Project Report

Group 36
COMP2021 Object-Oriented Programming (Fall 2020)
WANG Xiangzhi (19082878D)
WANG Zhe (19080037D)
YANG Junpeng (19107412D)
ZHANG Xianyi (19078513D)

1 Introduction

This document describes the design and implementation of the Comp Virtual File System (CVFS) by group 36. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

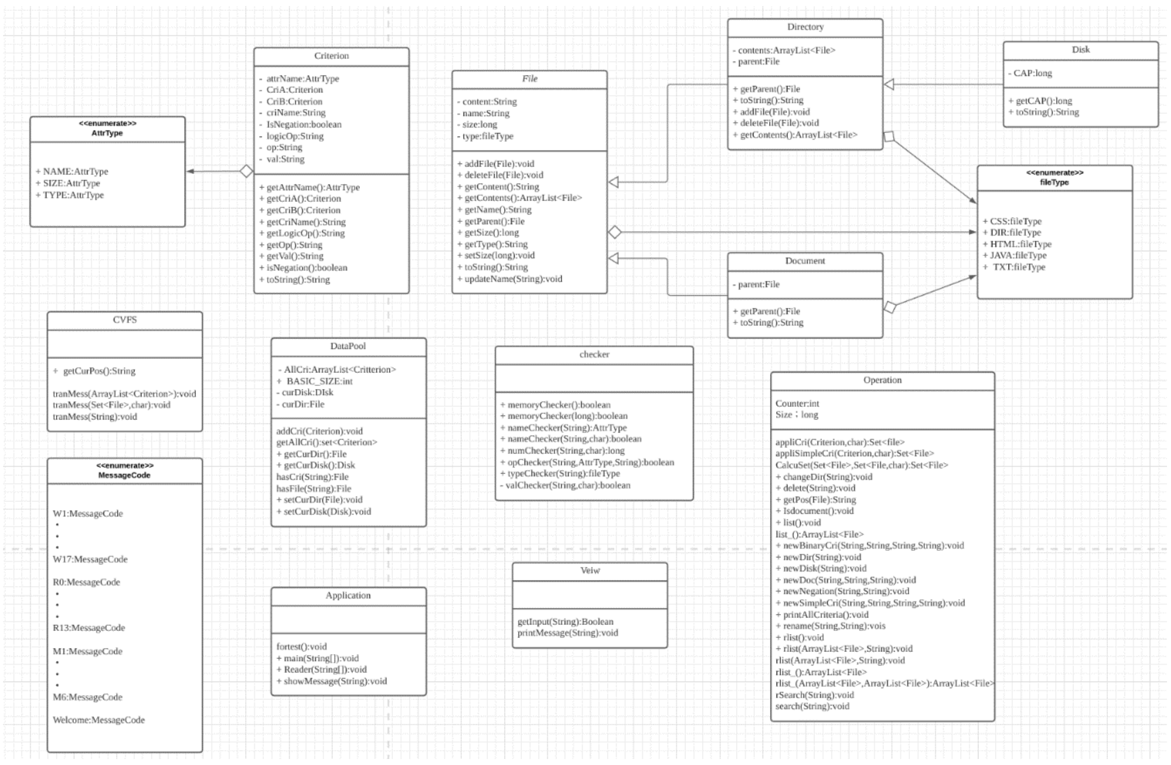
2 The Comp Virtual File System (CVFS)

In this section, we describe first the overall design of the CVFS and then the implementation details of the requirements.

2.1 Design

In this part, we give an overview of the system design and describe in general terms how different components fit together.

UML Class Diagram



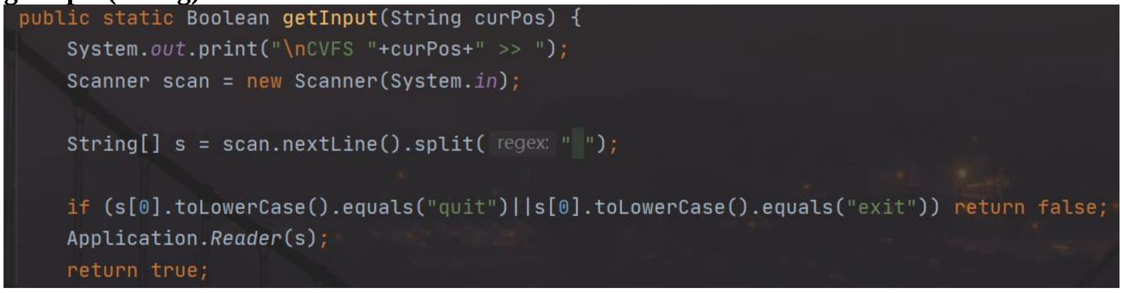
We used MVC Pattern as requested. Therefore, there are 3 main class used, which are:

2.1.1 View Part: Class View

View is a major part of the end of the CLI. It gets commands from the user and report issues to the user.

View contains only 2 methods:

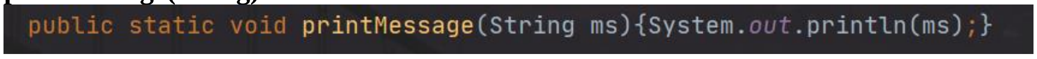
getInput(String): Boolean



```java
public static Boolean getInput(String curPos) {
    System.out.print("\nCVFS "+curPos+" >> ");
    Scanner scan = new Scanner(System.in);

    String[] s = scan.nextLine().split( regex: " ");

    if (s[0].toLowerCase().equals("quit")||s[0].toLowerCase().equals("exit")) return false;
    Application.Reader(s);
    return true;
```

The parameter is used to tell the user the current position (current working directory) of CVFS.
Command will be split base on blanks to achieve the purpose of word segmentation.
If the first word is “quit” or “exit” (ignore letter cases), false will be returned directly. Then the system will be shut down.
Else the split Command will be stored in an array and delivered to **Application.reader(String[]):void** to do further operation.

printMessage(String):void



```java
public static void printMessage(String ms){System.out.println(ms);}
```

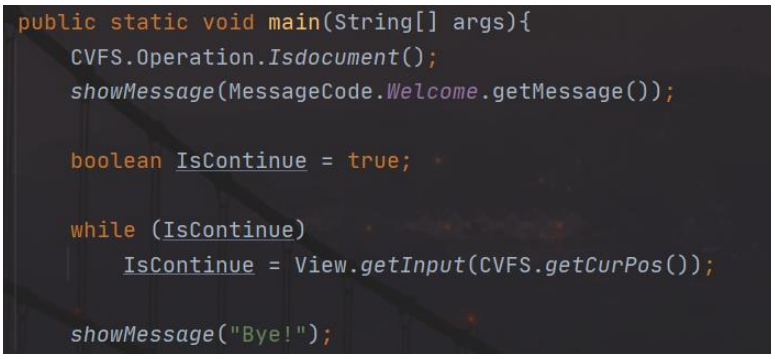
Model and Controller are using this method to tell issues to the user whenever needs. Most of the cases, users get to known whether a wrong command is entered or whether a command is accessed successfully.

2.1.2 Controller Part: public class Application

Application is the **Controller** Part of the MVC Pattern which is a bridge between **View** and **Model**. In addition, controlling the rhythm of the whole system is also the function of the Controller, that is why **main(String[]):void** is put in this part.

Here are the methods:

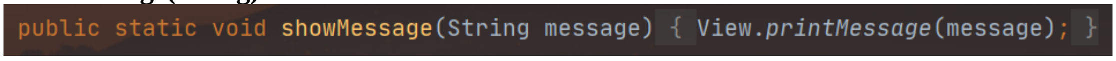
main(String[]):void



```java
public static void main(String[] args){
    CVFS.Operation.Isdocument();
    showMessage(MessageCode.Welcome.getMessage());

    boolean IsContinue = true;

    while (IsContinue)
        IsContinue = View.getInput(CVFS.getCurPos());

    showMessage("Bye!");
```

First, main creates the default criterion **Isdocument** and stores it.
Then, show the welcome messages, and get ready to scanner the user input.
Unless the user enters “**exit**” or “**quit**” to let **getInput** return false, the system never stops.

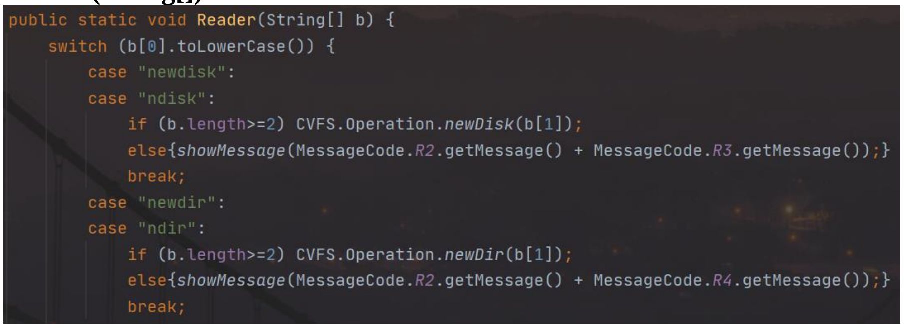
showMessage(String):void



```java
public static void showMessage(String message) { View.printMessage(message); }
```

Because Model cannot call the methods in View directly, showMessage() is used to help Model and Application deliver message to the CLI.

Reader(String[]):void



```java
public static void Reader(String[] b) {
    switch (b[0].toLowerCase()) {
        case "newdisk":
        case "ndisk":
            if (b.length>=2) CVFS.Operation.newDisk(b[1]);
            else{showMessage(MessageCode.R2.getMessage() + MessageCode.R3.getMessage());}
            break;
        case "newdir":
        case "ndir":
            if (b.length>=2) CVFS.Operation.newDir(b[1]);
            else{showMessage(MessageCode.R2.getMessage() + MessageCode.R4.getMessage());}
            break;
```

Reader is a long method and this pic is just part of it.
The key of Reader is to recognize the first word in the parameter array(ignore letter cases), if the 1st word is a keyword contained in the switch cases, the length of the array will be checked. The command is only valid when the content of the array is plenary, else a Warning Message will show and CLI will wait for next input.
If the first word not in the switch cases, “cannot recognized” issue will be shown to the user.
In addition, in the convenience of entering, abbreviations can also be recognized. For example, “printAllCriterion” can be written as “pac”,“newDoc” can be written as “ndoc”, etc. Details are in the User Manual.

2.1.3 Model part: Model subpackage

In the Model subpackage, we have:

dynamic class:

abstract class File();
class Document extends File();
class Directory extends File();

class Disk extends Document();

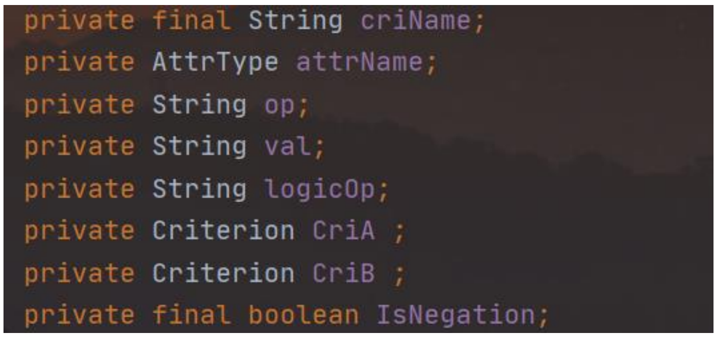
File is built to implement polymorphism, in the convenience of storing all **Directories** and **Documents** in ArrayList(Per directory has one instead of two). So all files are File type but are instantiated by Directory or Document. That is why there are some empty methods in the class File. They are overrides in Document and Directory.

For **Disk**, it must build a field to store the **Capacity**, and it still need the field contents and corresponding methods of **Directory**. There is no need to add any field in File for Disk or rewrite some method again at **Disk**. Thus, we just let it extend **Directory** and instantiate itself. After checking and adjustment, they can be friendly used.

class Criterion();

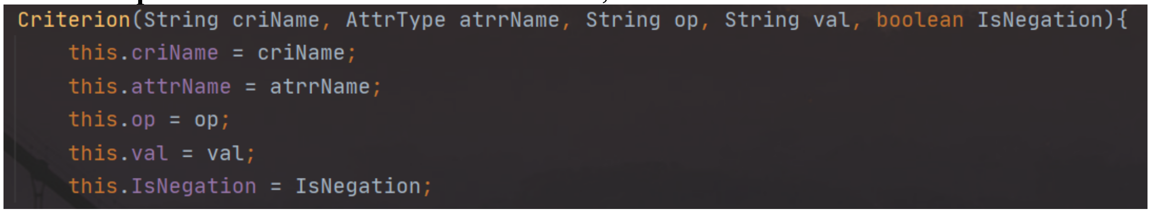
The most interesting one must be the class **Criterion**. Again, in the convenience of storing all Criterions in an arrayList. We put **Simple Criterion**, **Negation Criterion** and **Binary Criterion** into one single class.

Here are our operations:
Double Constructor, One field.
Actually all variables for both Simple Criterion and Negation Criterion are contained in the fields of class **Criterion**.



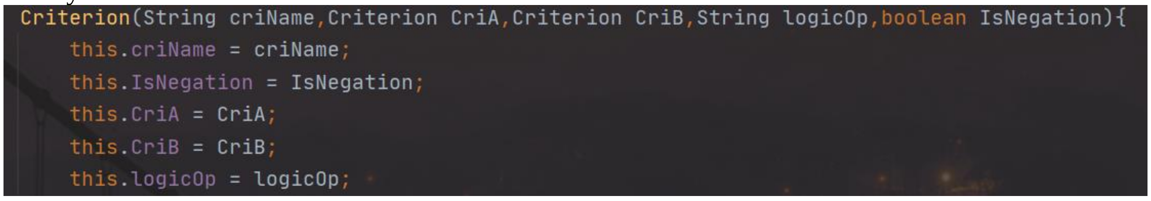
```
private final String criName;
private AttrType attrName;
private String op;
private String val;
private String logicOp;
private Criterion CriA ;
private Criterion CriB ;
private final boolean IsNegation;
```

Once a **Simple Criterion** is needed to be instantiated, the first Constructor will be used.



```
Criterion(String criName, AttrType atrrName, String op, String val, boolean IsNegation){
    this.criName = criName;
    this.attrName = atrrName;
    this.op = op;
    this.val = val;
    this.IsNegation = IsNegation;
```

Binary will use the second one:



```
Criterion(String criName,Criterion CriA,Criterion CriB,String logicOp,boolean IsNegation){
    this.criName = criName;
    this.IsNegation = IsNegation;
    this.CriA = CriA;
    this.CriB = CriB;
    this.logicOp = logicOp;
```

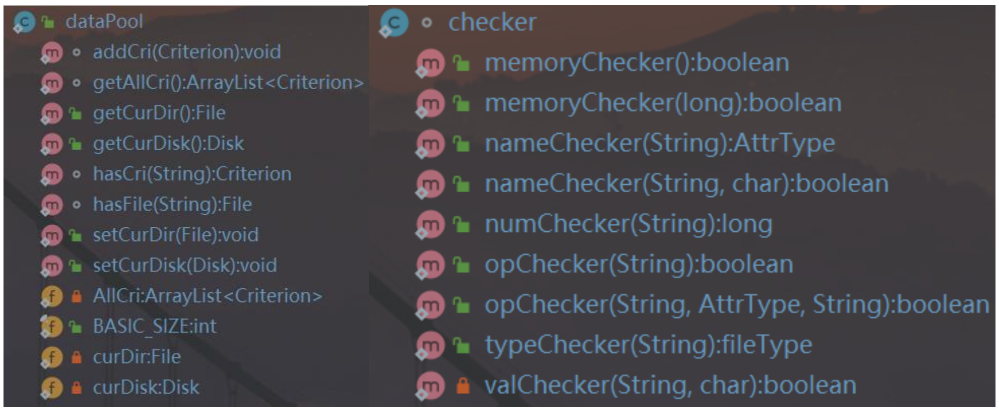
So that unused field will not occupy any storage. In addition, the identifier for **Simple Criterion** and **Binary Criterion** is the filed **logicOp** (not null for Binary ones and null for Simple ones).
As shown above, the field **IsNegation** of **Negation Criterion** will be true, else false.

Static classes:

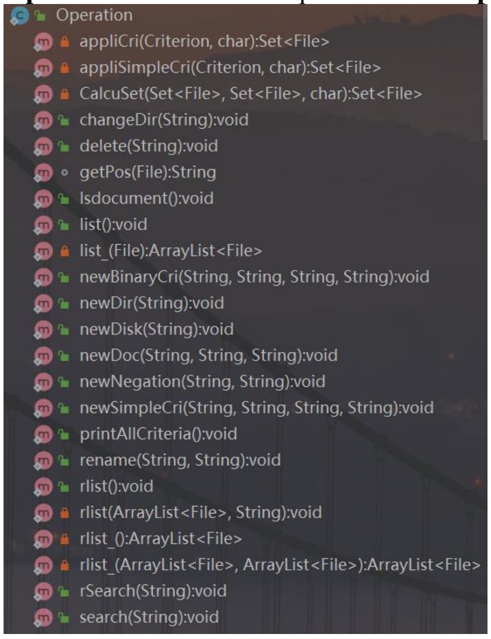
Public static class dataPool(); in public class cvfs();

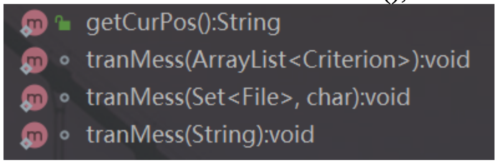
Public static class checker(); in public class cvfs();
Public static class Operation(); in public class cvfs();

DataPool is used for storing real-time data. It has fields **curDisk** and **curDir** to store current disk and directory, **allCri** to store all **Criterion** created at this time. Method **hasFile(String):File** is to check whether a file is stored in the working directory (if the working directory is the Disk, **curDir** = null and other methods can judge it), **hasCri(String):Criterion** is to check whether a Criterion exists in system at this time.


```
dataPool
  addCri(Criterion):void
  getAllCri():ArrayList<Criterion>
  getCurDir():File
  getCurDisk():Disk
  hasCri(String):Criterion
  hasFile(String):File
  setCurDir(File):void
  setCurDisk(Disk):void
  AllCri:ArrayList<Criterion>
  BASIC_SIZE:int
  curDir:File
  curDisk:Disk
```
```
checker
  memoryChecker():boolean
  memoryChecker(long):boolean
  nameChecker(String):AttrType
  nameChecker(String, char):boolean
  numChecker(String):long
  opChecker(String):boolean
  opChecker(String, AttrType, String):boolean
  typeChecker(String):fileType
  valChecker(String, char):boolean
```

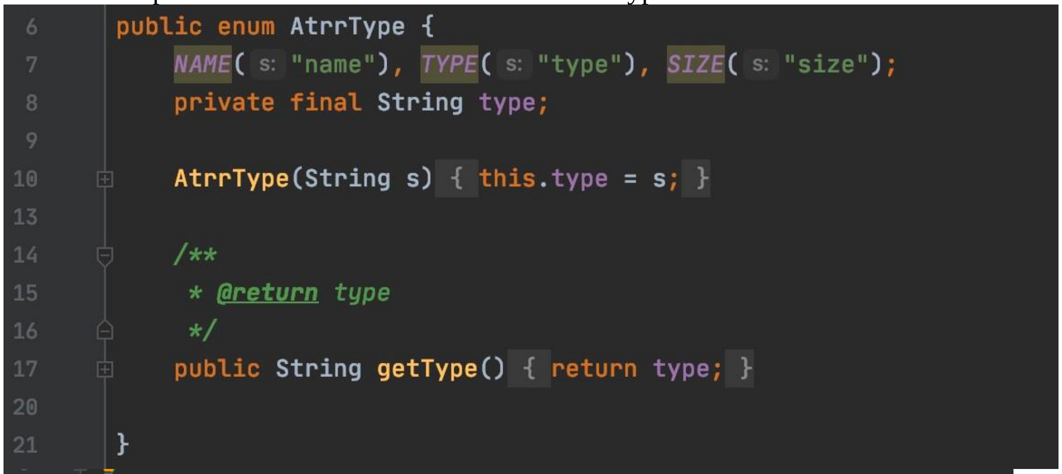
checker contains all methods of checking the legality of a user-input element except keywords (already checked in Reader in Application of Controller part) .

