Project Report

Group 15

COMP2021 Object-Oriented Programming (Fall 2020)

1. **KONG, Siu Chun (19071318d)**
2. **NG, Tsz Hin (19053208d)**
3. **YEUNG, Lai Wing (19061722d)**
4. **CHAN, Pak Hin (19071928d)**

1 Introduction

This document describes the design and implementation of the Comp Virtual File System (CVFS) by group 15. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

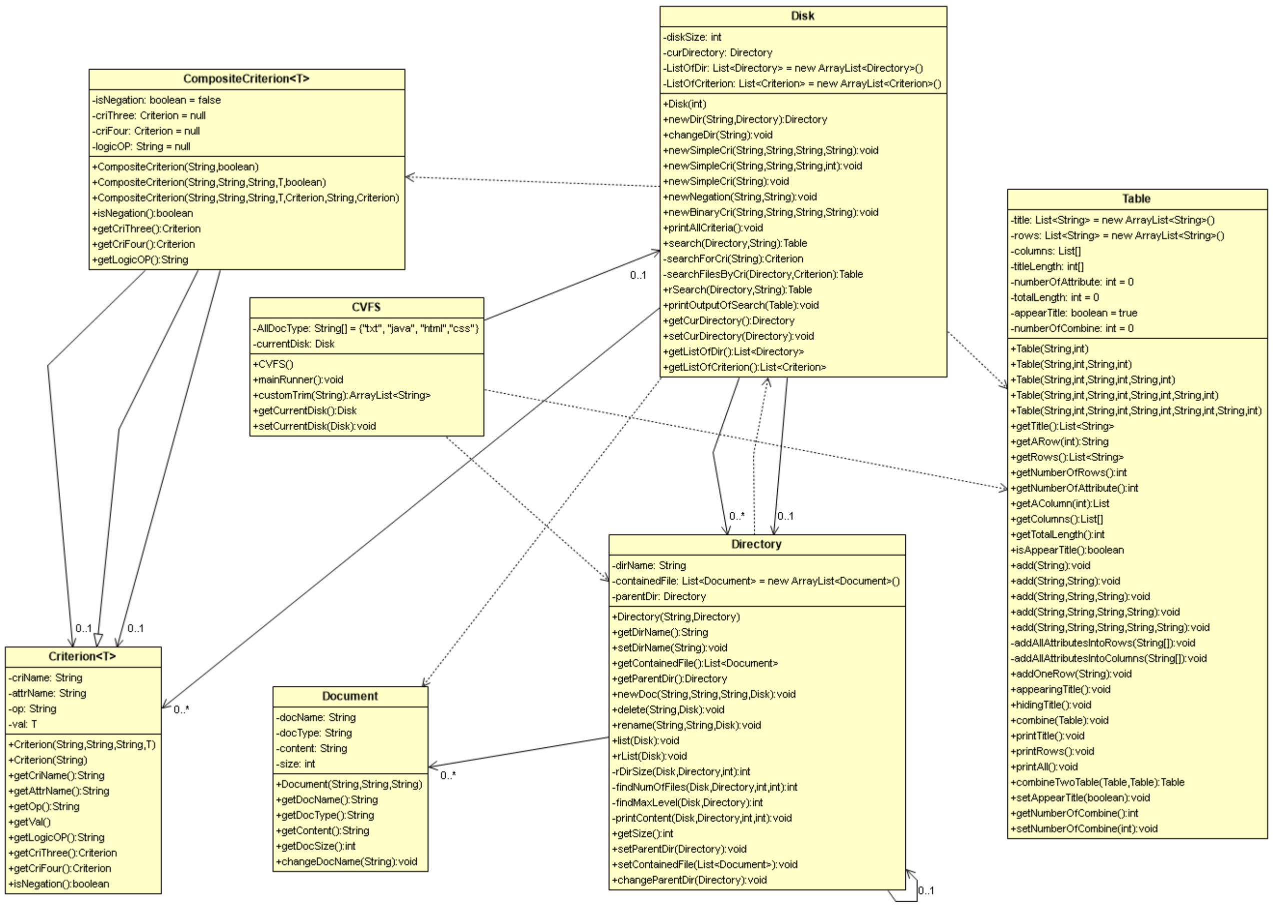
The main goal of this project is to develop an in-memory Virtual File System (VFS), named the Comp VFS (CVFS). The system will be built using java. The system will provide a commend line interface (CLI) for user to interact with the file system.

With varies commands implemented in the CVFS command line interface (CLI), the user can perform different actions to disk, directories, files and criteria, like creating a new disk, deleting a file, searching files based on a criterion. Once the command(s) input by the user is/are invalid, the error message will be printed out and remind the user with a correct format for the command he/she may want to execute. Finally, the user can set up the Comp VFS (CVFS) by using this CLI.

