Cairo University
Faculty of Computers & Artificial Intelligence
Computer Science Department
Operating Systems 1 Course

Assignment 1 – Command Line Interpreter

In this assignment, you will write a ***Command Line Interpreter (CLI)*** for your operating system.

Your CLI should allow the user to enter the input through the keyboard. After the user writes the command and presses enter, the string is parsed, and the indicated command executed.

The CLI will keep accepting different commands from the user until the user writes "exit", then the CLI terminates.

Your program structure and the list of required commands are listed below.

Program Structure:

Your program should contain **2** major classes: ***Parser & Terminal.***

```
class Parser {
   String commandName;
   String[] args;

  //This method will divide the input into commandName and args
  //where "input" is the string command entered by the user
   public boolean parse(String input){...}

   public String getCommandName(){...}

   public String[] getArgs(){...}
}
```

```
public class Terminal {
   Parser parser;

   //Implement each command in a method, for example:
   public String pwd(){...}
   public void cd(String[] args){...}
   // ...


   //This method will choose the suitable command method to be called
   public void chooseCommandAction(){...}

   public static void main(String[] args){...}
}
```

Required Commands:

Command Name	*What You Must Implement*
pwd	Takes **no arguments** and prints the current path.
cd	Implement all these cases: 1. **cd** takes **no arguments** and changes the current path to the path of your home directory. 2. **cd** takes **1 argument** which is ".." (e.g. **cd ..**) and changes the current directory to the previous directory. 3. **cd** takes **1 argument** which is either the **full path or the relative (short) path** and changes the current path to that path.
ls	Takes **no arguments** and lists the contents of the current directory sorted alphabetically.
mkdir	Takes **1 or more arguments** and creates a directory for each argument. Each argument can be: • **Directory name** (in this case the new directory is created in the current directory) **Path** (full/short) that ends with a directory name (in this casethe new directory is created in the given path)
rmdir	Implement all these cases: 1. **rmdir** takes **1 argument** which is "*" (e.g. **rmdir ***) and removes all the **empty directories** in the current directory. • **rmdir** takes **1 argument** which is either the **full path or the relative (short) path** and removes the given directory **only if it is empty.**
touch	**2.** Takes **1 argument** which is either the **full path or the relative (short) path** that ends with a file name and creates this file.
cp	Takes **2 arguments**, both are files and copies the first onto the second.
cp -r	Takes **2 arguments**, both are directories (empty or not) and copies the first directory (with all its content) into the second one.

rm	Takes **1 argument** which is a file name that exists in the current directory and removes this file.
cat	Takes **1 argument** and prints the file's content or takes **2 arguments** and concatenates the content of the 2 files and prints it.

wc	Wc stands for "word count," and as the name suggests, it is mainly used for counting purpose. By default, it displays **four-columnar output.**
	First column shows **number of lines** present in a file specified, second column shows **number of words** present in the file, third column shows **number of characters** present in file and fourth column itself is the **file name** which are given as argument
	Example:
	`wc file.txt`
	Output:
	`9 79 483 file.txt`
	Explanation:
	`# 9 lines, 79 word, 483 character with spaces, file name`

>	Format: ***command > FileName*** Redirects the output of the first command to be written to a file. If the file doesn't exist, it will be created. If the file exists, its original content will be **replaced.** Example: `echo Hello World > myfile.txt ls > file`
>>	Like > but **appends** to the file if it exists.
`zip`	zip command (Compressing files and directories): • Basic usage: To compress one or more files into a new .zip archive: Example: zip archive_name.zip file1.txt file2.jpg This creates archive_name.zip containing file1.txt and file2.jpg. • Compressing a directory recursively: To compress an entire directory and its contents: Example: zip -r archive-name.zip directory-to-compress/ The -r option ensures that subdirectories and their files are included in the archive.
unzip	• Basic usage: To extract all files from a .zip archive into the current directory: Example: unzip archive-name.zip • Extracting to a specific directory: To extract the contents to a different location: Example: unzip archive-name.zip -d /path/to/destination/