Operation contains all requirements in **report_description.pdf**

```
Operation
  appliCri(Criterion, char):Set<File>
  appliSimpleCri(Criterion, char):Set<File>
  CalcuSet(Set<File>, Set<File>, char):Set<File>
  changeDir(String):void
  delete(String):void
  getPos(File):String
  lsdocument():void
  list():void
  list_(File):ArrayList<File>
  newBinaryCri(String, String, String, String):void
  newDir(String):void
  newDisk(String):void
  newDoc(String, String, String):void
  newNegation(String, String):void
  newSimpleCri(String, String, String, String):void
  printAllCriteria():void
  rename(String, String):void
  rlist():void
  rlist(ArrayList<File>, String):void
  rlist_():ArrayList<File>
  rlist_(ArrayList<File>, ArrayList<File>):ArrayList<File>
  rSearch(String):void
  search(String):void
```
Details will be explained in Requirements below.

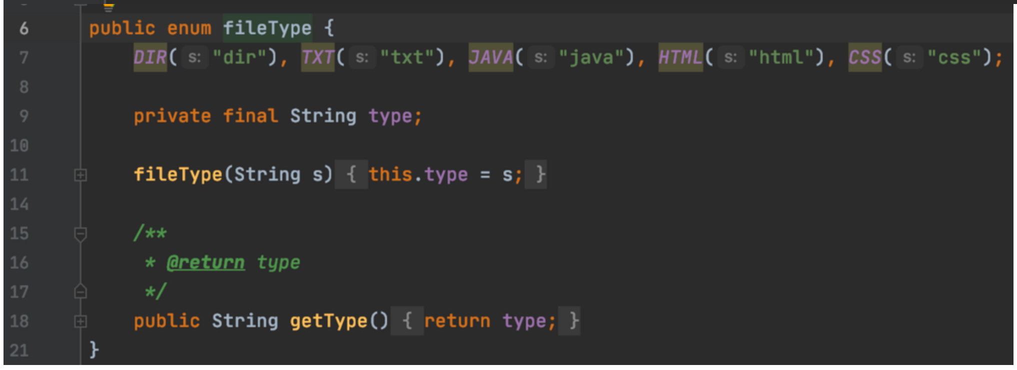
Static method in Public class cvfs();

```
getCurPos():String
tranMess(ArrayList<Criterion>):void
tranMess(Set<File>, char):void
tranMess(String):void
```
getCurPos can get current working path and send it in string format to **Application.**
tranMess() is used for sending message to **Application** to tell the User

2.1.4 Enums In CFVS

We created two java class files named: **AtrrType.java**, **fileType.java to** store the types of different Simple Criterions attributes and different file types.

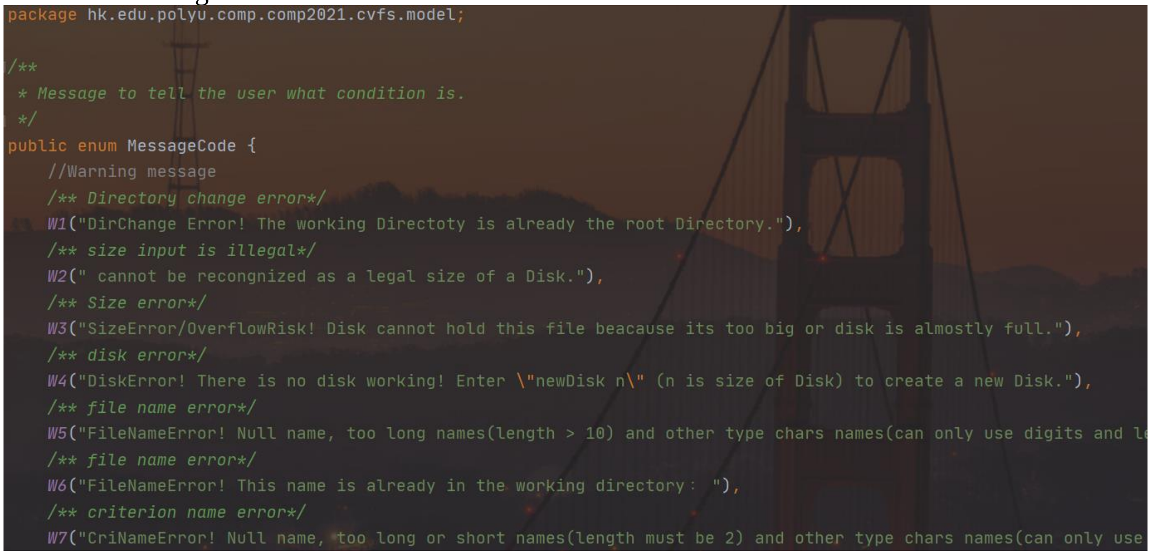


```
public enum AtrrType {
    NAME( s: "name"), TYPE( s: "type"), SIZE( s: "size");
    private final String type;

    AtrrType(String s) { this.type = s; }

    /**
     * @return type
     */
    public String getType() { return type; }

}
```



```
public enum fileType {
    DIR( s: "dir"), TXT( s: "txt"), JAVA( s: "java"), HTML( s: "html"), CSS( s: "css");

    private final String type;

    fileType(String s) { this.type = s; }

    /**
     * @return type
     */
    public String getType() { return type; }
}
```

MessageCode.java is built for enumerating all possible Warnings, Reminders and Condition Messages.



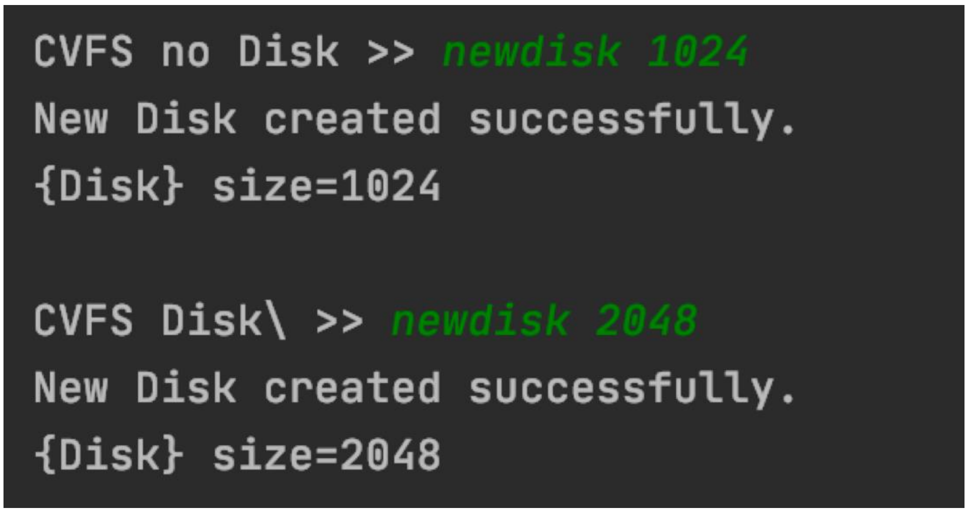
```
package hk.edu.polyu.comp.comp2021.cvfs.model;

/**
 * Message to tell the user what condition is.
 */
public enum MessageCode {
    //Warning message
    /** Directory change error*/
    W1("DirChange Error! The working Directoty is already the root Directory."),
    /** size input is illegal*/
    W2(" cannot be recongnized as a legal size of a Disk."),
    /** Size error*/
    W3("SizeError/OverflowRisk! Disk cannot hold this file beacause its too big or disk is almostly full."),
    /** disk error*/
    W4("DiskError! There is no disk working! Enter \"newDisk n\" (n is size of Disk) to create a new Disk."),
    /** file name error*/
    W5("FileNameError! Null name, too long names(length > 10) and other type chars names(can only use digits and l
    /** file name error*/
    W6("FileNameError! This name is already in the working directory： "),
    /** criterion name error*/
    W7("CriNameError! Null name, too long or short names(length must be 2) and other type chars names(can only use
```

2.2 Requirements

In this part, we describe the implementation details of the requirements including
1) whether it is implemented
2) how we implemented the requirement
3) how we handled various error conditions.

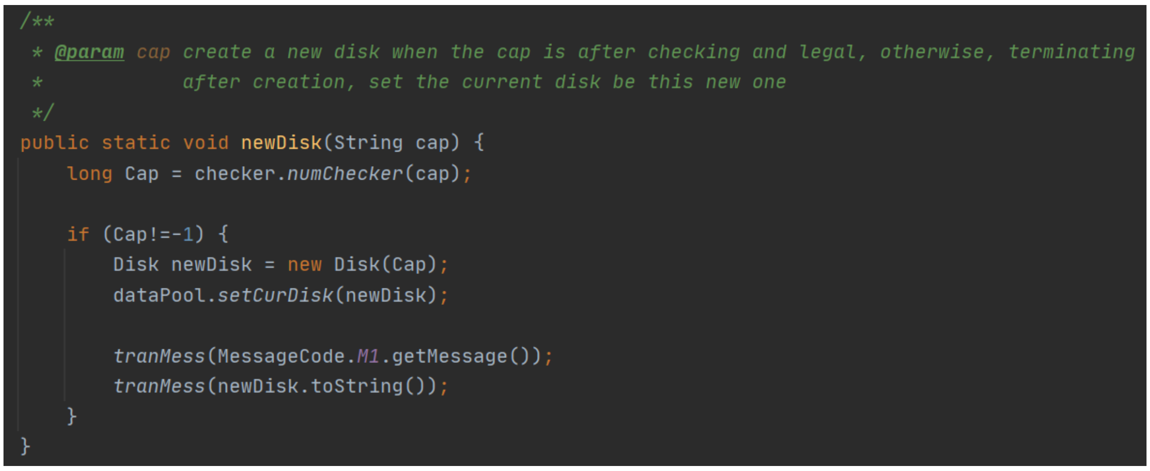
2.2.1[REQ1]

1) The requirement is implemented. The sample input and output are shown below:



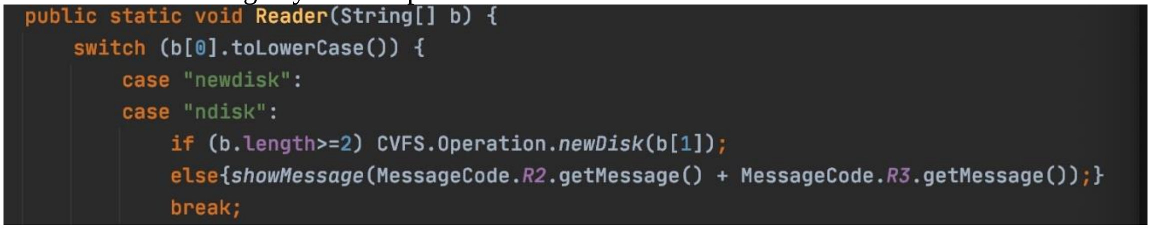
```
CVFS no Disk >> newdisk 1024
New Disk created successfully.
{Disk} size=1024

CVFS Disk\ >> newdisk 2048
New Disk created successfully.
{Disk} size=2048
```

2) Implementation details.
Command “**newDisk** diskSize” is implemented as below.



```
/**
 * @param cap create a new disk when the cap is after checking and legal, otherwise, terminating
 *            after creation, set the current disk be this new one
 */
public static void newDisk(String cap) {
    long Cap = checker.numChecker(cap);

    if (Cap!=-1) {
        Disk newDisk = new Disk(Cap);
        dataPool.setCurDisk(newDisk);

        tranMess(MessageCode.M1.getMessage());
        tranMess(newDisk.toString());
    }
}
```

Reader checks the legality of the input.



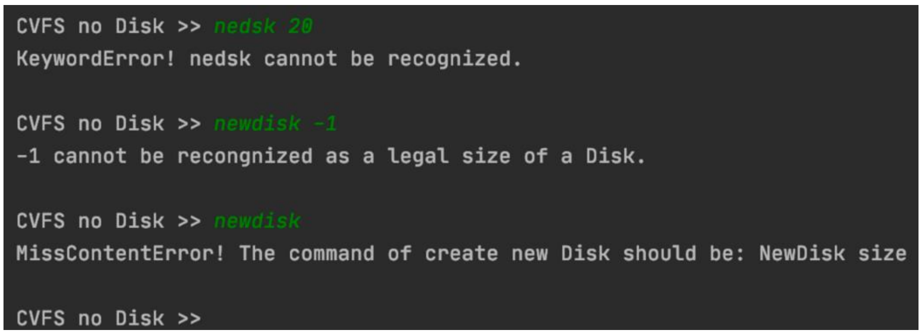
```
public static void Reader(String[] b) {
    switch (b[0].toLowerCase()) {
        case "newdisk":
        case "ndisk":
            if (b.length>=2) CVFS.Operation.newDisk(b[1]);
            else{showMessage(MessageCode.R2.getMessage() + MessageCode.R3.getMessage());}
            break;
```

Inside the "**newDisk** diskSize" operation, we check the legality of the max size (size must be greater than 0). If legal, it will create a new disk in maximize size (which means the previous one is closed) and print the message "New Disk created successfully.".

3) Error conditions and how they are handled.

There are 3 kinds of errors:

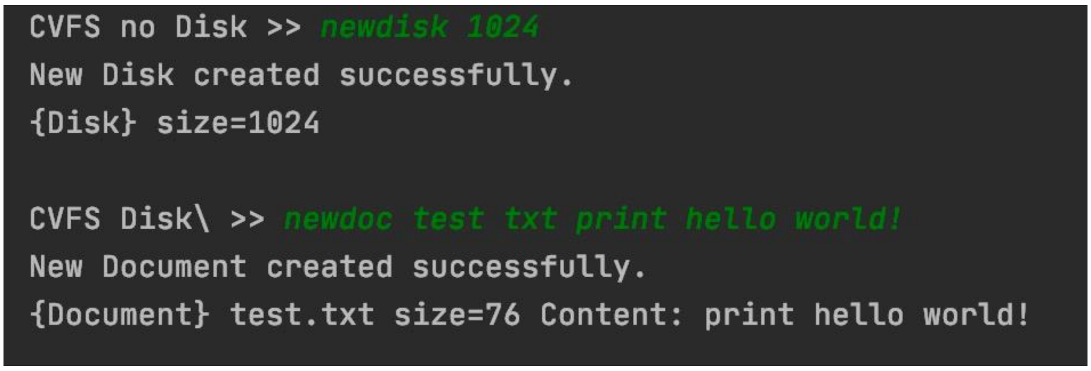
(1) **KeywordError**. If the keyword is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.
(2) **InputError**. If the input size is illegal (input size <= 0), the CVFS system will report: xxx cannot be recognized as a legal size of a Disk.
(3) **MissContentError**. If user forgets to input size, the system will report: MissContentError! The command of create new Disk should be: NewDisk size.



```
CVFS no Disk >> nedsk 20
KeywordError! nedsk cannot be recognized.

CVFS no Disk >> newdisk -1
-1 cannot be recongnized as a legal size of a Disk.

CVFS no Disk >> newdisk
MissContentError! The command of create new Disk should be: NewDisk size

CVFS no Disk >>
```

2.2.2[REQ2]

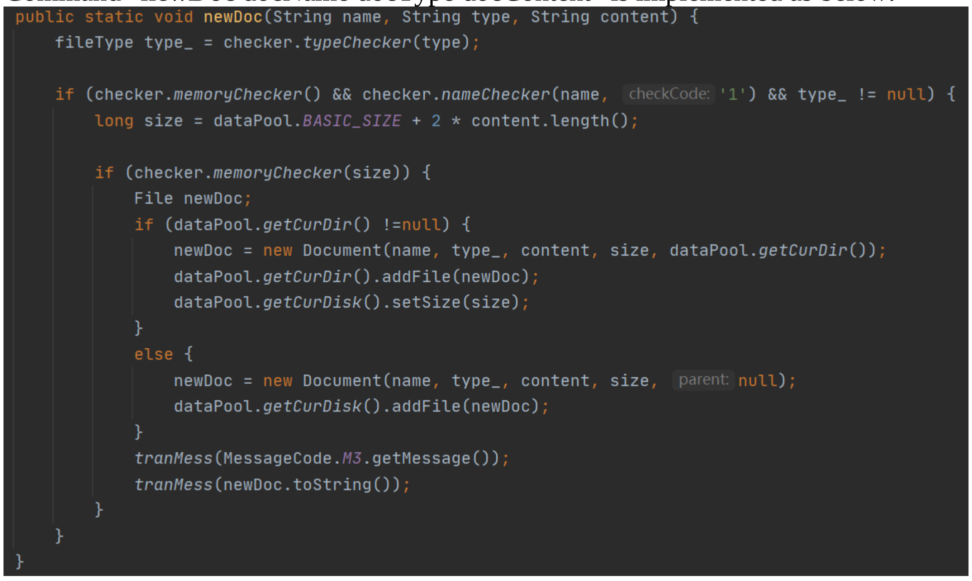
1) **The requirement is implemented. The sample input and output are shown below:**



```
CVFS no Disk >> newdisk 1024
New Disk created successfully.
{Disk} size=1024

CVFS Disk\ >> newdoc test txt print hello world!
New Document created successfully.
{Document} test.txt size=76 Content: print hello world!
```

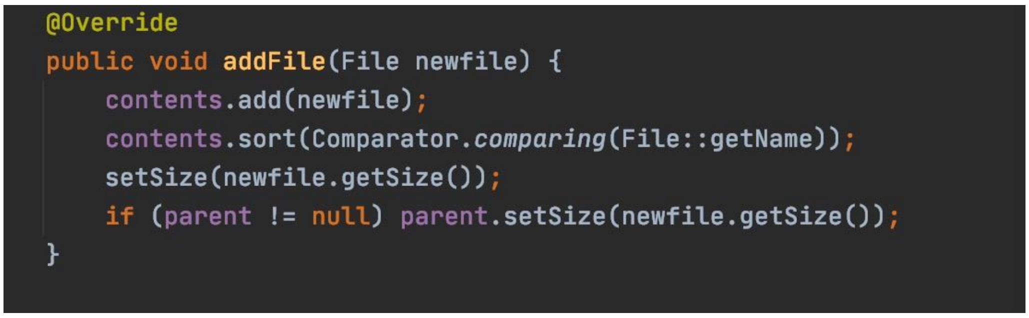
2) Implementation details.

Command “**newDoc** docName docType docContent” is implemented as below.



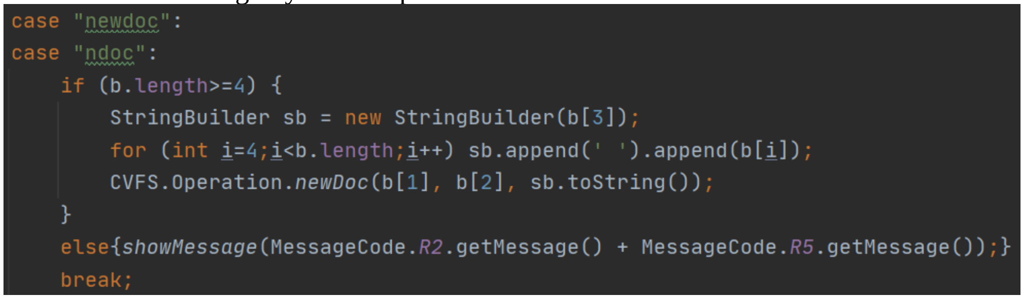
```java
public static void newDoc(String name, String type, String content) {
    fileType type_ = checker.typeChecker(type);

    if (checker.memoryChecker() && checker.nameChecker(name,  checkCode: '1') && type_ != null) {
        long size = dataPool.BASIC_SIZE + 2 * content.length();

        if (checker.memoryChecker(size)) {
            File newDoc;
            if (dataPool.getCurDir() !=null) {
                newDoc = new Document(name, type_, content, size, dataPool.getCurDir());
                dataPool.getCurDir().addFile(newDoc);
                dataPool.getCurDisk().setSize(size);
            }
            else {
                newDoc = new Document(name, type_, content, size,  parent: null);
                dataPool.getCurDisk().addFile(newDoc);
            }
            tranMess(MessageCode.M3.getMessage());
            tranMess(newDoc.toString());
        }
    }
}
```

It calls the function “**addFile** File newfile”.



```java
@Override
public void addFile(File newfile) {
    contents.add(newfile);
    contents.sort(Comparator.comparing(File::getName));
    setSize(newfile.getSize());
    if (parent != null) parent.setSize(newfile.getSize());
}
```

Reader checks the legality of the input.