2 The Comp Virtual File System (CVFS)

The following describes the overall design of the CVFS and then the implementation details of the requirements.

2.1 Design


Disk
-diskSize: int
-curDirectory: Directory
-ListOfDir: List<Directory> = new ArrayList<Directory>()
-ListOfCriterion: List<Criterion> = new ArrayList<Criterion>()
+Disk(int)
+newDir(String,Directory):Directory
+changeDir(String):void
+newSimpleCri(String,String,String,String):void
+newSimpleCri(String,String,String,int):void
+newSimpleCri(String):void
+newNegation(String,String):void
+newBinaryCri(String,String,String,String):void
+printAllCriteria():void
+search(Directory,String):Table
-searchForCri(String):Criterion
-searchFilesByCri(Directory,Criterion):Table
+rSearch(Directory,String):Table
+printOutputOfSearch(Table):void
+getCurDirectory():Directory
+setCurDirectory(Directory):void
+getListOfDir():List<Directory>
+getListOfCriterion():List<Criterion>
CompositeCriterion<T>
-isNegation: boolean = false
-criThree: Criterion = null
-criFour: Criterion = null
-logicOP: String = null
+CompositeCriterion(String,boolean)
+CompositeCriterion(String,String,String,T,boolean)
+CompositeCriterion(String,String,String,T,Criterion,String,Criterion)
+isNegation():boolean
+getCriThree():Criterion
+getCriFour():Criterion
+getLogicOP():String
CVFS
-AllDocType: String[] = {"txt", "java", "html","css"}
-currentDisk: Disk
+CVFS()
+mainRunner():void
+customTrim(String):ArrayList<String>
+getCurrentDisk():Disk
+setCurrentDisk(Disk):void
Table
-title: List<String> = new ArrayList<String>()
-rows: List<String> = new ArrayList<String>()
-columns: List[]
-titleLength: int[]
-numberOfAttribute: int = 0
-totalLength: int = 0
-appearTitle: boolean = true
-numberOfCombine: int = 0
+Table(String,int)
+Table(String,int,String,int)
+Table(String,int,String,int,String,int)
+Table(String,int,String,int,String,int,String,int)
+Table(String,int,String,int,String,int,String,int,String,int)
+getTitle():List<String>
+getARow(int):String
+getRows():List<String>
+getNumberOfRows():int
+getNumberOfAttribute():int
+getAColumn(int):List
+getColumns():List[]
+getTotalLength():int
+isAppearTitle():boolean
+add(String):void
+add(String,String):void
+add(String,String,String):void
+add(String,String,String,String):void
+add(String,String,String,String,String):void
-addAllAttributesIntoRows(String[]):void
-addAllAttributesIntoColumns(String[]):void
+addOneRow(String):void
+appearingTitle():void
+hidingTitle():void
+combine(Table):void
+printTitle():void
+printRows():void
+printAll():void
+combineTwoTable(Table,Table):Table
+setAppearTitle(boolean):void
+getNumberOfCombine():int
+setNumberOfCombine(int):void
Directory
-dirName: String
-containedFile: List<Document> = new ArrayList<Document>()
-parentDir: Directory
+Directory(String,Directory)
+getDirName():String
+setDirName(String):void
+getContainedFile():List<Document>
+getParentDir():Directory
+newDoc(String,String,String,Disk):void
+delete(String,Disk):void
+rename(String,String,Disk):void
+list(Disk):void
+rList(Disk):void
-rDirSize(Disk,Directory,int):int
-findNumOfFiles(Disk,Directory,int,int):int
-findMaxLevel(Disk,Directory):int
-printContent(Disk,Directory,int,int):void
+getSize():int
+setParentDir(Directory):void
+setContainedFile(List<Document>):void
+changeParentDir(Directory):void
Criterion<T>
-criName: String
-attrName: String
-op: String
-val: T
+Criterion(String,String,String,T)
+Criterion(String)
+getCriName():String
+getAttrName():String
+getOp():String
+getVal()
+getLogicOP():String
+getCriThree():Criterion
+getCriFour():Criterion
+isNegation():boolean
Document
-docName: String
-docType: String
-content: String
-size: int
+Document(String,String,String)
+getDocName():String
+getDocType():String
+getContent():String
+getDocSize():int
+changeDocName(String):void
0..1
0..*
0..1
0..1
0..1
0..*
0..*
0..1

The full resolution of the diagram is in the project folder.

2.1 Requirement

[REQ1] **newDisk** *diskSize*

1) The requirement is implemented.

