pipeTempOut.close();

}

catch(Exception e) {catchExceptions(e);}

// Execute Process 2 using the output file of Process 1 as input.

try{

System.out.println("In pipe command2 " + arg2);

Process proc2 = Runtime.getRuntime().exec(new String[] {"sh", "-c", arg2, "pipeTemp"});

System.out.println("Executed command");

BufferedReader procRead2 = new BufferedReader(new InputStreamReader (proc2.getInputStream()));

System.out.println("Created Buffered Reader");

//Trying something different

String procOut2 = "";

while((procOut2 = procRead2.readLine()) != null){System.out.println(procOut2);}

procRead2.close();

proc2.destroy();

pipeTempFile.delete();

}

catch(Exception e) {catchExceptions(e);}

// Add the command to the shell history.

history.add(arg);

}

\*/

// Functional workaround for runPipeCommand.

**private** **static** **void** runPipeCommand (String arg) {

*runProcess*(arg);

}

// Execute the process.

// From https://coderanch.com/t/538967/java/Running-commands-Runtime-getRuntime-exec

// From https://stackoverflow.com/questions/3403226/how-to-run-linux-commands-in-java-code

**private** **static** **void** runProcess (String arg) {

Process proc = **null**;

String procOut;

**try**{

proc = Runtime.*getRuntime*().exec(**new** String[] {"sh", "-c", arg});

BufferedReader procRead = **new** BufferedReader(**new** InputStreamReader (proc.getInputStream()));

**while**((procOut = procRead.readLine()) != **null**) {System.***out***.println(procOut);}

}

**catch**(Exception e) {*catchExceptions*(e);}

proc.destroy();

// System.out.println("In regular process " + arg);

// Add the command to the shell history.

*history*.add(arg);

}

}