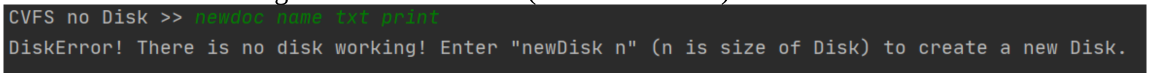
```java
case "newdoc":
case "ndoc":
    if (b.length>=4) {
        StringBuilder sb = new StringBuilder(b[3]);
        for (int i=4;i<b.length;i++) sb.append(' ').append(b[i]);
        CVFS.Operation.newDoc(b[1], b[2], sb.toString());
    }
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R5.getMessage());}
    break;
```

Inside the “**newDoc** docName docType docContent” command, we first check the legality of the type, the legality of the name and whether there is a disk working, then calculate the size using the given formula. After checking the remaining memory size, it creates a new File (as a document). Then judge whether the current directory is empty, if so, add the document to disk, else add the document to disk and set the remain size. At last, print the message “New Document created successfully.”.

3) Error conditions and how they are handled.

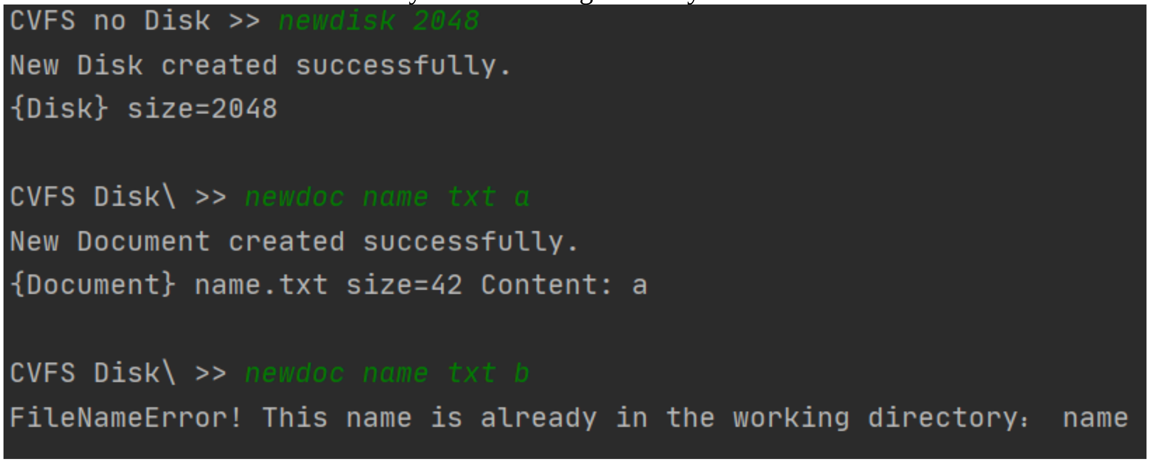
There are 6 kinds of errors:

(1) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



```
CVFS no Disk >> newdoc name txt print
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

(2) **FileNameError.** If the file name is repeated, the system will report: FileNameError! This name is already in the working directory: xxx



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdoc name txt a
New Document created successfully.
{Document} name.txt size=42 Content: a

CVFS Disk\ >> newdoc name txt b
FileNameError! This name is already in the working directory:  name
```

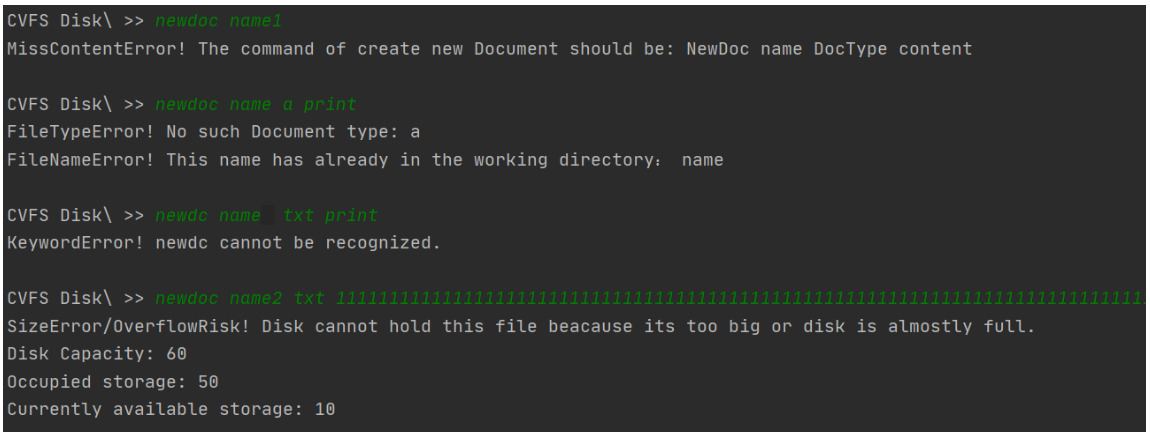
(3) **MissContentError**. If the input misses one of the components, the system will report: MissContentError! The command of create new Document should be: NewDoc name Doctype content.

(4) **FileTypeError.** If the input type is illegal, the CVFS system will report: FileTypeError! No such Document type: a

(5) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.

(6) **SizeError /OverflowRisk.** If the size of the new document is greater than the remain size of the disk, the system will report:

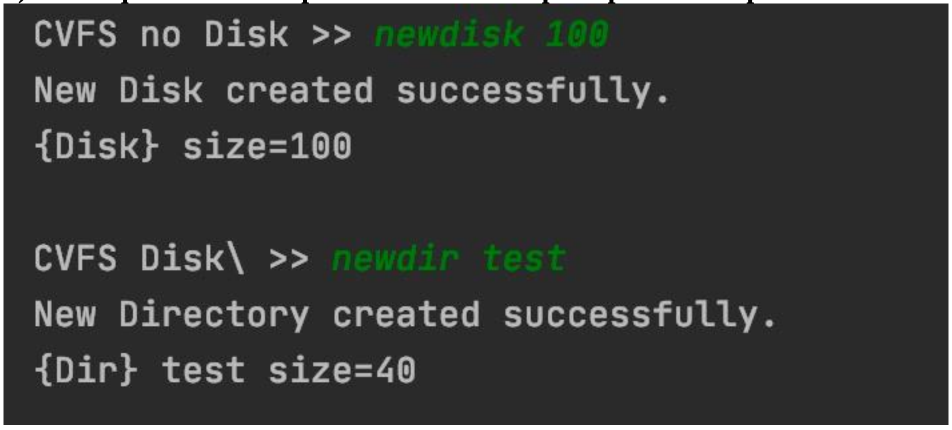
SizeError /OverflowRisk! Disk cannot hold this file beacause it’s too big or disk is almost full. Disk Capacity: xxx
Occupied storage: xxx
Currently available storage: xxx



```
CVFS Disk\ >> newdoc name1
MissContentError! The command of create new Document should be: NewDoc name DocType content

CVFS Disk\ >> newdoc name a print
FileTypeError! No such Document type: a
FileNameError! This name has already in the working directory:  name

CVFS Disk\ >> newdc name  txt print
KeywordError! newdc cannot be recognized.

CVFS Disk\ >> newdoc name2 txt 1111111111111111111111111111111111111111111111111111111111111111111111111111
SizeError/OverflowRisk! Disk cannot hold this file beacause its too big or disk is almostly full.
Disk Capacity: 60
Occupied storage: 50
Currently available storage: 10
```

2.2.3[REQ3]

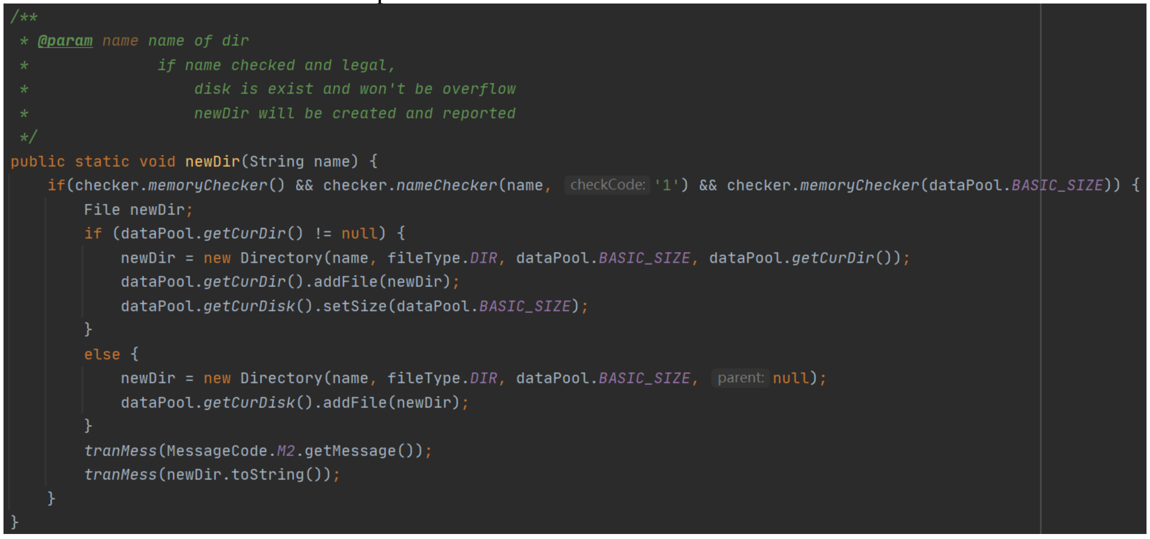
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> newdisk 100
New Disk created successfully.
{Disk} size=100

CVFS Disk\ >> newdir test
New Directory created successfully.
{Dir} test size=40
```

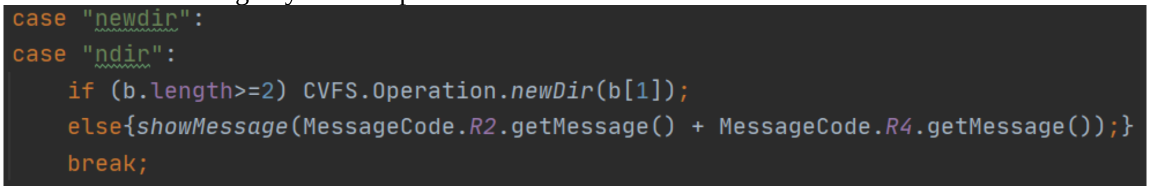
2) Implementation details.

Command “**newDir** dirName” is implemented as below.



```
/**
 * @param name name of dir
 *              if name checked and legal,
 *                  disk is exist and won't be overflow
 *                  newDir will be created and reported
 */
public static void newDir(String name) {
    if(checker.memoryChecker() && checker.nameChecker(name, checkCode: '1') && checker.memoryChecker(dataPool.BASIC_SIZE)) {
        File newDir;
        if (dataPool.getCurDir() != null) {
            newDir = new Directory(name, fileType.DIR, dataPool.BASIC_SIZE, dataPool.getCurDir());
            dataPool.getCurDir().addFile(newDir);
            dataPool.getCurDisk().setSize(dataPool.BASIC_SIZE);
        }
        else {
            newDir = new Directory(name, fileType.DIR, dataPool.BASIC_SIZE, parent: null);
            dataPool.getCurDisk().addFile(newDir);
        }
        tranMess(MessageCode.M2.getMessage());
        tranMess(newDir.toString());
    }
}
```

Reader checks the legality of the input.



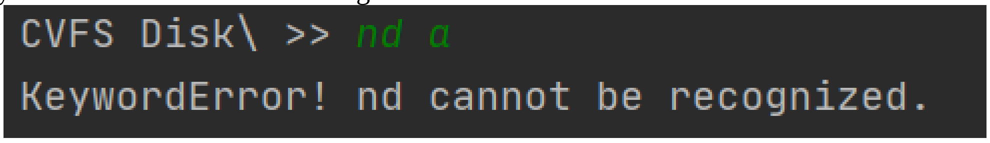
```
case "newdir":
case "ndir":
    if (b.length>=2) CVFS.Operation.newDir(b[1]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R4.getMessage());}
    break;
```

Inside the "**newDir** dirName" command, we check the legality of the name, whether there is a disk working and remaining memory size. Then create a new File (as a directory) and judge whether the current directory is empty, if so, add the new directory to disk, else let the new directory be in the working directory. At last, print the message "New Directory created successfully.".

3) Error conditions and how they are handled.

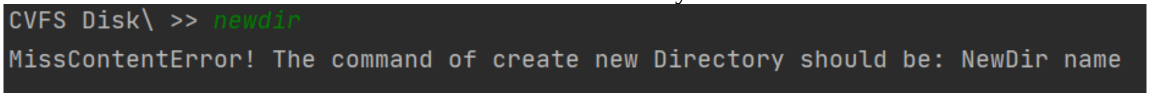
There are 5 kinds of errors:

(1) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



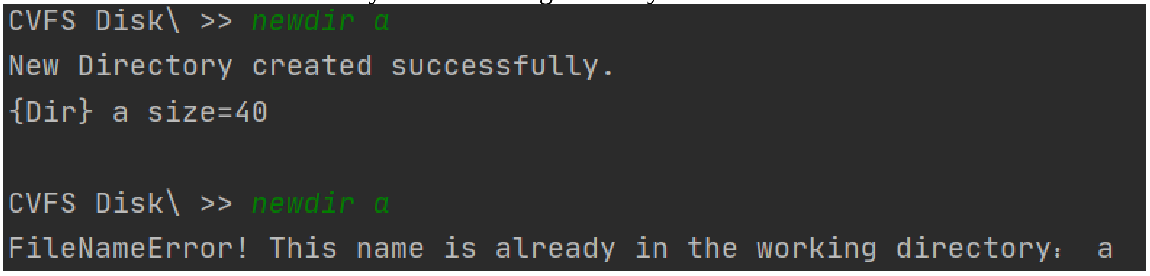
```
CVFS Disk\ >> nd a
KeywordError! nd cannot be recognized.
```

(2) **MissContentError**. If the input misses one of the components, the system will report: MissContentError! The command of create new Directory should be: NewDir name.



```
CVFS Disk\ >> newdir
MissContentError! The command of create new Directory should be: NewDir name
```

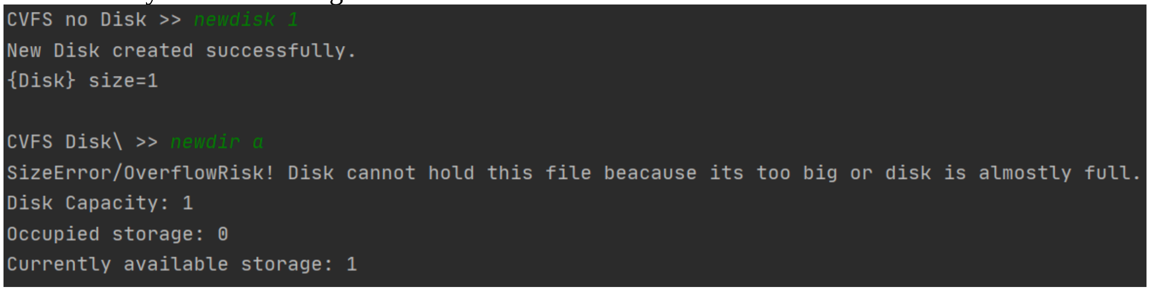
(3) **FileNameError.** If the input name is repeated, the system will report: FileNameError! This name is already in the working directory: xxx



```
CVFS Disk\ >> newdir a
New Directory created successfully.
{Dir} a size=40

CVFS Disk\ >> newdir a
FileNameError! This name is already in the working directory:  a
```

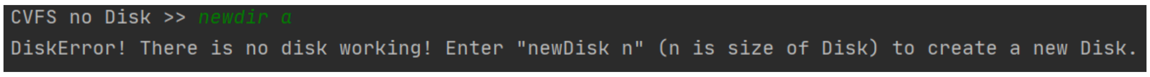
(4) **SizeError /OverflowRisk.** If the size of the new document is greater than the remain size of the disk, the system will report:

SizeError /OverflowRisk! Disk cannot hold this file beacause it's too big or disk is almost full. Disk Capacity: xxx
Occupied storage: xxx
Currently available storage: xxx



```
CVFS no Disk >> newdisk 1
New Disk created successfully.
{Disk} size=1

CVFS Disk\ >> newdir a
SizeError/OverflowRisk! Disk cannot hold this file beacause its too big or disk is almostly full.
Disk Capacity: 1
Occupied storage: 0
Currently available storage: 1
```

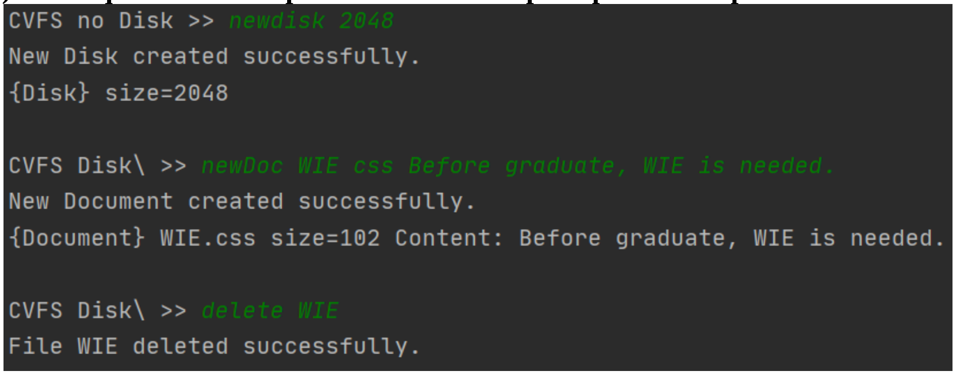
(5) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



```
CVFS no Disk >> newdir a
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

2.2.4[REQ4]

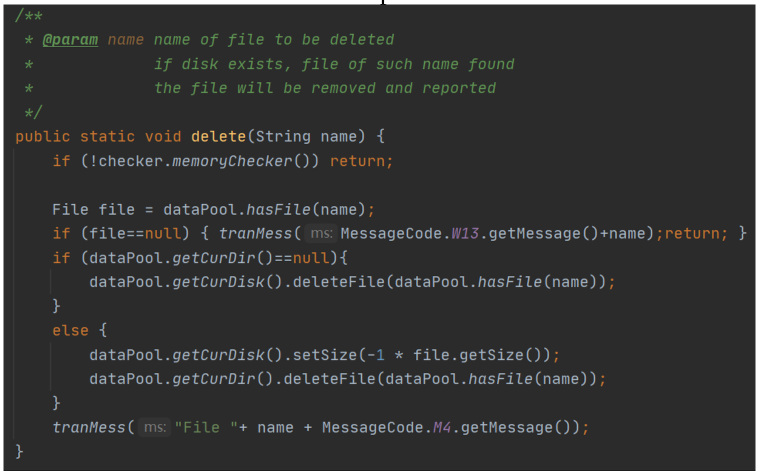
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newDoc WIE css Before graduate, WIE is needed.
New Document created successfully.
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.