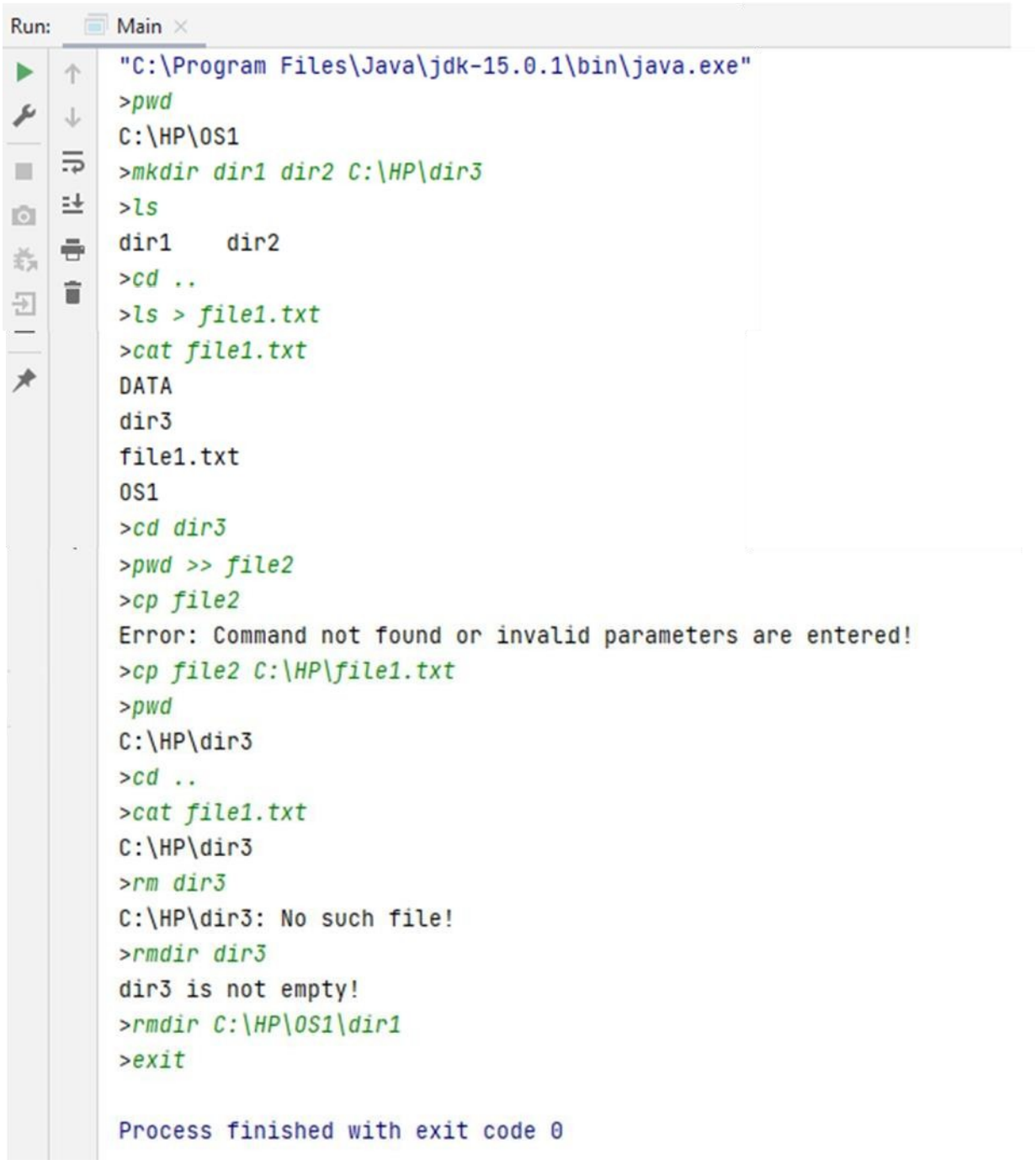
Notes:

- You must implement the ***"exit"*** command which will allow the CLI to terminate.
- If the user enters a ***wrong command or bad parameters*** (invalid path, file instead of directory in certain commands, etc.), **the program should print some error messages without terminating.**
- You must handle all the mentioned cases in each command.
- You can refer to the lab document for further information on the commands.

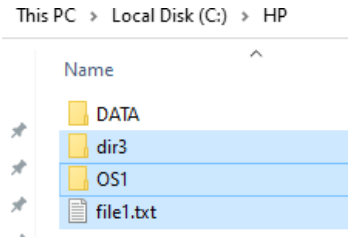
Assignment Rules:

- Your program should be written in ***Java.***
- You can add more attributes/methods, but ***you can't change the given classes' structure, or you will lose 1 mark (from 6).***
- You can use built-in functions and predefined classes in Java. ***Do not use "exec"*** to implement any of these commands or you will lose marks.

Sample Input & Output:



Files and folders must be manipulated on your OS.



Submission Rules:

- You must submit ***only one ".java" file*** containing the source code (2 classes).
- The submitted file name must follow this format: **ID1_ID2_ID3_Group** (e.g. 20210000_20210001_20210002_DS1)
- The assignment is submitted in groups of ***5*** students.
- The assignment submission will be on **Classroom** (You are not allowed to send your assignment by email).
- ***Cheating*** is totally prohibited and won't be tolerated (any similarity between your code and any other source will be assigned **NEGATIVE** without argument).

Grading Criteria:

Parsing commands and choosing the command action	**5 marks**
Handling short paths and full paths	**4 mark**
(1 mark for each implemented command)	**15 marks**
Total **24 marks**	

Note: The assignment grade will be scaled to 6 marks.