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: ***(You will choose only 12 commands to implement)***

Command Name	***What You Must Implement***
echo	Takes **1 argument** and prints it.
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.
ls -r	Takes **no arguments** and lists the contents of the current directory in **reverse order.**
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 case the 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. 2. **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	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.

>	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.

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:

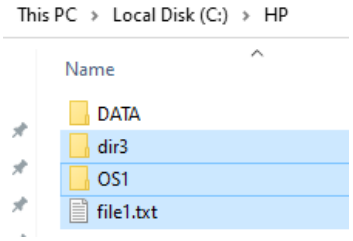
- Your program should be written in ***Java.***
- You can add more attributes/methods, but ***you can't change the given classes' structure, or you will lose 1 mark (from 5).***
- You can use built-in functions and predefined classes in Java. ***Do not use "exec"*** to implement any of these commands or you will lose marks.

Sample Input & Output:

```
Run:  Main
"C:\Program Files\Java\jdk-15.0.1\bin\java.exe"
>pwd
C:\HP\OS1
>mkdir dir1 dir2 C:\HP\dir3
>ls
dir1    dir2
>cd ..
>ls > file1.txt
>cat file1.txt
DATA
dir3
file1.txt
OS1
>cd dir3
>pwd >> file2
>cp file2
Error: Command not found or invalid parameters are entered!
>cp file2 C:\HP\file1.txt
>pwd
C:\HP\dir3
>cd ..
>cat file1.txt
C:\HP\dir3
>rm dir3
C:\HP\dir3: No such file!
>rmdir dir3
dir3 is not empty!
>rmdir C:\HP\OS1\dir1
>exit

Process finished with exit code 0
```

****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. 20190000_20190001_20190002_DS1)
- The assignment is submitted in groups of ***maximum 3*** students **(the minimum is 2).**
- The deadline will be announced on Blackboard and ***no late submission*** is allowed.
- The assignment submission will be on **Blackboard** (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	**2 marks**
Handling short paths and full paths	**1 mark**
Choose and implement 12 commands *(1 mark for each implemented command)*	**12 marks**
Total **15 marks**	

Note: The assignment grade will be scaled to 5 marks.