CVFS Disk\ >> delete WIE
File WIE deleted successfully.
```

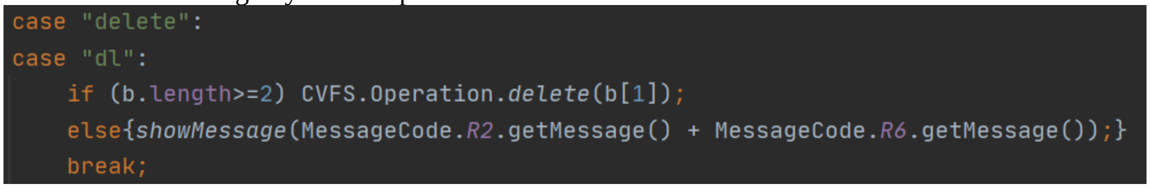
2) Implementation details.

Command “**delete** fileName” is implemented as below.



```
/**
 * @param name name of file to be deleted
 *             if disk exists, file of such name found
 *             the file will be removed and reported
 */
public static void delete(String name) {
    if (!checker.memoryChecker()) return;

    File file = dataPool.hasFile(name);
    if (file==null) { tranMess( ms: MessageCode.W13.getMessage()+name);return; }
    if (dataPool.getCurDir()==null){
        dataPool.getCurDisk().deleteFile(dataPool.hasFile(name));
    }
    else {
        dataPool.getCurDisk().setSize(-1 * file.getSize());
        dataPool.getCurDir().deleteFile(dataPool.hasFile(name));
    }
    tranMess( ms: "File "+ name + MessageCode.M4.getMessage());
}
```

Reader checks the legality of the input.



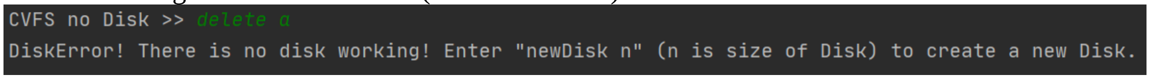
```
case "delete":
case "dl":
    if (b.length>=2) CVFS.Operation.delete(b[1]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R6.getMessage());}
    break;
```

Inside the “**delete** fileName” command, we check whether there is a disk working. Then find the file with the specific name and judge whether the current directory is empty, if so, delete the file directly from disk, else delete the file in directory and set the size of the disk minus 1. At last, print the message “File xxx deleted successfully.”.

3) Error conditions and how they are handled.

There are 4 kinds of errors:

(1) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.

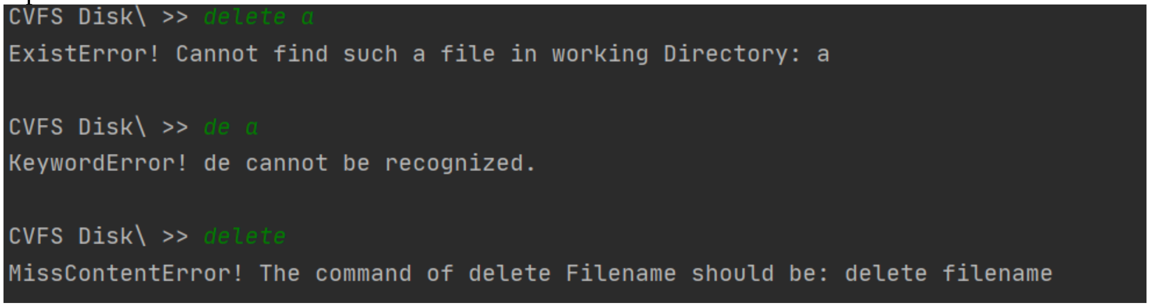


```
CVFS no Disk >> delete a
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

(2) **ExistError.** If there isn’t a file with the specific name, the CVFS system will report: ExistError! Cannot find such a file in working Directory: a.

(3) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.

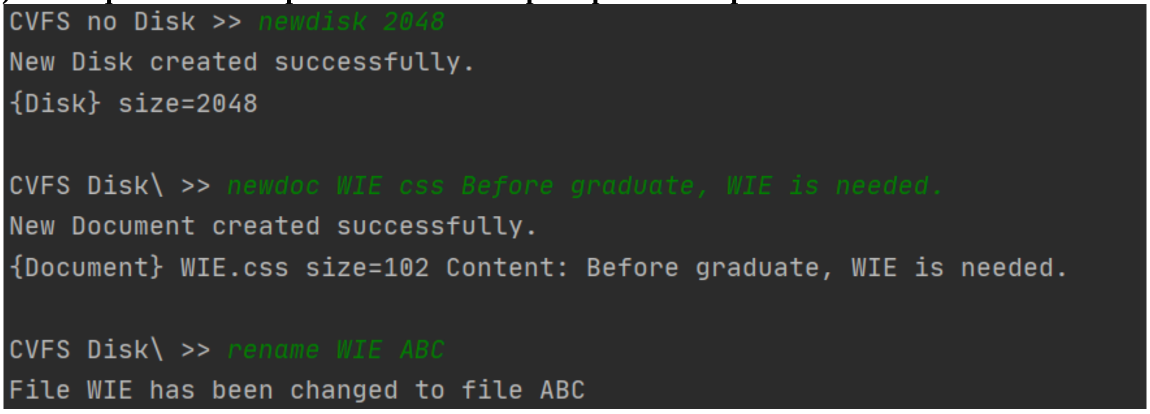
(4) **MissContentError.** If the input misses one of the components, the system will report: MissContentError! The command of delete Filename should be: delete filename.



```
CVFS Disk\ >> delete a
ExistError! Cannot find such a file in working Directory: a

CVFS Disk\ >> de a
KeywordError! de cannot be recognized.

CVFS Disk\ >> delete
MissContentError! The command of delete Filename should be: delete filename
```

2.2.5[REQ5]

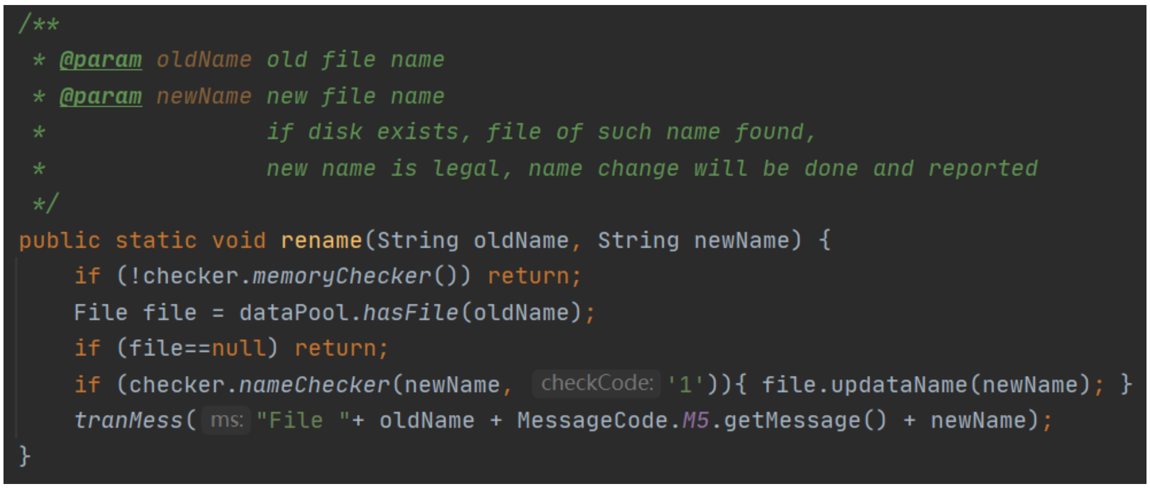
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdoc WIE css Before graduate, WIE is needed.
New Document created successfully.
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.

CVFS Disk\ >> rename WIE ABC
File WIE has been changed to file ABC
```

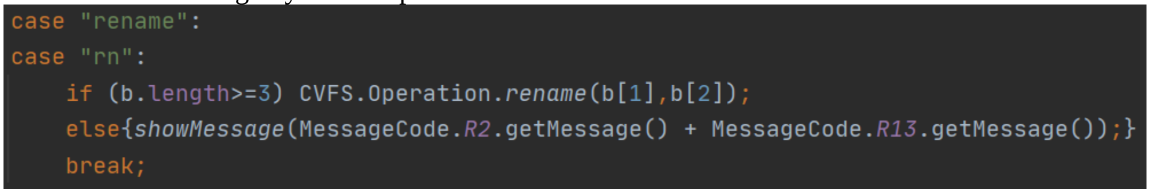
2) Implementation details.

Command “**rename** oldFileName newFileName” is implemented as below.



```
/**
 * @param oldName old file name
 * @param newName new file name
 *                if disk exists, file of such name found,
 *                new name is legal, name change will be done and reported
 */
public static void rename(String oldName, String newName) {
    if (!checker.memoryChecker()) return;
    File file = dataPool.hasFile(oldName);
    if (file==null) return;
    if (checker.nameChecker(newName,  checkCode: '1')){ file.updataName(newName); }
    tranMess( ms: "File "+ oldName + MessageCode.M5.getMessage() + newName);
}
```

Reader checks the legality of the input.



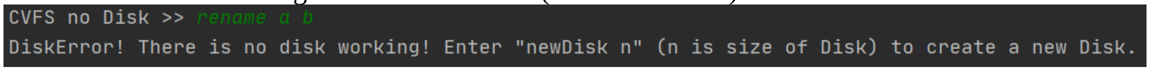
```
case "rename":
case "rn":
    if (b.length>=3) CVFS.Operation.rename(b[1],b[2]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R13.getMessage());}
    break;
```

Inside the "**rename** oldFileName newFileName" command, we check whether there is a disk working. Then find the file with the old name and judge whether there exists the file. If so, check the legality of the new name and rename the old one when legal. At last, print the message "File xxx has been changed to file xxxx.".

3) Error conditions and how they are handled.

There are 5 kinds of errors:

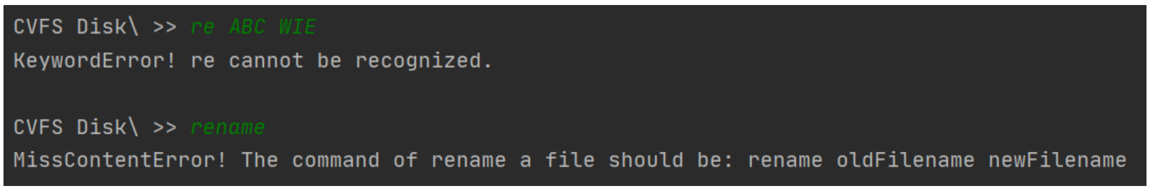
(1)**DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



```
CVFS no Disk >> rename a b
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

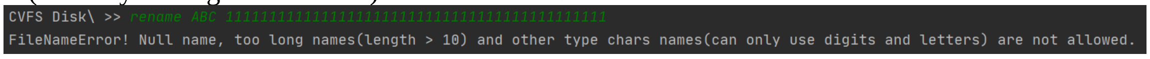
(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.

(3) **MissContentError.** If the input misses one of the components, the system will report: MissContentError! The command of renaming a file should be: rename oldFilename newFilename.



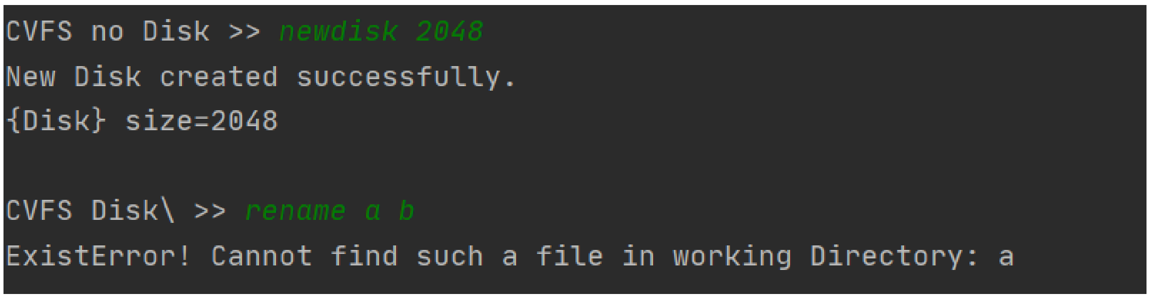
```
CVFS Disk\ >> re ABC WIE
KeywordError! re cannot be recognized.

CVFS Disk\ >> rename
MissContentError! The command of rename a file should be: rename oldFilename newFilename
```

(4) **FilenameError.** If the new name is illegal, the CVFS system will report: FileNameError! Null name, too long names (length > 10) and other type chars names (can only use digits and letters) are not allowed.



```
CVFS Disk\ >> rename ABC 11111111111111111111111111111111111111111111
FileNameError! Null name, too long names(length > 10) and other type chars names(can only use digits and letters) are not allowed.
```

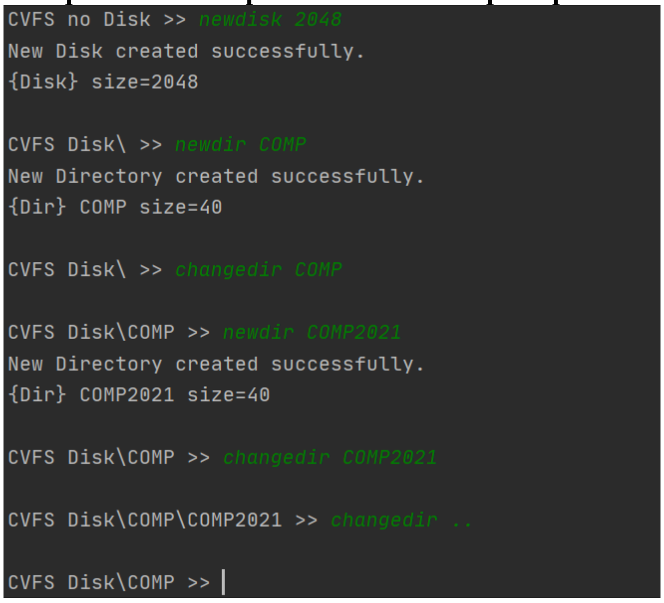
(5) **ExistError.** If there is not a file with the specific name, the CVFS system will report: ExistError! Cannot find such a file in working Directory: a.



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> rename a b
ExistError! Cannot find such a file in working Directory: a
```

2.2.6[REQ6]

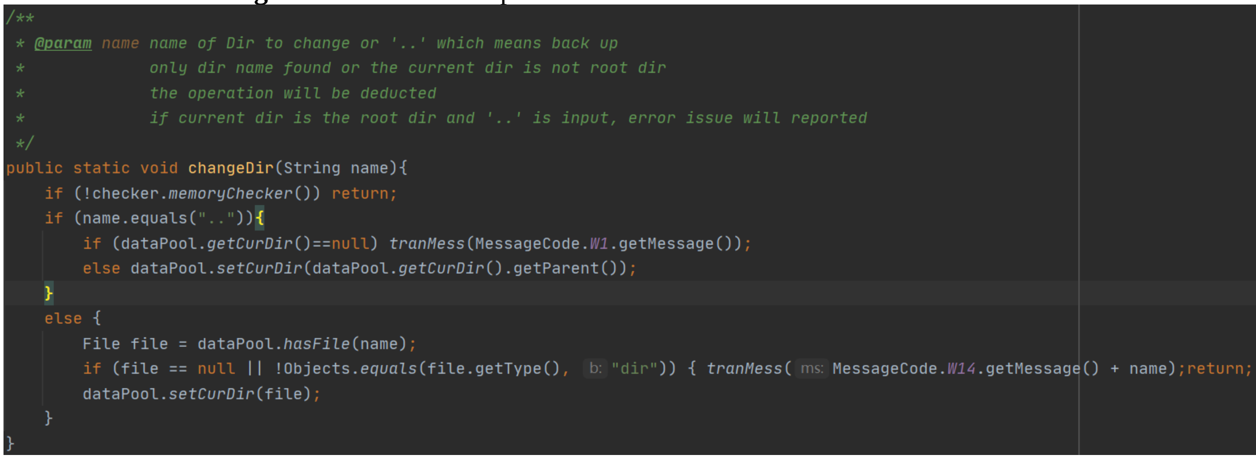
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP >> changedir COMP2021

CVFS Disk\COMP\COMP2021 >> changedir ..

CVFS Disk\COMP >> |
```

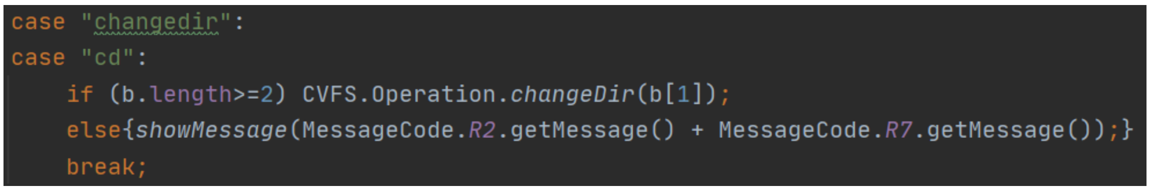
2) Implementation details.

Command “**changeDir** dirName” is implemented as below.



```
/**
 * @param name name of Dir to change or '..' which means back up
 *             only dir name found or the current dir is not root dir
 *             the operation will be deducted
 *             if current dir is the root dir and '..' is input, error issue will reported
 */
public static void changeDir(String name){
    if (!checker.memoryChecker()) return;
    if (name.equals("..")){
        if (dataPool.getCurDir()==null) tranMess(MessageCode.W1.getMessage());
        else dataPool.setCurDir(dataPool.getCurDir().getParent());
    }
    else {
        File file = dataPool.hasFile(name);
        if (file == null || !Objects.equals(file.getType(),  b: "dir")) { tranMess( ms: MessageCode.W14.getMessage() + name);return;
        dataPool.setCurDir(file);
    }
}
```

Reader checks the legality of the input.

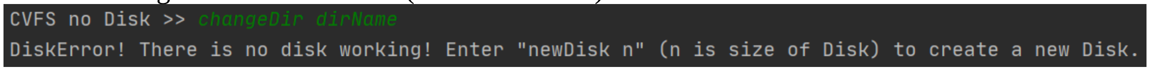


```
case "changedir":
case "cd":
    if (b.length>=2) CVFS.Operation.changeDir(b[1]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R7.getMessage());}
    break;
```

Inside the "**changeDir** dirName" command, we check whether there is a disk working. Then find the file with the specific name. If the name is "..", judge whether current directory is null. If not, use the parent directory of the working directory as the new working directory. If the name is not "..", judge whether there exists the file with type directory. If exists, use that directory as the new working directory.

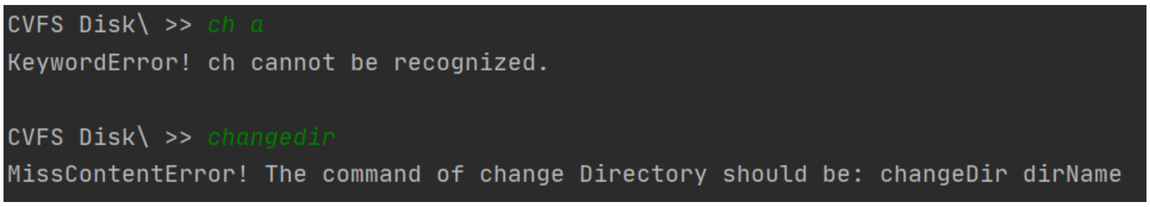
3) Error conditions and how they are handled.
There are 6 kinds of errors:

(1) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



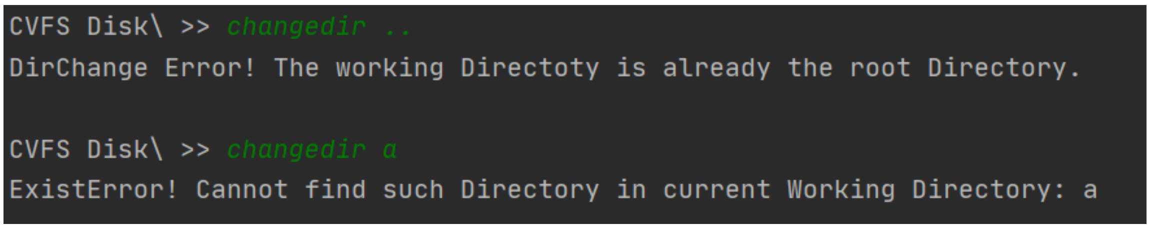
```
CVFS no Disk >> changeDir dirName
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.
(3) **MissContentError.** If the input misses one of the components, the system will report: MissContentError! The command of changing Directory should be: changeDir dirname.