2) First, we will check whether the user input enough information or not by splitting and counting the length of the input. Then, we will check whether the disk size inputted by the user ("diskSize") is integer or not. Next, we will assign the current disk in the CVFS class to the new disk class created at that moment and pass the "diskSize" value to the Disk method in the Disk Class.

With revoking the Disk method in the Disk Class, the maximum disk size ("diskSize") in our "Disk" class will be set/reset to the value passed from the CVFS class. If the current directory ("curDirectory") equals to nothing (null), we will create a newDirectory class by using the newDir method in the Disk class with assigning dirName as root and parentDir as null. Finally, "newDirectory" will be the "curDirectory". If the current directory ("curDirectory") does not equal to nothing (null) which means that there is a directory linked to the current directory, we will clear the parent directory ("parentDir") for it by setting the "parentDir" as null.

3) There are two error conditions. First of all, there is an error if the user input too many parameters which is also known as more information than needed, we will classify as invalid input and print the "Invalid input – too many parameter" error message. We detect the error by using an if statement. Secondly, there is an error if the disk size ("diskSize") input by the user is not an integer. We detect the error by using the try and catch method with checking whether we can assign the "diskSize" input by the user to the int variable. Once the operation is unable to execute, we will treat it as the invalid input and print the error message of "Invalid input – parameter is not integer."

[REQ2] **newDoc** *docName docType docContent*

1) The requirement is implemented.

2) The "newDoc" method is used to create documents under the current working directory. It will create the specific file according to the user input which are the thee parameter that get passed in during the process of evoking the method, then an object of document will be created. Furthermore, for the parameter of the "docCentent" we allow user to use " " to embrace the desire content which allow them to have space in the content of the document.

3) The method will check if the correct number of parameters are entered. Then, the type and requirement of the parameters will be checked. For example, the "doctype" must be any of the following (java, css, txt, html), otherwise error will be shown. Also, the method will check if the "docName" user enter is compile to the requirements, which is only English, and numbers and the length of the name must be less than or equal to 10.

[REQ3] **newDir** *dirName*

1) The requirement is implemented.

2) This method enable user to create a new directory under the current working directory of user choice. The method will accept only one parameter which is the Name of the directory that is going to be created.

3)In the beginning of the method, we will check if the parameter inputted is correct, which including the checking of, if the number of parameters enter is enough, if the directory name only contains English letter and numbers, also if there already consist another directory with the same name error show upon to the user.

[REQ4] **delete** *fileName*

1) The requirement is implemented.

2) The delete method allow user to delete a file in the working directory simply by entering the file name of the file which desired to delete.

3) The method will check if the parameter entered is valid base on the length and the type of the character, then the method will further if the entered file name exists in the working directory, if no file with the same name found, the method will return error.

[REQ5] **rename** *oldFileName newFileName*

1) The requirement is implemented.

2) The rename method will handle two parameter which is oldFileName and newFileName. It will search the current directory for the oldFileName which is the name the user wants to be changed, after the specific document found, the method will change the file name to the newFileName that user inputted earlier.

3)The method will firstly, if the parameter is compiled to the desired type and without space, then it will check if the oldFileName's file exist, if not error will be shown. In another case, if the file is found but the newFileName that the user entered earlier already exist in the same directory, error will also be shown.

[REQ6] **changeDir** *dirName*

1) The requirement is implemented.

2) This method will allow user to change the current working directory. There are two cases, first is the case of user going into sub-folders which is originally in the working directory, the other case is that when user input ".." for the directory, the method will help the user to change the working directory back up a level.

3) First the method will check the basic requirement of the parameters entered are valid or not, then in the first case of going further into sub-directory, the method will scan the working directory to check if the directory user inputted exists, if not error will be shown. In the other case of going back up aa level the method will check if the user is already in the root directory, if yes error will be shown to notice the user that he/she is already at the root directory.

[REQ7] **list**

1) The requirement is implemented.

2) This method will list out the file name, file type, file size, and directory in the working directory in a table format. The method will loop the entire disk to search for the directories which parent directory is the working directory and print it out. After printing all the directory under the working directory, the method will look into the file contained in the working directory and print it out. Lastly, the method will also report the total number of files and total size of the file listed above.

3) There are no error cases in this method, since no matter in which directory the list of content can also be printed out, even the working directory is empty.

[REQ8] **rList**

1) The requirement is implemented.

2) This method is very similar to the above method (REQ 7) which the difference is when the list method reaches another directory the method will only print it our but will not go into the sub-directory to print the content inside while the rList method will. In order to achieve the above requirement, we used a recursive method to do so. In simple words, when the method encounters another directory under the same working directory it will re-run the method of printing the content in the sub-directory with indentation when going in each level.