```
CVFS Disk\ >> ch a
KeywordError! ch cannot be recognized.

CVFS Disk\ >> changedir
MissContentError! The command of change Directory should be: changeDir dirName
```

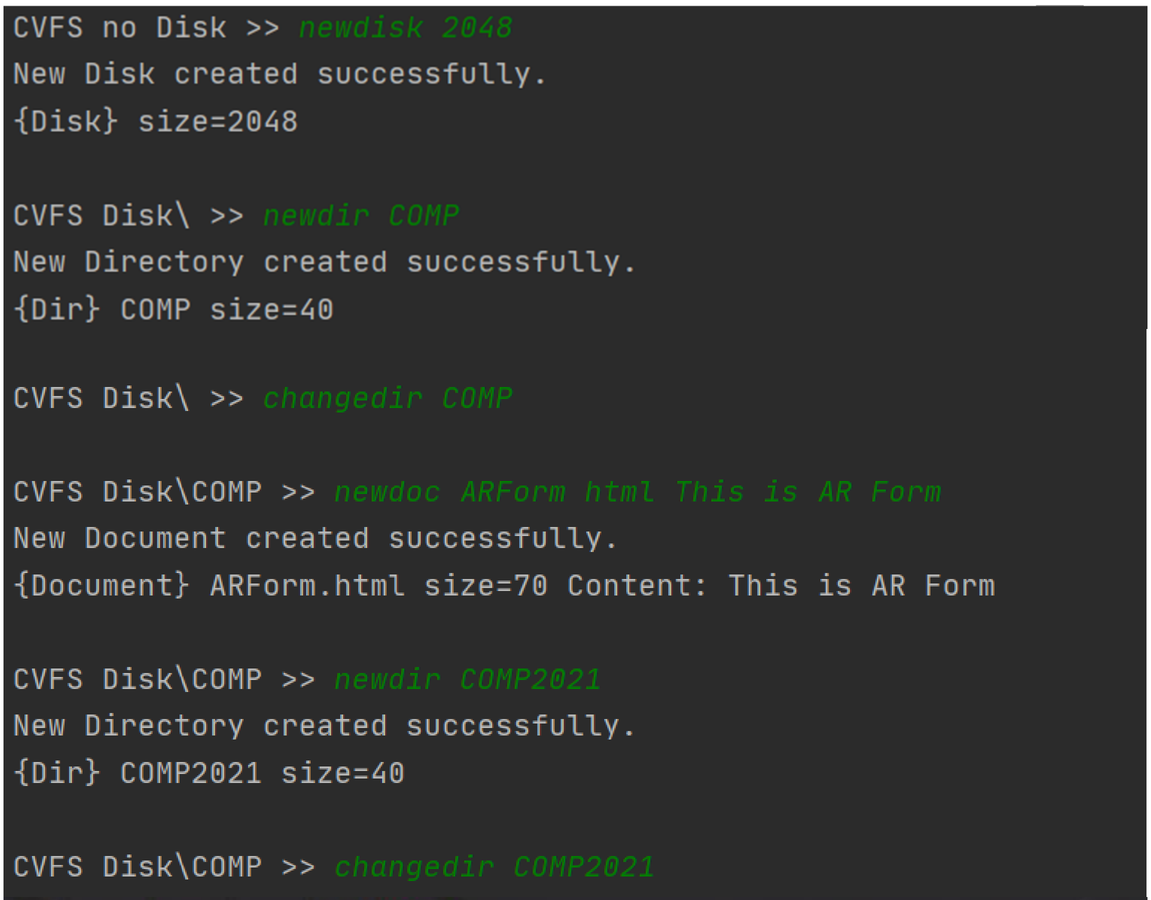
(4) **Directory change error.** While dirname is "..", if current directory is null, the CVFS system will output: DirChange Error! The working Directoty is already the root Directory.
(5) **ExistError .** While dianame is not "..", if there does not exist the file with type directory, the CVFS system will report: ExistError! Cannot find such Directory in current Working Directory: xxx



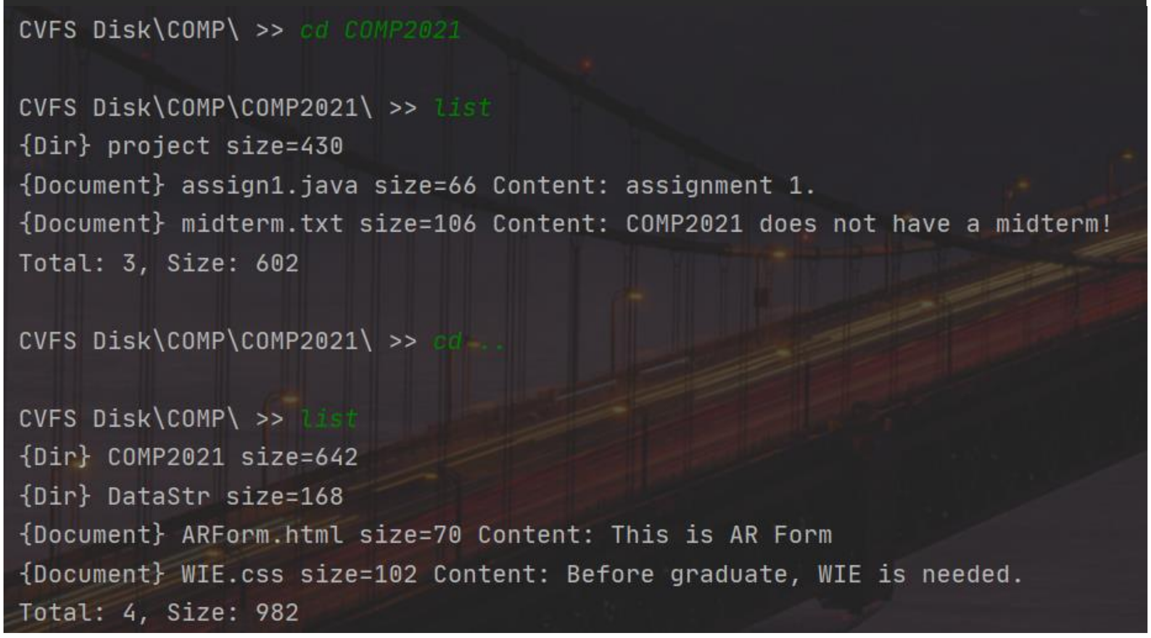
```
CVFS Disk\ >> changedir ..
DirChange Error! The working Directoty is already the root Directory.

CVFS Disk\ >> changedir a
ExistError! Cannot find such Directory in current Working Directory: a
```

2.2.7[REQ7]

1) The requirement is implemented. The sample input and output are shown below:



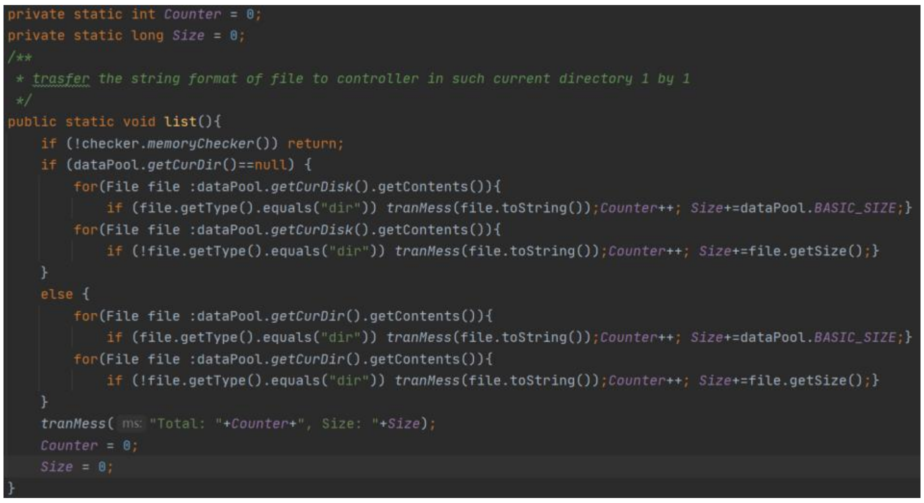
```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP >> newdoc ARForm html This is AR Form
New Document created successfully.
{Document} ARForm.html size=70 Content: This is AR Form

CVFS Disk\COMP >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP >> changedir COMP2021
```



```
CVFS Disk\COMP\ >> cd COMP2021

CVFS Disk\COMP\COMP2021\ >> list
{Dir} project size=430
{Document} assign1.java size=66 Content: assignment 1.
{Document} midterm.txt size=106 Content: COMP2021 does not have a midterm!
Total: 3, Size: 602

CVFS Disk\COMP\COMP2021\ >> cd ..

CVFS Disk\COMP\ >> list
{Dir} COMP2021 size=642
{Dir} DataStr size=168
{Document} ARForm.html size=70 Content: This is AR Form
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.
Total: 4, Size: 982
```

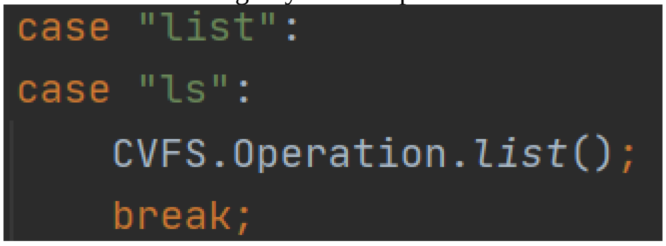
2) Implementation details.

Command “**list**” is implemented as below.



```
private static int Counter = 0;
private static long Size = 0;
/**
 * trasfer the string format of file to controller in such current directory 1 by 1
 */
public static void list(){
    if (!checker.memoryChecker()) return;
    if (dataPool.getCurDir()==null) {
        for(File file :dataPool.getCurDisk().getContents()){
            if (file.getType().equals("dir")) tranMess(file.toString());Counter++; Size+=dataPool.BASIC_SIZE;}
        for(File file :dataPool.getCurDisk().getContents()){
            if (!file.getType().equals("dir")) tranMess(file.toString());Counter++; Size+=file.getSize();}
    }
    else {
        for(File file :dataPool.getCurDir().getContents()){
            if (file.getType().equals("dir")) tranMess(file.toString());Counter++; Size+=dataPool.BASIC_SIZE;}
        for(File file :dataPool.getCurDir().getContents()){
            if (!file.getType().equals("dir")) tranMess(file.toString());Counter++; Size+=file.getSize();}
    }
    tranMess( ms: "Total: "+Counter+", Size: "+Size);
    Counter = 0;
    Size = 0;
}
```

Reader checks the legality of the input.



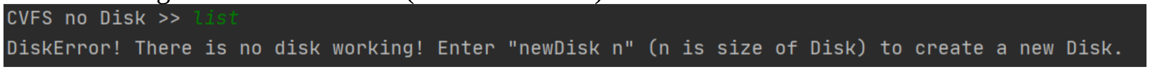
```
case "list":
case "ls":
    CVFS.Operation.list();
    break;
```

Inside the "**list**" command, we first check whether there is a disk working. If current directory is null, we get files from the current disk and add the sizes to get total size. If current directory is not null, we get files directly from the current directory and add the sizes to get total size

3) Error conditions and how they are handled.

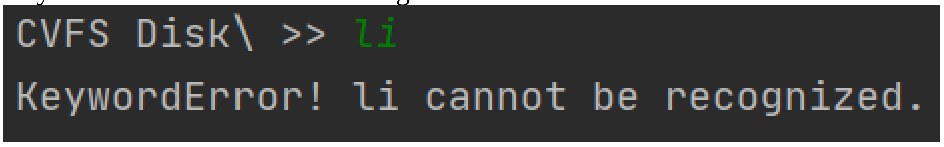
There are 2 errors.

(1) **DiskError**. If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



```
CVFS no Disk >> list
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

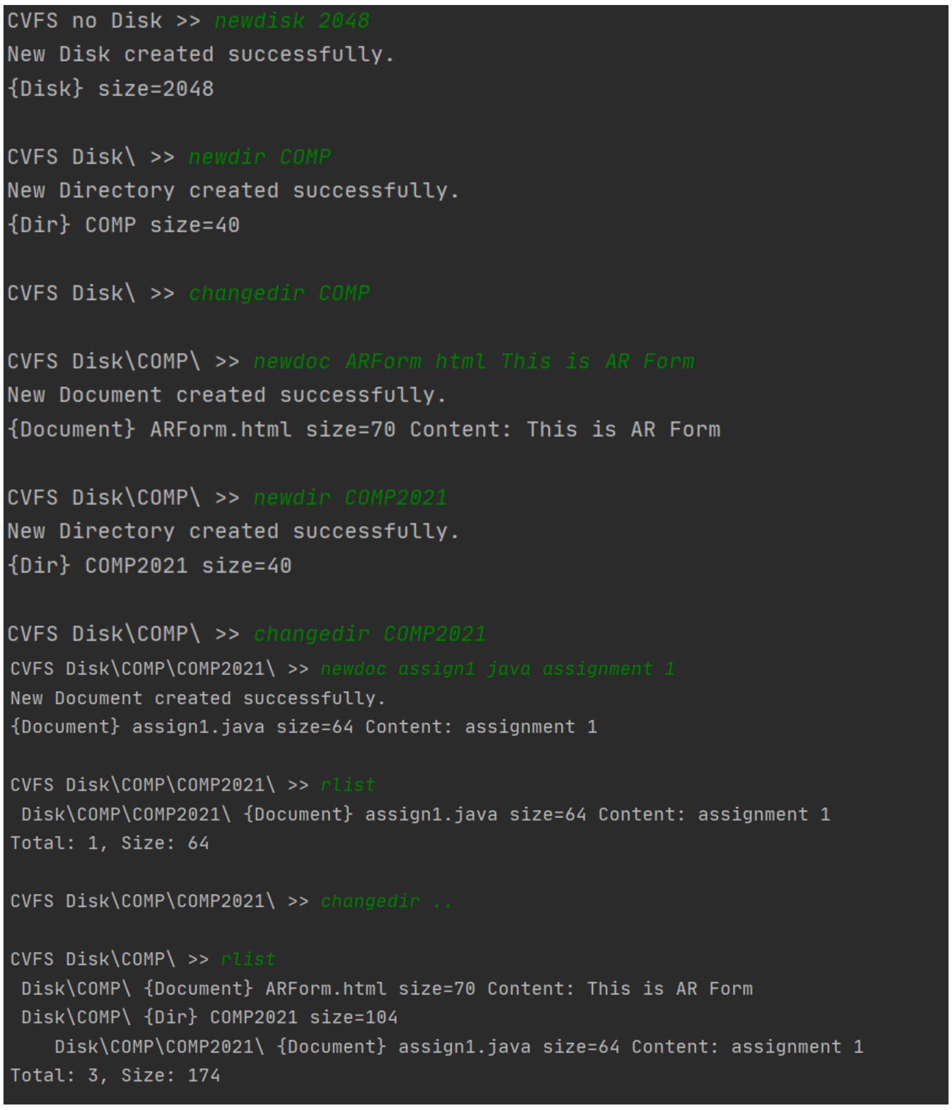
(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



```
CVFS Disk\ >> li
KeywordError! li cannot be recognized.
```

2.2.8[REQ8]

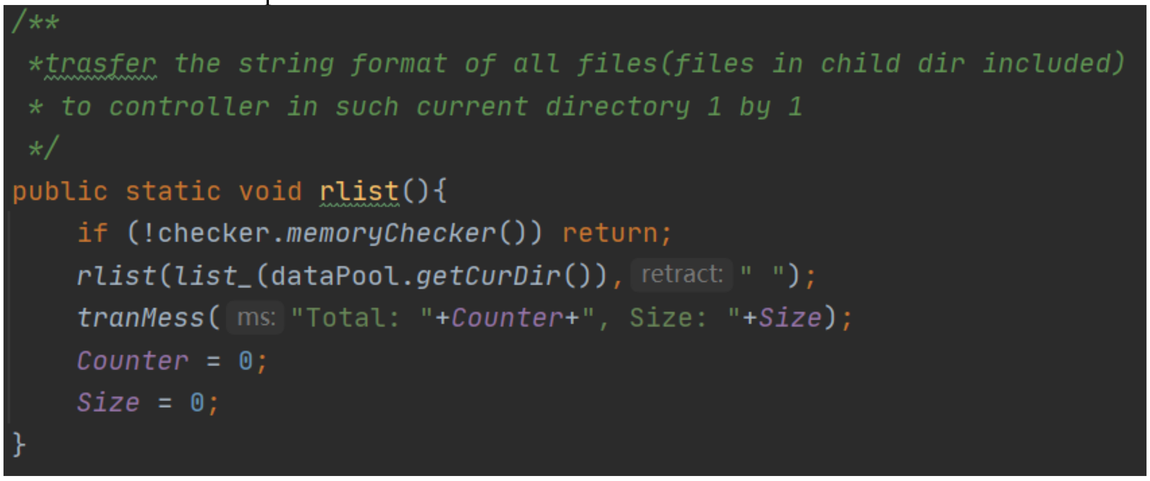
1) The requirement is implemented. The sample input and output are shown below:



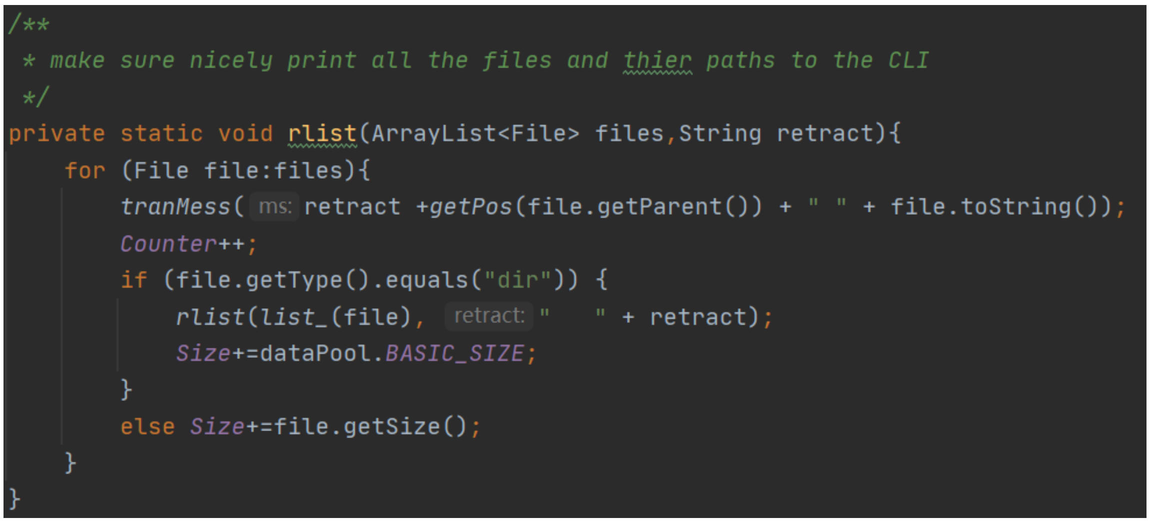
```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP\ >> newdoc ARForm html This is AR Form
New Document created successfully.
{Document} ARForm.html size=70 Content: This is AR Form

CVFS Disk\COMP\ >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP\ >> changedir COMP2021
CVFS Disk\COMP\COMP2021\ >> newdoc assign1 java assignment 1
New Document created successfully.
{Document} assign1.java size=64 Content: assignment 1

CVFS Disk\COMP\COMP2021\ >> rlist
 Disk\COMP\COMP2021\ {Document} assign1.java size=64 Content: assignment 1
Total: 1, Size: 64

CVFS Disk\COMP\COMP2021\ >> changedir ..

CVFS Disk\COMP\ >> rlist
 Disk\COMP\ {Document} ARForm.html size=70 Content: This is AR Form
 Disk\COMP\ {Dir} COMP2021 size=104
    Disk\COMP\COMP2021\ {Document} assign1.java size=64 Content: assignment 1
Total: 3, Size: 174
```

2) Implementation details.

Command “**rlist**” is implemented as below.

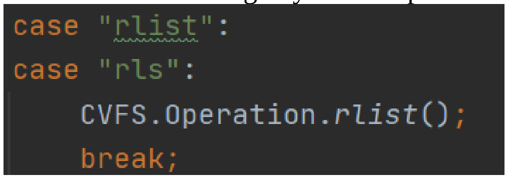


```
/**
 *trasfer the string format of all files(files in child dir included)
 * to controller in such current directory 1 by 1
 */
public static void rlist(){
    if (!checker.memoryChecker()) return;
    rlist(list_(dataPool.getCurDir()), retract: " ");
    tranMess( ms: "Total: "+Counter+", Size: "+Size);
    Counter = 0;
    Size = 0;
}
```



```
/**
 * make sure nicely print all the files and thier paths to the CLI
 */
private static void rlist(ArrayList<File> files,String retract){
    for (File file:files){
        tranMess( ms: retract +getPos(file.getParent()) + " " + file.toString());
        Counter++;
        if (file.getType().equals("dir")) {
            rlist(list_(file),  retract: "    " + retract);
            Size+=dataPool.BASIC_SIZE;
        }
        else Size+=file.getSize();
    }
}
```

Reader checks the legality of the input.



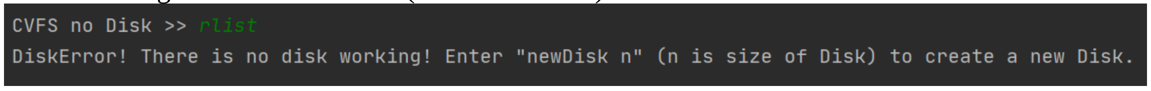
```
case "rlist":
case "rls":
    CVFS.Operation.rlist();
    break;
```

Inside the “**rlist**” command, we check whether there is a disk working. Then we get files from the current directory. If the file type is directory, we run the function again to get files in this children directory. At last, add the sizes to get total size.

3) Error conditions and how they are handled.

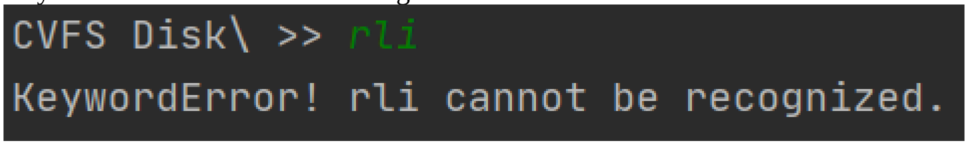
There are 2 errors.

(3) DiskError. If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



```
CVFS no Disk >> rlist
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

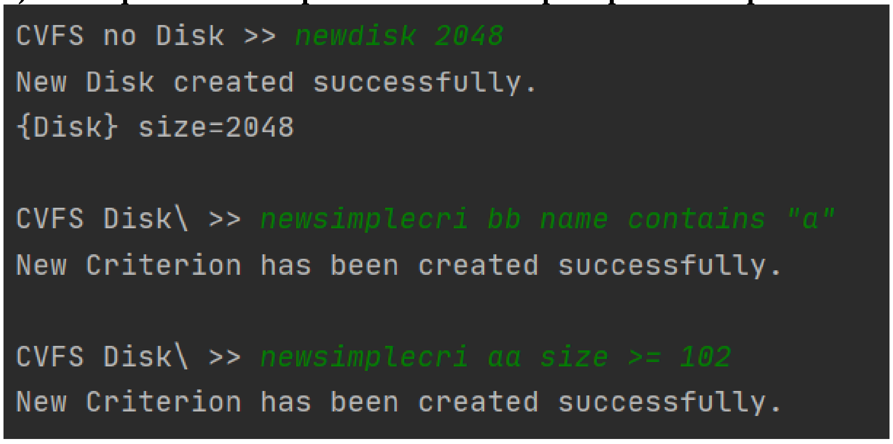
(4) KeywordError. If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



```
CVFS Disk\ >> rli
KeywordError! rli cannot be recognized.
```

2.2.9[REQ9]

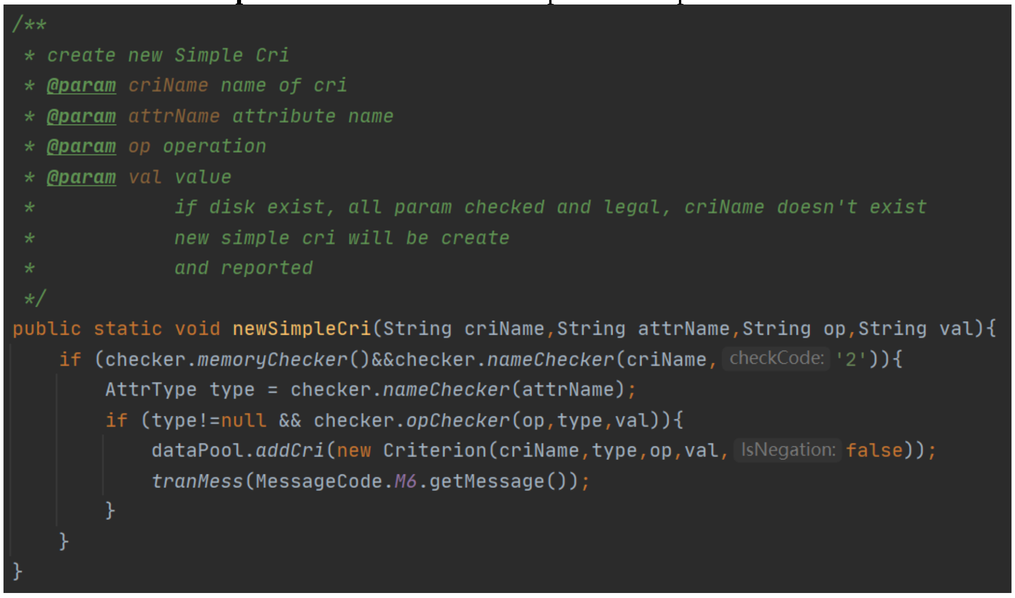
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newsimplecri bb name contains "a"
New Criterion has been created successfully.

CVFS Disk\ >> newsimplecri aa size >= 102
New Criterion has been created successfully.
```

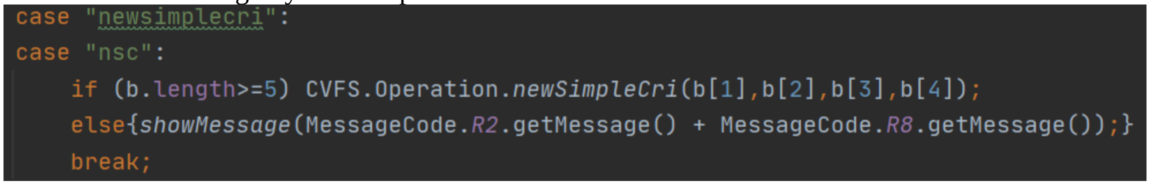
2) Implementation details.

Command “**newSimpleCri** criName attrName op val” is implemented as below.



```java
/**
 * create new Simple Cri
 * @param criName name of cri
 * @param attrName attribute name
 * @param op operation
 * @param val value
 *            if disk exist, all param checked and legal, criName doesn't exist
 *            new simple cri will be create
 *            and reported
 */
public static void newSimpleCri(String criName,String attrName,String op,String val){
    if (checker.memoryChecker()&&checker.nameChecker(criName, checkCode: '2')){
        AttrType type = checker.nameChecker(attrName);
        if (type!=null && checker.opChecker(op,type,val)){
            dataPool.addCri(new Criterion(criName,type,op,val, IsNegation: false));
            tranMess(MessageCode.M6.getMessage());
        }
    }
}
```

Reader checks the legality of the input.



```java
case "newsimplecri":
case "nsc":
    if (b.length>=5) CVFS.Operation.newSimpleCri(b[1],b[2],b[3],b[4]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R8.getMessage());}
    break;
```

Inside the “**newSimpleCri** criName attrName op val” command, we check the criname’s legality and whether there is a disk working. Then check the attrname and convert it to attrtype. If type is not null and type and op follows the following rule:

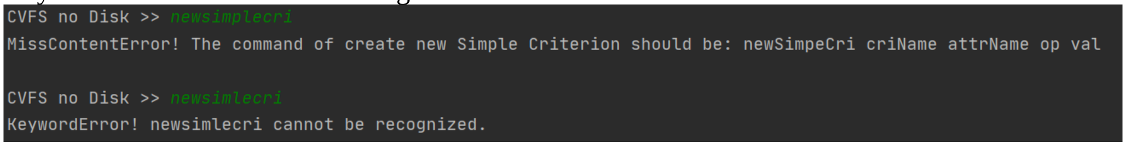
“If attrName is name, op must be contains and val must be a string in double quote;
If attrName is type, op must be equals and val must be a string in double quote;
If attrName is size, op can be >, <, >=, <=, ==, or !=, and val must be an integer.”,

we add the simple criterion to datapool. At last, print the message “New Criterion has been created successfully.”.

3) Error conditions and how they are handled.
There are 7 kinds of errors:

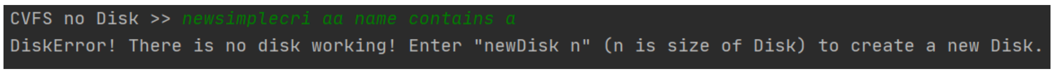
(1) **MissContentError.** If the input misses one of the components, the system will report: MissContentError! The command of create new Simple Criterion should be: newSimpleCri criName attrName op val.

(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



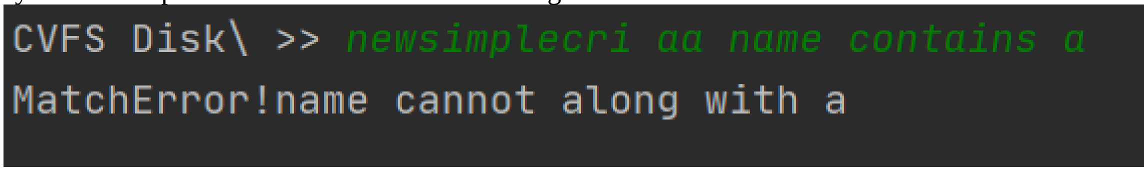
```
CVFS no Disk >> newsimplecri
MissContentError! The command of create new Simple Criterion should be: newSimpeCri criName attrName op val

CVFS no Disk >> newsimlecri
KeywordError! newsimlecri cannot be recognized.
```

(3) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.



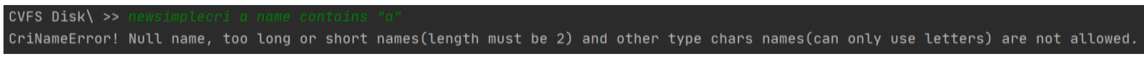
```
CVFS no Disk >> newsimplecri aa name contains a
DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
```

(4) **MatchError.** If the input attrName, op and val does not fit, the CVFS system will report: MatchError! xxx cannot along with xx.



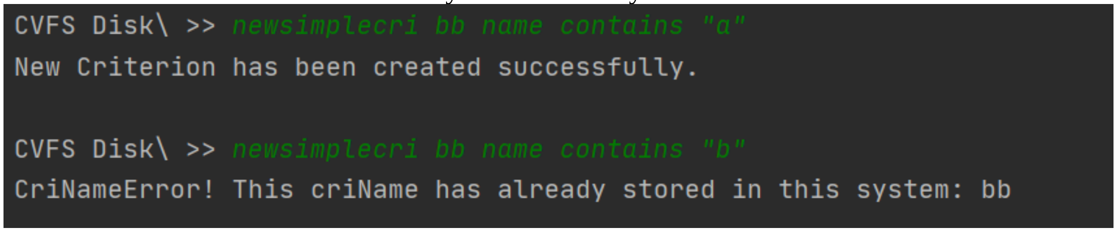
```
CVFS Disk\ >> newsimplecri aa name contains a
MatchError!name cannot along with a
```

(5) **CriNameError.** If the criname is not legal (name length is not 2)m the CVFS system will report: CriNameError! Null name, too long or short names(length must be 2) and other type chars names(can only use letters) are not allowed.



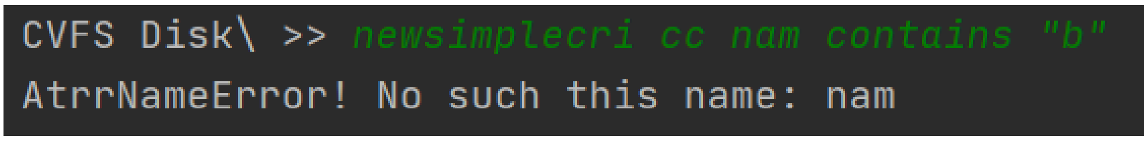
```
CVFS Disk\ >> newsimplecri a name contains "a"
CriNameError! Null name, too long or short names(length must be 2) and other type chars names(can only use letters) are not allowed.
```

(6) If the criname is already created before, the CVFS system will report:
This criName has already stored in this system: xxx



```
CVFS Disk\ >> newsimplecri bb name contains "a"
New Criterion has been created successfully.

CVFS Disk\ >> newsimplecri bb name contains "b"
CriNameError! This criName has already stored in this system: bb
```

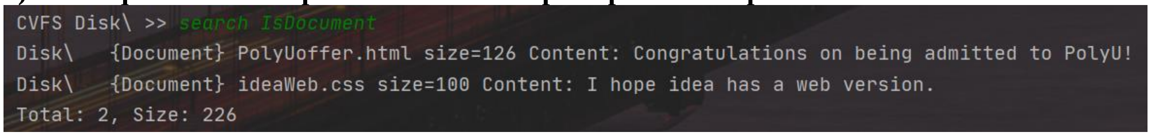
(7) AtrrNameError. If the input AtrrName is not legal, the CVFS system will report:
AtrrNameError! No such this name: xxx



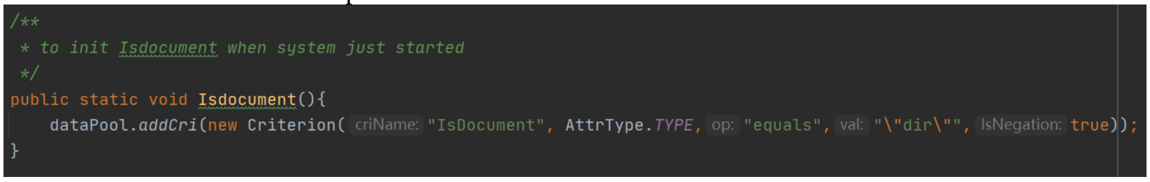
```
CVFS Disk\ >> newsimplecri cc nam contains "b"
AtrrNameError! No such this name: nam
```

2.2.10[REQ10]

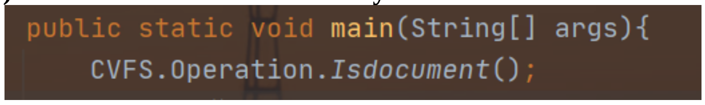
1) The requirement is implemented. The sample input and output are shown below:


```
CVFS Disk\ >> search IsDocument
Disk\   {Document} PolyUoffer.html size=126 Content: Congratulations on being admitted to PolyU!
Disk\   {Document} ideaWeb.css size=100 Content: I hope idea has a web version.
Total: 2, Size: 226
```

2) Implementation details.
Criterion “**IsDocument”** is implemented as below.

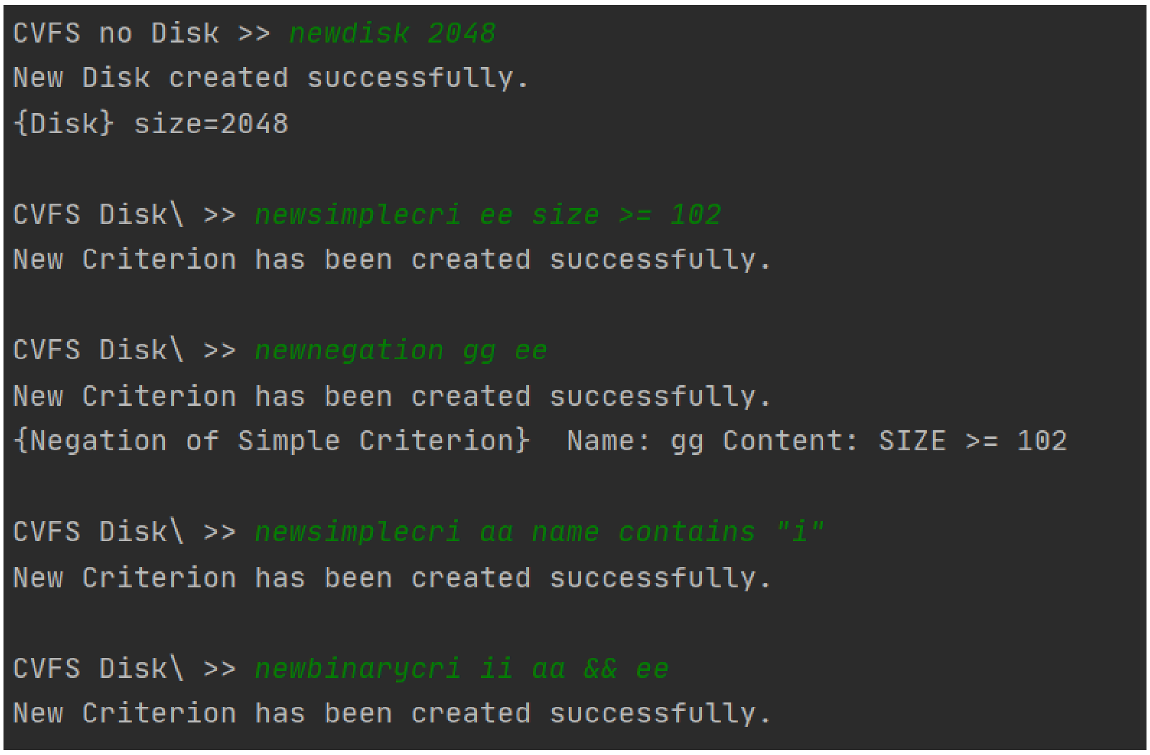

```java
/**
 * to init Isdocument when system just started
 */
public static void Isdocument(){
    dataPool.addCri(new Criterion( criName: "IsDocument", AttrType.TYPE, op: "equals", val: "\"dir\"", IsNegation: true));
}
```
Add the criterion that evaluates to true if not on a directory.

3) Error conditions and how they are handled.

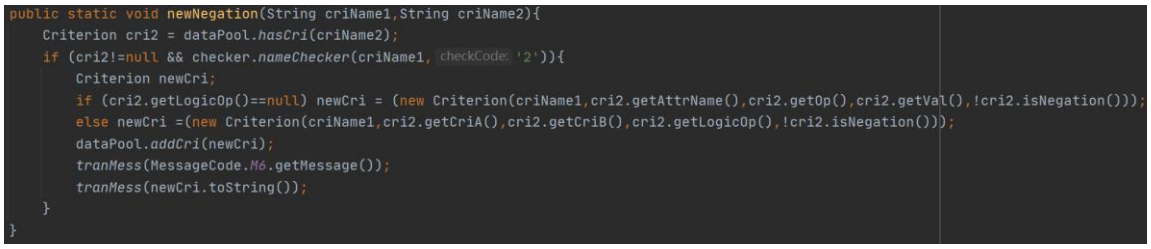

```java
public static void main(String[] args){
    CVFS.Operation.Isdocument();
```
The default simple criterion is added to the system once the system starts.