3) Similar to (REQ7) this method will not encounter error case, since in any directory the list of content would also be able to print out.

[REQ9] **newSimpleCri** *criName attrName op val*

1) The requirement is implemented.

2) First, we will check whether the user input enough parameters or not by splitting and counting the length of the input. Then, we will check whether the criteria name ("criName") input by the user is exactly 2 characters or not, whether both of the characters in the criteria name ("criName") are letters or not and whether the attribute name ("attrName") input by the user is either name / type or size or not. For each case, we will check whether the op and val can comply with the related requirements. Once the input passes all the tests, we will pass 4 parameters of the user input to the newSimpleCri method in the Disk Class individually.

With revoking the newSimpleCri method in the Disk Class, we will check whether the "criName" input by the user is used before. If there is no duplicate "criName", we will pass the parameters required to the Criterion method in the Criterion method for creating this new criterion and add the criterion formed by the Criterion method in the Criterion class to the ListOfCriterion in the Disk class.

3) There are in total twelve cases which could lead to an error occurs.

The first case is that we will classify as invalid input and print the "Invalid input – too many parameter" error message if the user input too many parameters which is also known as more information than needed. We treat the input as invalid input and detect the error by using an if statement. The second case is that we will classify as invalid input and print the "Invalid input – too few parameter" error message if the number of parameters input by the user is less than the parameters required which is also known as user input less information than needed. We treat the input as invalid input and detect the error by using an else if statement. The third case is that the number of the characters of the "criName" input by the user is not equal to 2. As a result, we will print out the error message of "Invalid input - criName is not two letters". We detect the error by using an if statement. The fourth case is that the error message of "Invalid input - criName is not two letters" will be printed out if the one/both of the character(s) "criName" input by the user is/are not letter(s). We detect the error by using an if statement. The fifth case is that the "attrName" input by the user is not either name, type or size. Then, the error message of "Invalid input - Incorrect attrName" must be obtained if attrName = name" will be printed out. We treat the input as invalid input and detect the error by using an if statement.

If the “attrName” input by the user is “name”, two errors may happen. Firstly, the error message of "Invalid input - Incorrect op. \"contains\" must be obtained if attrName = name" will be printed out if the “op” input by the user is not “contains”. We treat the input as invalid input and detect the error by using an if statement. Secondly, the error message of "Invalid input - Incorrect val. val must be a string and in double quote" will be printed out if the “val” input by the user is not a string and/or it is not in the form of double quote. We treat the input as invalid input and detect the error by using an if statement.

If the “attrName” input by the user is “type”, two errors may happen. Firstly, the error message of "Invalid input - Incorrect op. \"equals\" must be obtained if attrName = type" will be printed out if the “op” input by the user is not “equals”. We treat the input as invalid input and detect the error by using an if statement. Secondly, the error message of "Invalid input - Incorrect val. val must be a string and in double quote" will be printed out if the “val” input by the user is not a string and/or it is not in the form of double quote. We treat the input as invalid input and detect the error by using an if statement.

If the “atttrName” input by the user is “size”, two errors may happen. Firstly, the error message of "Invalid input - Incorrect op. \">\", \"<\", \">=\", \"<=\", \"==\", \"!=\" must be obtained if attrName = size" will be printed out if the “op” input by the user is not either ">"/ "<"/ ">="/ "<="/ "==" or "!=". We treat the input as invalid input and detect the error by using an if statement. Secondly, the error message of "Invalid input - val is not integer." will be printed out if the “op” input by the user is not an integer. We treat the input as invalid input and detect the error by using the try and catch method.

Finally, we will check whether the “criName” input is used before by using a for loop for looping the “ListOfCriterion” and an if statement. If the “criName” is used before, we will throw a new IllegalArgumentException().

[REQ10] **IsDocument**

1) The requirement is implemented.

2) In the beginning, we will pass the criteria name (“criName”) which is the “IsDocument” to the newSimpleCri method in the Disk class.

With revoking the newSimpleCri method in the Disk Class, we will check whether the “criName” input by the user is used before. If there is no duplicate “criName”, we will pass the parameters required to the Criterion method in the Criterion method for creating this new criterion and add the criterion formed by the Criterion method in the Criterion class to the ListOfCriterion in the Disk class.

3) There are 2 cases which leads to an error. Firstly, we will throw a new IllegalArgumentException() if the “IsDocument” is used as the “criName” before. We detect the error by using an if statement. Secondly, we will throw a new IllegalArgumentException() if the “criName” input by the user is not “IsDocument”. We detect the error by using an if statement.

[REQ11] **newNegation** *criName1 criName2* **newBinaryCri** *criName1 criName3 logicOp criName4*

1) The requirement is implemented.

2) For newNegation method, we will check whether the number of parameters equals to the number of required parameters. We create a new criterion using the CompositeCriterion method in the CompositeCriterion class to exclude the criteria in criName2. The criName of this new criterion will be criName1 and the type will be composite criterion.

For newBinaryCri method, we will check whether the number of parameters equals to the number of required parameters. There are 2 cases. First, the logicOp is &&. In this case, we will create a new criterion using the CompositeCriterion method in the CompositeCriterion class by combining those criterions in criName3 and criName4. The criName of this new criterion will be criName1 and the type will be composite criterion. The second case is that the logicOp is ||. In this case, we will create a new criterion using the CompositeCriterion method in the CompositeCriterion class by including the criterion in criName3 first and excluding those criterions in criName4. The criName of this new criterion will also be criName1 and the type will be composite criterion.

3) The following error checking apply to both newNegation method and newBinaryCri method. First, we will check the number of parameters and print the error message if needed first in the CVFS class like the first case and the second case in newSimpleCri method. Then, we will check whether the number of the characters of the "criName1" input by the user is more than 2 in the Disk class. If yes, we will print out the error message of "Error: Only 2 characters are allowed in the name. Please choose a new name.". We detect the error by using an if statement. Finally, we will check whether the "criName1" input is used before by using a for loop for looping the "ListOfCriterion" and an if statement. If the "criName1" is used before, we will print the error message of "Error: The name already exists. Please choose a new criterion name.".

[REQ12] printAllCriteria

1) The requirement is implemented.

2) For printAllCriteria method, we will check whether the number of parameters equals to the number of required parameters. Then, we will call the printAllCriteria method in the Disk class. We will first create a table using our Table method in the Table class. Then, we will loop each criterion in our ListOfCriterion array list in the Disk class. We will check whether there is "var" in each criterion. If there is no "var", we will pass "null" in that parameter ("curVal"). If there is "var", we will just change to string and pass it to the parameter ("curVal"). Then, we will check whether the class of the criterion is "CompositeCriterion". If the class of the criterion is "CompositeCriterion", we will pass the "logicOp" to the parameter ("curLogicOp"). Otherwise, we will pass null to the parameter ("curLogicOp"). We will pass all of the required parameters to the add method in the Table class for each criterion in the ListOfCriterion. Finally, we will call our printAll method in the Table class to print all of the criteria in a table format.

3) There is an error if the number of parameters input by the user does not equal to the number of required parameters, we will classify it as invalid input and print the "Invalid input - too many parameter" error message. We detect the error by using an if statement.

[REQ13] **search** *criName*

1) The requirement is implemented.

2) This function will list out the file which fulfils the requirement from *criName* and in the current working directory, it will not list any other files which is not in the working directory

3) This method will not encounter errors, since the error will be handed by the CVFS, if there is no file matches the requirement that the user inputted, the method search will not be called.

[REQ14] **rSearch** *criName*

1) The requirement is implemented.

2) This function is very similar to the search function, the major difference of the two function is that in the rSearch function it will look up all the files under the working directory including the sub-folders.

3) Similar to the above search method, pre-check will be done on the CVFS, if no matches the method will not be called.

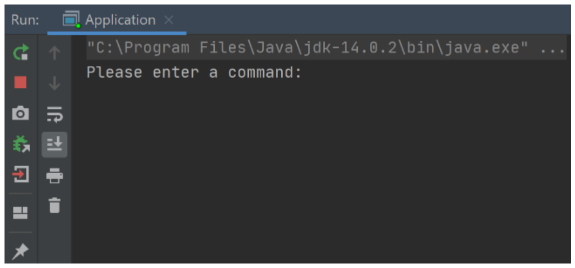
Bonus code:

1. exit
When the method of exit being called, the system will exit.

3 User Manual

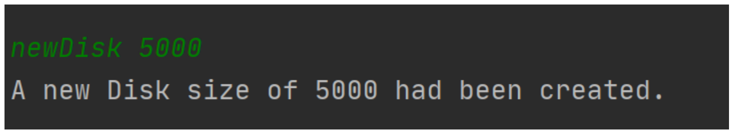
In this section, we explain how the CVFS works from a user's perspective.

When the program initializes the following screen will show up to ask for user input.



```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" ...
Please enter a command:
```

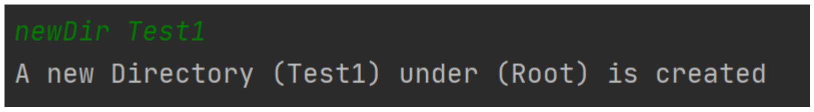
1. **newDisk** *diskSize*



```
newDisk 5000
A new Disk size of 5000 had been created.
```

This function is used to create a disk and a new working directory for the system as root directory.