2.2.11[REQ11]

1) The requirement is implemented. The sample input and output are shown below:

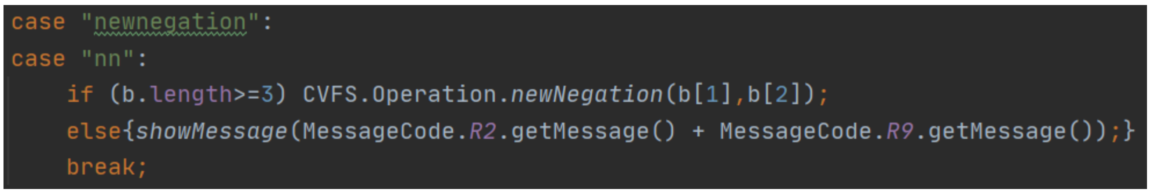

```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newsimplecri ee size >= 102
New Criterion has been created successfully.

CVFS Disk\ >> newnegation gg ee
New Criterion has been created successfully.
{Negation of Simple Criterion}  Name: gg Content: SIZE >= 102

CVFS Disk\ >> newsimplecri aa name contains "i"
New Criterion has been created successfully.

CVFS Disk\ >> newbinarycri ii aa && ee
New Criterion has been created successfully.
```

2) Implementation details.
For command “**newNegation** criName1 criName2”, we have implemented it as below.



```
public static void newNegation(String criName1,String criName2){
    Criterion cri2 = dataPool.hasCri(criName2);
    if (cri2!=null && checker.nameChecker(criName1, checkCode: '2')){
        Criterion newCri;
        if (cri2.getLogicOp()==null) newCri = (new Criterion(criName1,cri2.getAttrName(),cri2.getOp(),cri2.getVal(),!cri2.isNegation()));
        else newCri =(new Criterion(criName1,cri2.getCriA(),cri2.getCriB(),cri2.getLogicOp(),!cri2.isNegation()));
        dataPool.addCri(newCri);
        tranMess(MessageCode.M6.getMessage());
        tranMess(newCri.toString());
    }
}
```

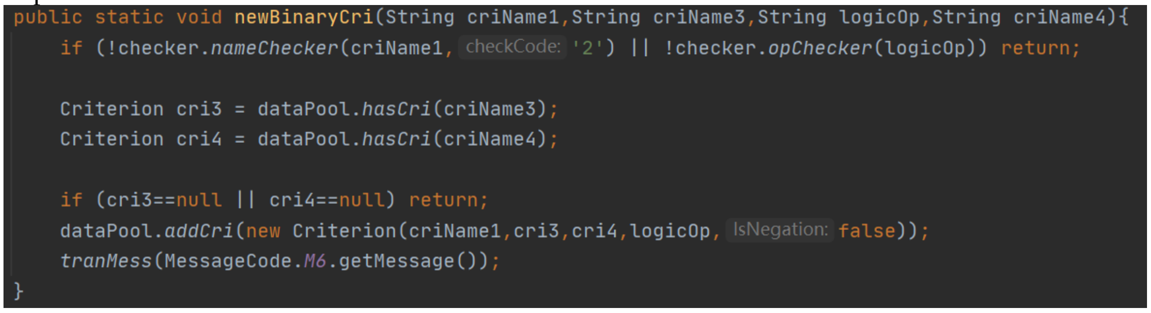
Reader checks the legality of the input.



```
case "newnegation":
case "nn":
    if (b.length>=3) CVFS.Operation.newNegation(b[1],b[2]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R9.getMessage());}
    break;
```

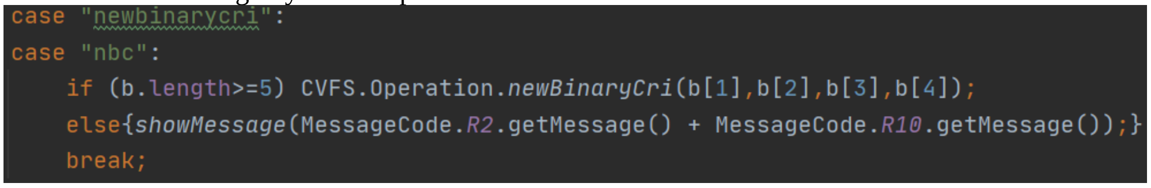
After decided cri2 is neither null nor the same as criName1, we construct the composite criterion referenced by criName1. If the **logicOp** of criterion2 is not null, we construct the negation using the binary way, else we use the simple way.

For command "**newBinaryCri** criName1 criName3 logicOp criName4", we have implemented it as below.



```
public static void newBinaryCri(String criName1,String criName3,String logicOp,String criName4){
    if (!checker.nameChecker(criName1, checkCode: '2') || !checker.opChecker(logicOp)) return;

    Criterion cri3 = dataPool.hasCri(criName3);
    Criterion cri4 = dataPool.hasCri(criName4);

    if (cri3==null || cri4==null) return;
    dataPool.addCri(new Criterion(criName1,cri3,cri4,logicOp, IsNegation: false));
    tranMess(MessageCode.M6.getMessage());
}
```

Reader checks the legality of the input.



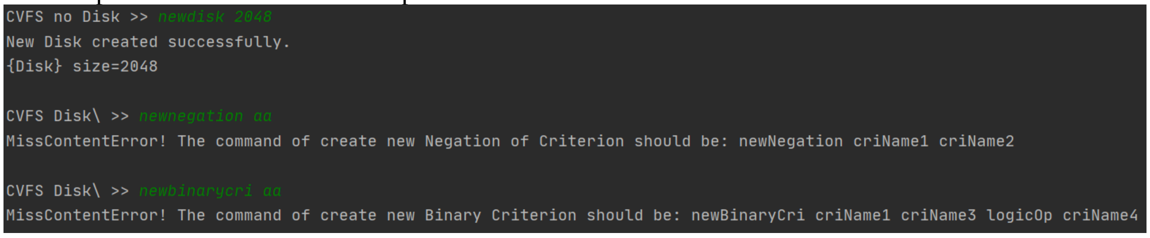
```
case "newbinarycri":
case "nbc":
    if (b.length>=5) CVFS.Operation.newBinaryCri(b[1],b[2],b[3],b[4]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R10.getMessage());}
    break;
```

"**newBinaryCri** criName1 criName3 logicOp criName4" construct criName3 op criName4 after deciding criName3 and criName4 are two existing and legal criteria and their **logicOp** is either && or ||.

3) Error conditions and how they are handled.

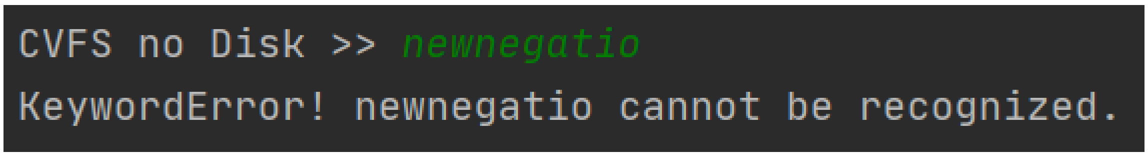
There are 4 kinds of errors:

(1) MissContentError. If the input misses one of the components, the system will report: MissContentError! The command of create new Simple Criterion should be: newSimpleCri criName attrName op val.



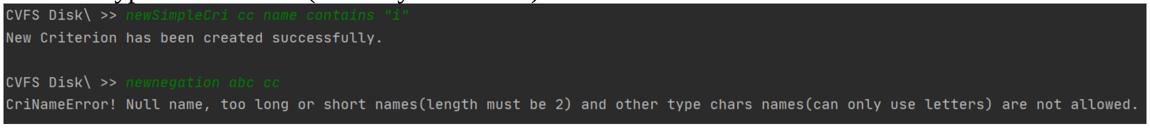
```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newnegation aa
MissContentError! The command of create new Negation of Criterion should be: newNegation criName1 criName2

CVFS Disk\ >> newbinarycri aa
MissContentError! The command of create new Binary Criterion should be: newBinaryCri criName1 criName3 logicOp criName4
```

(2) KeywordError. If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



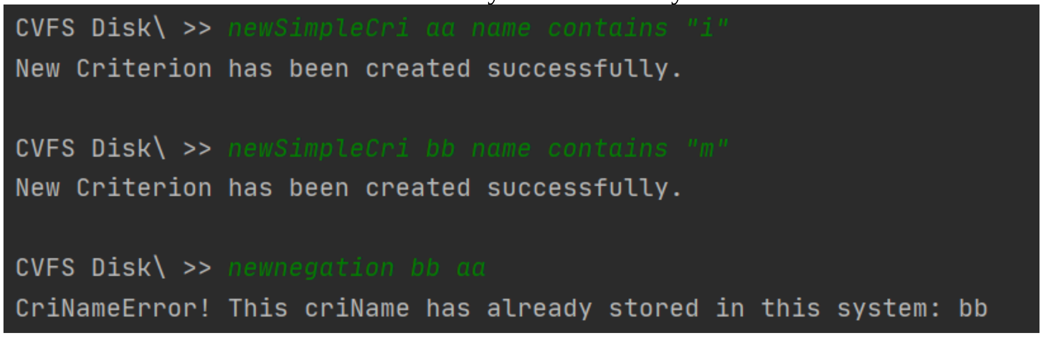
```
CVFS no Disk >> newnegatio
KeywordError! newnegatio cannot be recognized.
```

(3) CriNameError. When criName is checked illegal by the namechecker, the CVFS system will report: CriNameError! Null name, too long or too short names (length must be 2) and other type chars names (can only use letters) are not allowed.



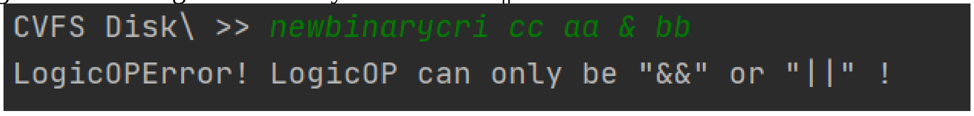
```
CVFS Disk\ >> newSimpleCri cc name contains "i"
New Criterion has been created successfully.

CVFS Disk\ >> newnegation abc cc
CriNameError! Null name, too long or short names(length must be 2) and other type chars names(can only use letters) are not allowed.
```

If criname2 is already stored in datapool, the CVFS system will report: CriNameError! This criName has already stored in this system: xxx



```
CVFS Disk\ >> newSimpleCri aa name contains "i"
New Criterion has been created successfully.

CVFS Disk\ >> newSimpleCri bb name contains "m"
New Criterion has been created successfully.

CVFS Disk\ >> newnegation bb aa
CriNameError! This criName has already stored in this system: bb
```

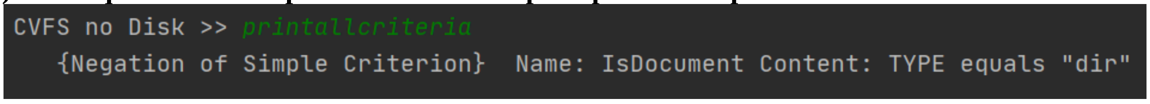
(4) LogicOPError. If logicOp is not && or ||, the CVFS system will report: LogicOPError! LogicOP can only be "&&" or "||"!



```
CVFS Disk\ >> newbinarycri cc aa & bb
LogicOPError! LogicOP can only be "&&" or "||" !
```

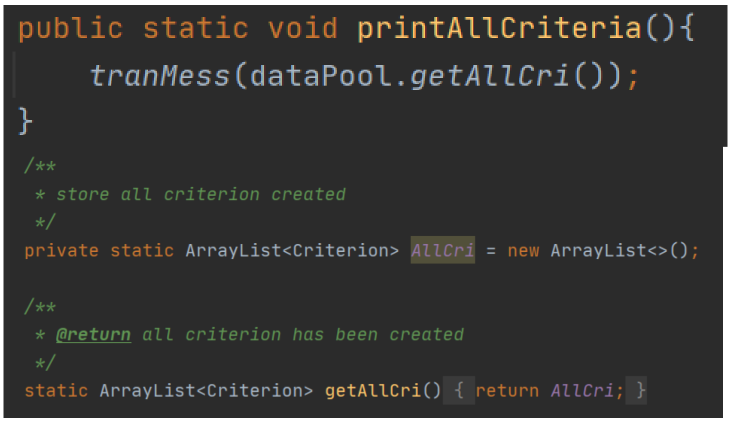
2.2.12[REQ12]

1) The requirement is implemented. The sample input and output are shown below:



```
CVFS no Disk >> printallcriteria
   {Negation of Simple Criterion}  Name: IsDocument Content: TYPE equals "dir"
```

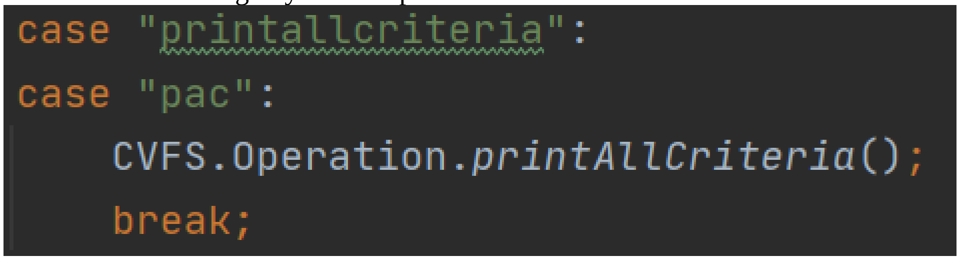
2) Implementation details.
Command "**printAllCriteria**" is implemented as below.



```
public static void printAllCriteria(){
    tranMess(dataPool.getAllCri());
}
```

```
/**
 * store all criterion created
 */
private static ArrayList<Criterion> AllCri = new ArrayList<>();

/**
 * @return all criterion has been created
 */
static ArrayList<Criterion> getAllCri() { return AllCri; }
```

Reader checks the legality of the input.

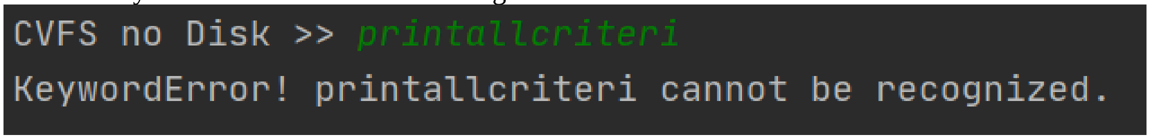


```
case "printallcriteria":
case "pac":
    CVFS.Operation.printAllCriteria();
    break;
```

This command will print all criterions that are stored in the datapool.

3) Error conditions and how they are handled.

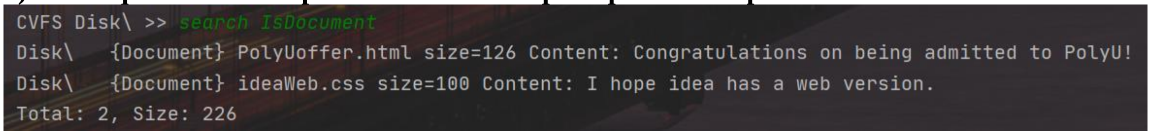
(1)KeywordError. If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.



```
CVFS no Disk >> printallcriteri
KeywordError! printallcriteri cannot be recognized.
```

2.2.13[REQ13]

1) The requirement is implemented. The sample input and output are shown below:

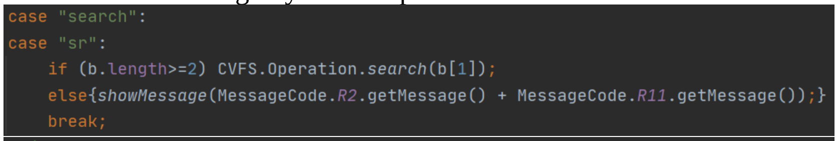


```
CVFS Disk\ >> search IsDocument
Disk\   {Document} PolyUoffer.html size=126 Content: Congratulations on being admitted to PolyU!
Disk\   {Document} ideaWeb.css size=100 Content: I hope idea has a web version.
Total: 2, Size: 226
```

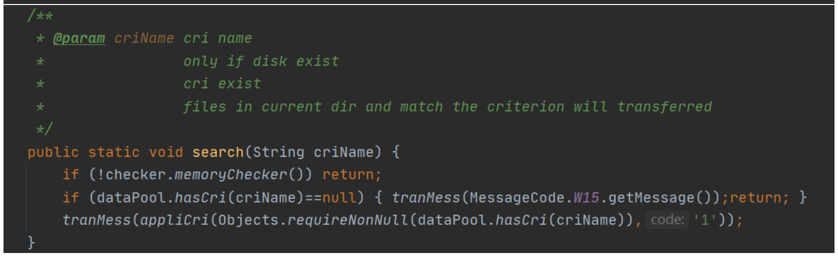
2) Implementation details.

Command "**search** criName" is implemented as below.

Reader checks the legality of the input.

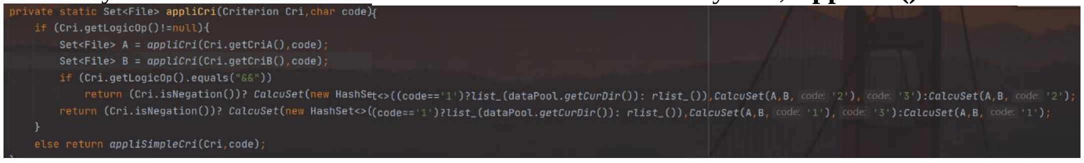


```
case "search":
case "sr":
    if (b.length>=2) CVFS.Operation.search(b[1]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R11.getMessage());}
    break;
```



```
/**
 * @param criName cri name
 *                only if disk exist
 *                cri exist
 *                files in current dir and match the criterion will transferred
 */
public static void search(String criName) {
    if (!checker.memoryChecker()) return;
    if (dataPool.hasCri(criName)==null) { tranMess(MessageCode.W15.getMessage());return; }
    tranMess(appliCri(Objects.requireNonNull(dataPool.hasCri(criName)), code: '1'));
}
```

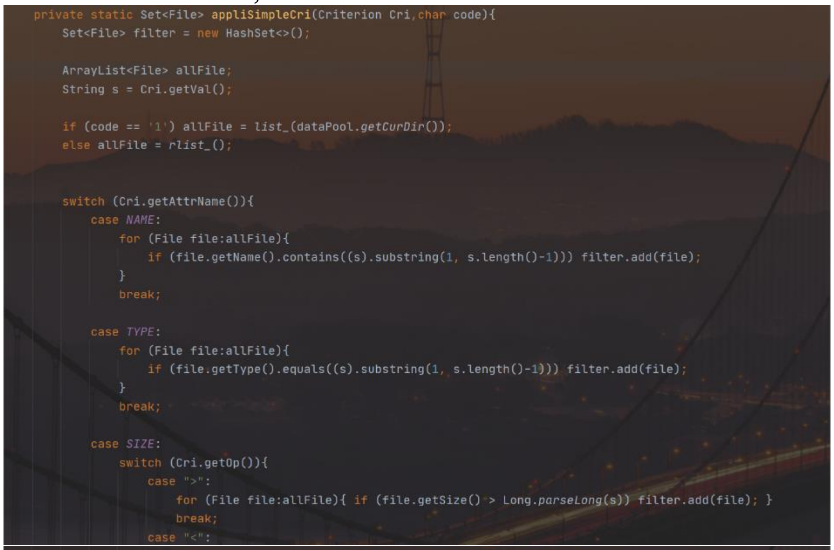
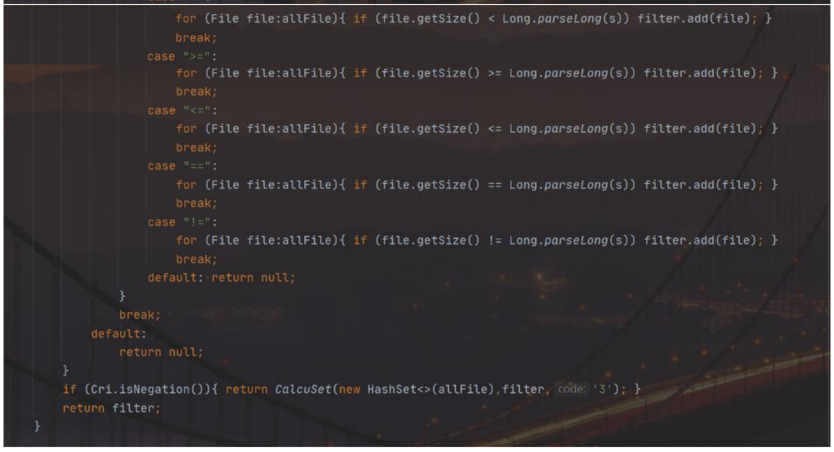
If system has disk and **criName** can be found in the system, **appliCri()** will be called



```
private static Set<File> appliCri(Criterion Cri,char code){
    if (Cri.getLogicOp()!=null){
        Set<File> A = appliCri(Cri.getCriA(),code);
        Set<File> B = appliCri(Cri.getCriB(),code);
        if (Cri.getLogicOp().equals("&&"))
            return (Cri.isNegation())? CalcuSet(new HashSet<>((code=='1')?list_(dataPool.getCurDir()): rlist_()),CalcuSet(A,B, code: '2'), code: '3'):CalcuSet(A,B, code: '2');
        return (Cri.isNegation())? CalcuSet(new HashSet<>((code=='1')?list_(dataPool.getCurDir()): rlist_()),CalcuSet(A,B, code: '1'), code: '3'):CalcuSet(A,B, code: '1');
    }
    else return appliSimpleCri(Cri,code);
```

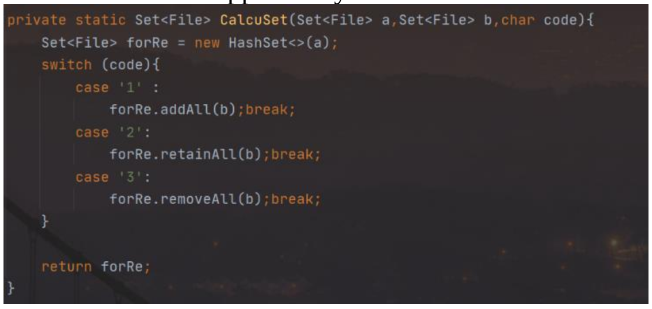
This is a method with Recursion which has been considered a **binary Criterion** may combined by **binary Criterion**, so **set calculation** and recursion are both needed to figure out the search result. In such method, if the subCriterion is still a **binary Criterion**, recursion will be conducted. If the subCriterion is a simple Criterion, then the below method will be conducted.