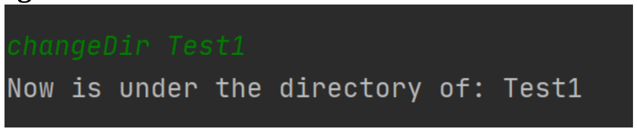
2. **newDir** *dirName*



```
newDir Test1
A new Directory (Test1) under (Root) is created
```

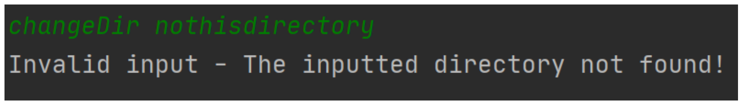
This function is used to create a directory under the current working directory with a name designed by the user. Also, the above line will be printed to notice the user that the current working directory and the newly created directory.

3. **changeDir** *dirName*



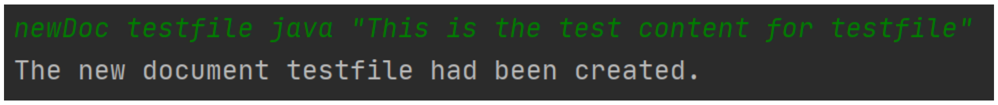
```
changeDir Test1
Now is under the directory of: Test1
```

This function can allow the user to change the current working directory to the desired location. If the dirName that user inputted is incorrect the following screen will show and ask the user to input again.



```
changeDir nothisdirectory
Invalid input - The inputted directory not found!
```

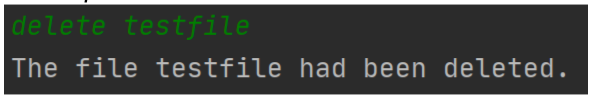
4. **newDoc** *docName docType docContent*



```
newDoc testfile java "This is the test content for testfile"
The new document testfile had been created.
```

This function will create a file according to the user input, this can create 4 type of files, such as “java”, “css”, “txt”, “html”. For the content of the file user can use “” to embrace the content needed, which allow user to have space in the content. If any of the above attribute had been inputted incorrectly error will be shown to notice the user.

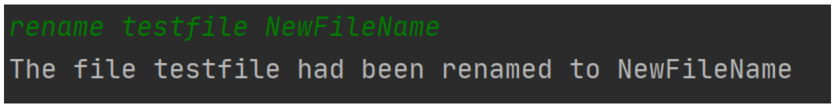
5. **delete** *fileName*



```
delete testfile
The file testfile had been deleted.
```

This function finds the file that the user inputted in the working directory, if the file is found, the system will delete. Otherwise, the system will print an error message to notice the user.

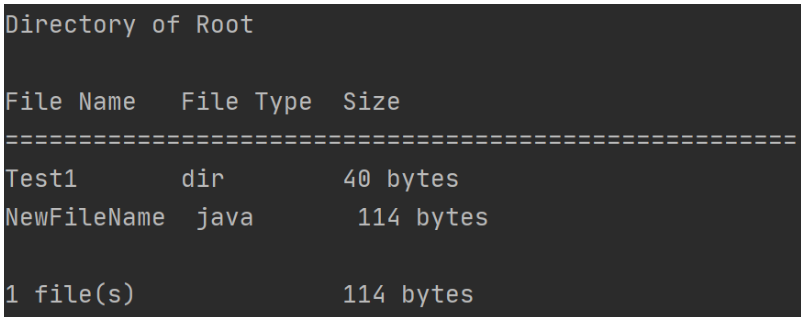
rename *oldFileName newFileName*



```
rename testfile NewFileName
The file testfile had been renamed to NewFileName
```

This function allows the user to change the file name of a specific file, simply by running the method with the two parameters of the old file name and the new file name. It will only find the file in the current directory, otherwise error message will return, also if the new file name entered already exists in the same directory error will also be shown.

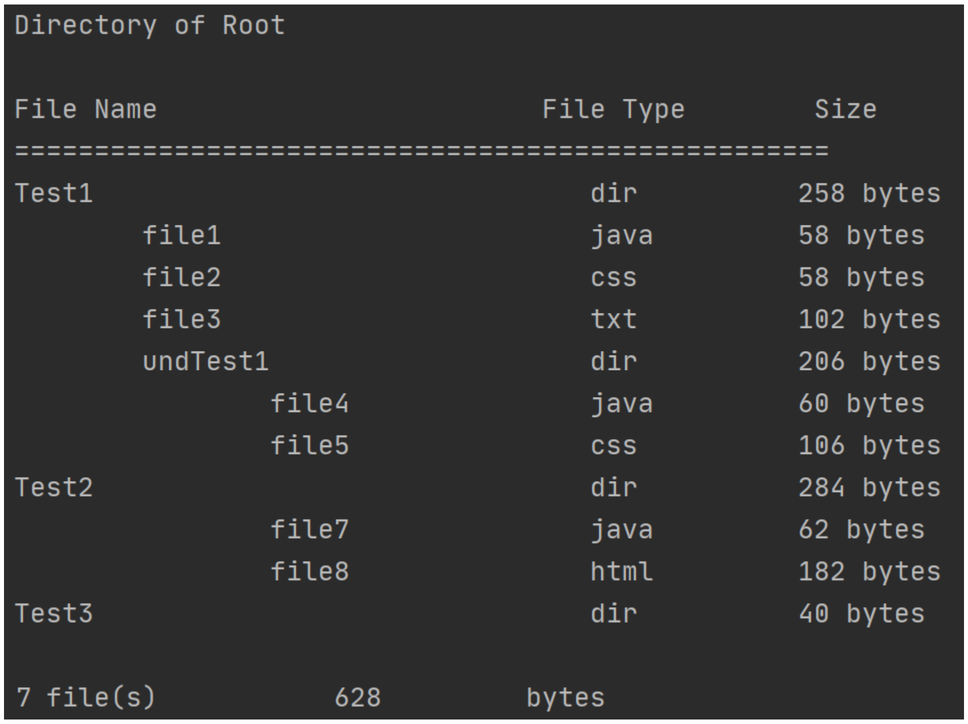
6. **list**



```
Directory of Root

File Name    File Type  Size
=======================================================
Test1        dir        40 bytes
NewFileName   java       114 bytes

1 file(s)               114 bytes
```

This function will list all the file under the same directory also with the file type and the file size. This function will only display all the items under the same directory, but it will not go in the sub-folder to search for the items in the others directories.

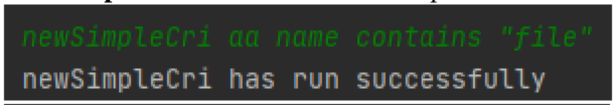
7. **rList**



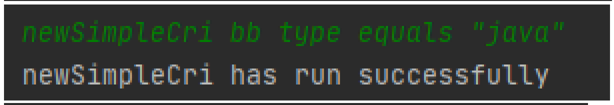
```
Directory of Root

File Name                         File Type        Size
====================================================
Test1                                dir          258 bytes
        file1                        java         58 bytes
        file2                        css          58 bytes
        file3                        txt          102 bytes
        undTest1                     dir          206 bytes
                file4                java         60 bytes
                file5                css          106 bytes
Test2                                dir          284 bytes
                file7                java         62 bytes
                file8                html         182 bytes
Test3                                dir          40 bytes

7 file(s)           628          bytes
```

The rList function will print out all the file in the directory including the sub-folders and the file under those directories.

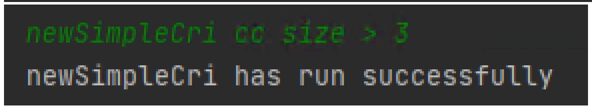
8. **newSimpleCri** *criName attrName op val*



```
newSimpleCri aa name contains "file"
newSimpleCri has run successfully
```



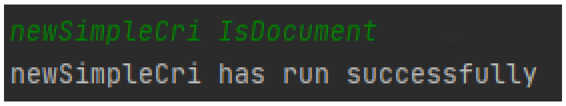
```
newSimpleCri bb type equals "java"
newSimpleCri has run successfully
```



```
newSimpleCri cc size > 3
newSimpleCri has run successfully
```

The newSimpleCri function will allow user to create the criteria by inputting the criteria name, type, operator and the value they want to use. If they want to create a criterion by using the type "name" or "type", they must double quote the value. The system will print the error message if the user made any error.

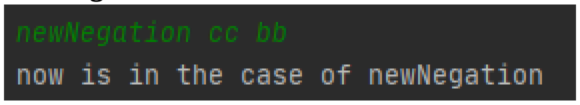
9. **IsDocument**



```
newSimpleCri IsDocument
newSimpleCri has run successfully
```

The IsDocument function requires the user to use it inside the newSimpleCri function. The system will print the error message if the user made any error.

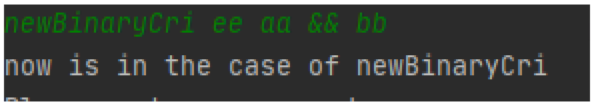
10. **newNegation** *criName1 criName2*



```
newNegation cc bb
now is in the case of newNegation
```

This method will assign (NOT criName2) into cirName1.

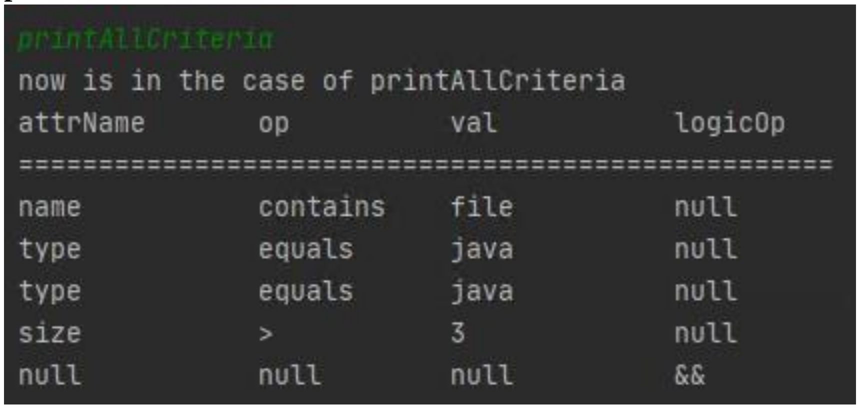
11. **newBinaryCri** *criName1 criName3 logicOp criName4*



```
newBinaryCri ee aa && bb
now is in the case of newBinaryCri
```

This method will combine the criName3 and criName4 according to the logicOp and store into criName1.

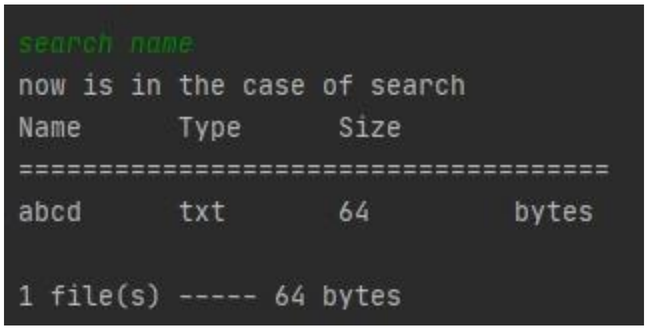
12. **printAllCriteria**



```
printAllCriteria
now is in the case of printAllCriteria
attrName      op          val         logicOp
===================================================
name          contains    file        null
type          equals      java        null
type          equals      java        null
size          >           3           null
null          null        null        &&
```

The printAllCriteria function will print all the criteria set by the user. If the criteria is composite criteria, the logicOp will be printed out.

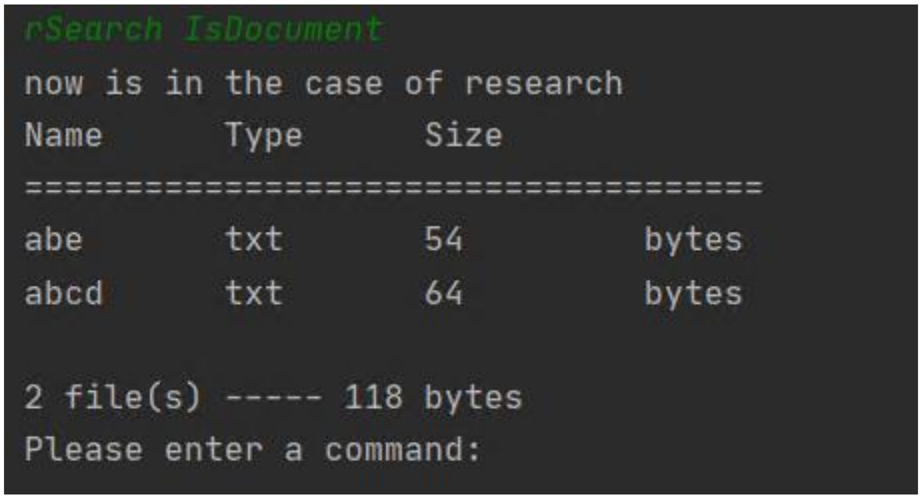
13. **search** *criName*



```
search name
now is in the case of search
Name      Type      Size
=====================================
abcd      txt       64         bytes

1 file(s) ----- 64 bytes
```

This method will search the current working directory according to the criName entered, and this method would only search the file in the current directory but not in the sub-folders.

14. rSearch *criName*



```
rSearch IsDocument
now is in the case of research
Name      Type      Size
=====================================
abe       txt       54         bytes
abcd      txt       64         bytes

2 file(s) ----- 118 bytes
Please enter a command:
```

This method is similar to “search”, except it shows all search results **recursively**, which the contents in the child directory is also shown.