Code ‘1’ is for **search**, while ‘2’ is for **research**.

```java
    private static Set<File> appliSimpleCri(Criterion Cri,char code){
        Set<File> filter = new HashSet<>();

        ArrayList<File> allFile;
        String s = Cri.getVal();

        if (code == '1') allFile = list_(dataPool.getCurDir());
        else allFile = rlist_();


        switch (Cri.getAttrName()){
            case NAME:
                for (File file:allFile){
                    if (file.getName().contains((s).substring(1, s.length()-1))) filter.add(file);
                }
                break;

            case TYPE:
                for (File file:allFile){
                    if (file.getType().equals((s).substring(1, s.length()-1))) filter.add(file);
                }
                break;

            case SIZE:
                switch (Cri.getOp()){
                    case ">":
                        for (File file:allFile){ if (file.getSize() > Long.parseLong(s)) filter.add(file); }
                        break;
                    case "<":
                        for (File file:allFile){ if (file.getSize() < Long.parseLong(s)) filter.add(file); }
                        break;
                    case ">=":
                        for (File file:allFile){ if (file.getSize() >= Long.parseLong(s)) filter.add(file); }
                        break;
                    case "<=":
                        for (File file:allFile){ if (file.getSize() <= Long.parseLong(s)) filter.add(file); }
                        break;
                    case "==":
                        for (File file:allFile){ if (file.getSize() == Long.parseLong(s)) filter.add(file); }
                        break;
                    case "!=":
                        for (File file:allFile){ if (file.getSize() != Long.parseLong(s)) filter.add(file); }
                        break;
                    default: return null;
                }
                break;
            default:
                return null;
        }
        if (Cri.isNegation()){ return CalcuSet(new HashSet<>(allFile),filter, code: '3'); }
        return filter;
    }
```

(String)**Contains()** used for name contains and (String)**equal()** used for type equals.
To Size op val, switch is used to specify which operation is.
Set calculation is supported by method below:



```java
private static Set<File> CalcuSet(Set<File> a,Set<File> b,char code){
    Set<File> forRe = new HashSet<>(a);
    switch (code){
        case '1' :
            forRe.addAll(b);break;
        case '2':
            forRe.retainAll(b);break;
        case '3':
            forRe.removeAll(b);break;
    }

    return forRe;
}
```

Code ‘1’ for Union, code ‘2’ for Intersection and code ‘3’ for difference.

The key rule is, for every Criterion searchs, get all files from **list_()** first, then keep eligible ones, when comes another one, if logicOp is &&, then take the intersection of the two, otherwise, take the Union of the two.
If the Criterion is a negation of another one, then get the set of another one, let universal set to minus the set, then get the negation.

3) Error conditions and how they are handled.

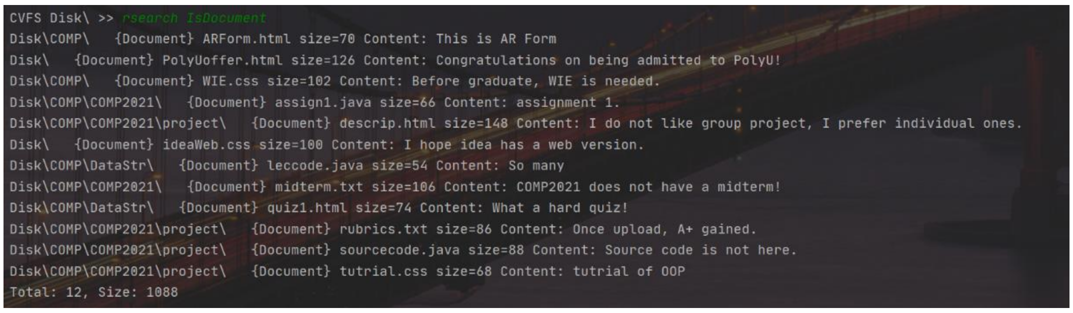
(1) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.
(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.

(3) **ExistError!** If the criName is not stored, the CVFS will report:
ExistError! Cannot find such a Criterion named: XXX

(4) **MissContentError.** If the input misses one of the components, the system will report:
MissContentError! The command of search files in current working directory by specified Criterion should be: Search CriName

2.2.14[REQ14]

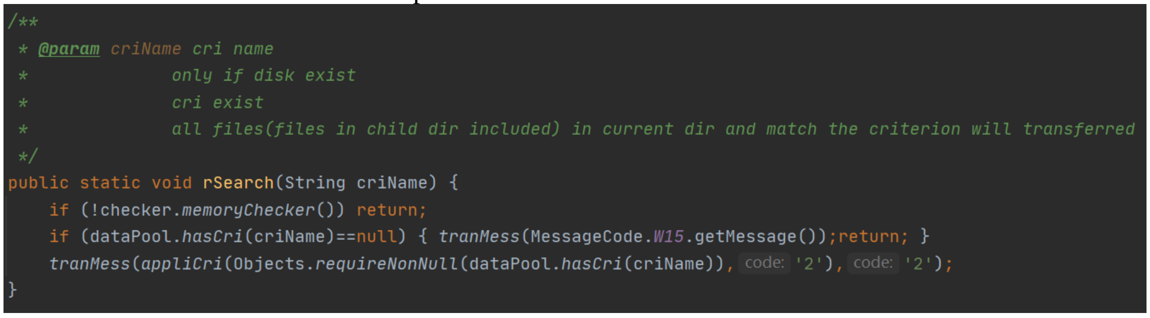
1) The requirement is implemented. The sample input and output are shown below:



```
CVFS Disk\ >> rsearch IsDocument
Disk\COMP\   {Document} ARForm.html size=70 Content: This is AR Form
Disk\   {Document} PolyUoffer.html size=126 Content: Congratulations on being admitted to PolyU!
Disk\COMP\   {Document} WIE.css size=102 Content: Before graduate, WIE is needed.
Disk\COMP\COMP2021\   {Document} assign1.java size=66 Content: assignment 1.
Disk\COMP\COMP2021\project\   {Document} descrip.html size=148 Content: I do not like group project, I prefer individual ones.
Disk\   {Document} ideaWeb.css size=100 Content: I hope idea has a web version.
Disk\COMP\DataStr\   {Document} leccode.java size=54 Content: So many
Disk\COMP\COMP2021\   {Document} midterm.txt size=106 Content: COMP2021 does not have a midterm!
Disk\COMP\DataStr\   {Document} quiz1.html size=74 Content: What a hard quiz!
Disk\COMP\COMP2021\project\   {Document} rubrics.txt size=86 Content: Once upload, A+ gained.
Disk\COMP\COMP2021\project\   {Document} sourcecode.java size=88 Content: Source code is not here.
Disk\COMP\COMP2021\project\   {Document} tutrial.css size=68 Content: tutrial of OOP
Total: 12, Size: 1088
```

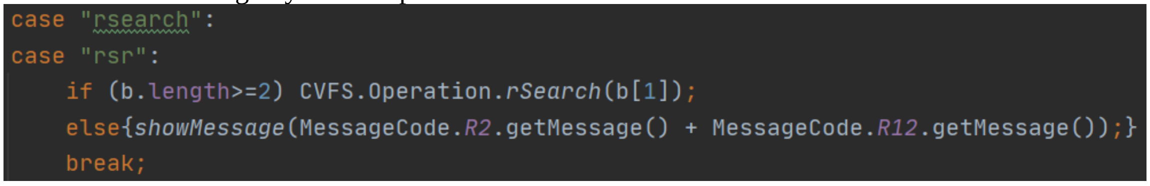
2) Implementation details.

Command "**rSearch** criName" is implemented as below.



```
/**
 * @param criName cri name
 *              only if disk exist
 *              cri exist
 *              all files(files in child dir included) in current dir and match the criterion will transferred
 */
public static void rSearch(String criName) {
    if (!checker.memoryChecker()) return;
    if (dataPool.hasCri(criName)==null) { tranMess(MessageCode.W15.getMessage());return; }
    tranMess(appliCri(Objects.requireNonNull(dataPool.hasCri(criName)), code: '2'), code: '2');
}
```

Reader checks the legality of the input.



```
case "rsearch":
case "rsr":
    if (b.length>=2) CVFS.Operation.rSearch(b[1]);
    else{showMessage(MessageCode.R2.getMessage() + MessageCode.R12.getMessage());}
    break;
```

The key rules of **rSearch** is the same of **search**, the only different of them is: the universal set. The list_() returns the universal set for **search** while rlist_() returns the universal set for **rSearch**.

3) Error conditions and how they are handled.

(1) **DiskError.** If there is no disk working, the system will report: DiskError! There is no disk working! Enter "newDisk n" (n is size of Disk) to create a new Disk.

(2) **KeywordError.** If the command is wrong, the CVFS system will report: KeywordError! xxx cannot be recognized.

(3) **ExistError!** If the criName is not stored, the CVFS will report:
ExistError! Cannot find such a Criterion named: XXX

(4) **MissContentError.** If the input misses one of the components, the system will report:
MissContentError! The command of search all files in current working directory by specified Criterion should be: rSearch CriName

2.2.15 [BON1]

1) The requirement is not implemented.

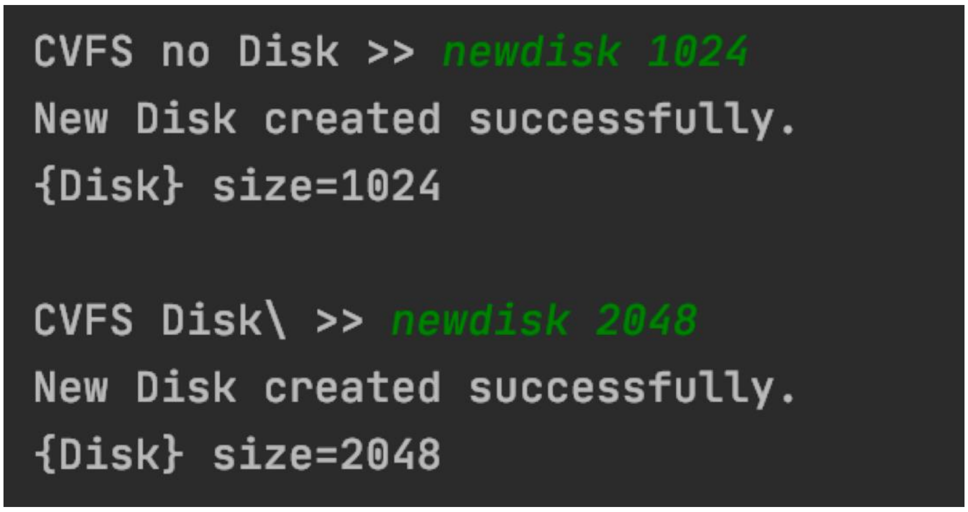
2.2.16 [BON2]

1) The requirement is not implemented.

3 User Manual

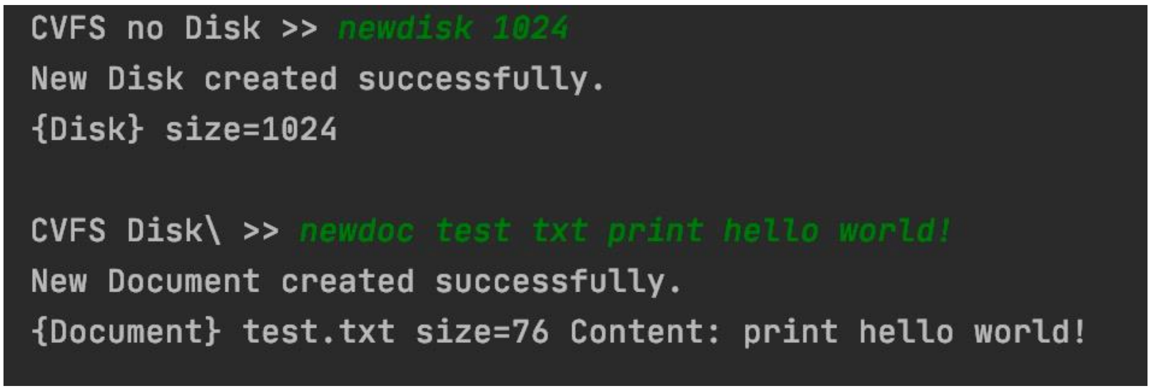
In this section, we explain how the CVFS works from a user's perspective.
Describe here all the commands that the system supports. For each command, use screenshots to show the results under different input

1. **Create a new disk.**
   If you want to create a new disk, just implement this program, and input **newdisk diskname** to create a new disk. (Notice: if there exists a disk, the command prompt will show as: **CVFS no Disk**\ >>, else will show **CVFS Disk**\ >>) If you successfully create a new disk, the system will hint like: New Disk created successfully. The screenshot is shown below:



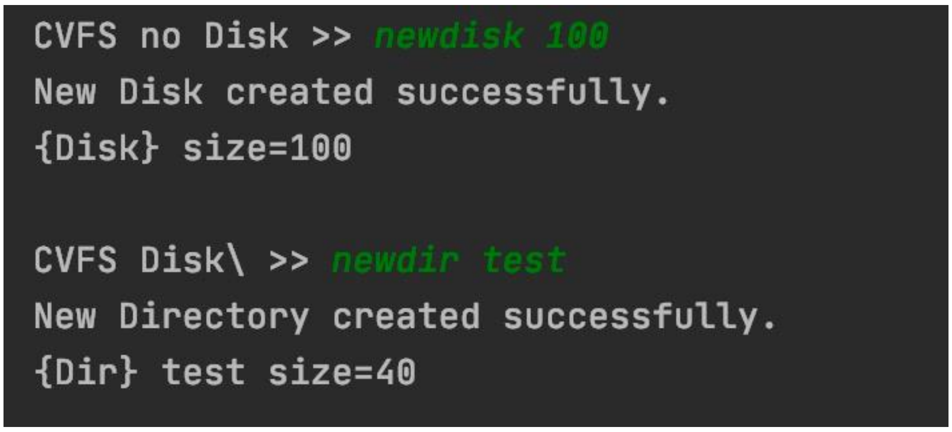
```
CVFS no Disk >> newdisk 1024
New Disk created successfully.
{Disk} size=1024

CVFS Disk\ >> newdisk 2048
New Disk created successfully.
{Disk} size=2048
```

2. **Create a new document.**
   If you want to create a new document, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, you should input: newdoc filename filetype content to create a new document. If you successfully create a new document, the system will show a hint like: **New Document created successfully**. And show your document and its name, type, size and content like: **{Document} xxx.xx size=xx Content: xxxxx**. The screenshot is shown below:



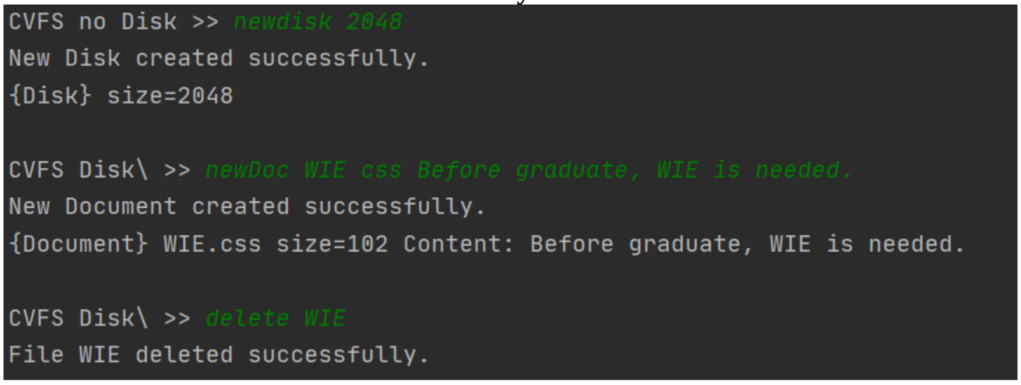
```
CVFS no Disk >> newdisk 1024
New Disk created successfully.
{Disk} size=1024

CVFS Disk\ >> newdoc test txt print hello world!
New Document created successfully.
{Document} test.txt size=76 Content: print hello world!
```

3. **Create a new directory.**
   If you want to create a new directory, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, you should input: newdir directoryname to create a new directory. If you successfully create a new directory, the system will show a hint like: **New Directory created successfully**. And show your directory and its name, size like: **{Dir} xxx size=xx**. The screenshot is shown below:



```
CVFS no Disk >> newdisk 100
New Disk created successfully.
{Disk} size=100

CVFS Disk\ >> newdir test
New Directory created successfully.
{Dir} test size=40
```

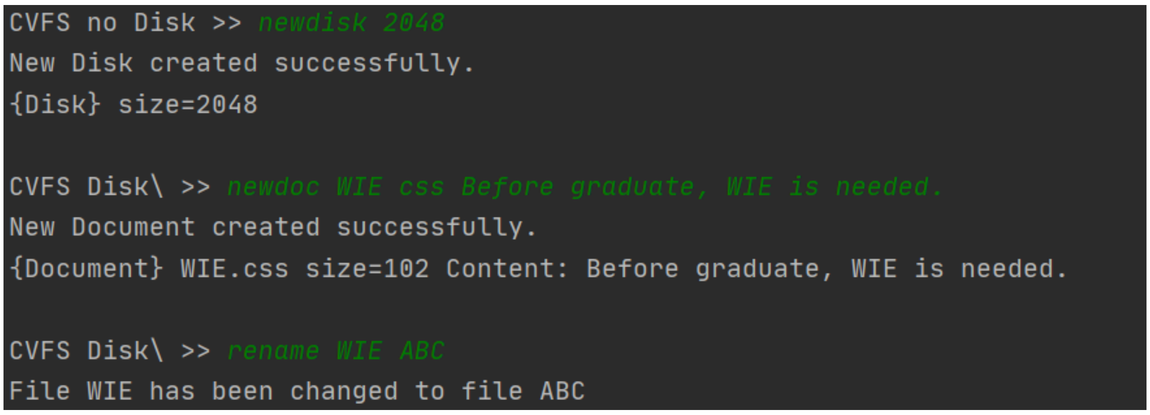
4. **Delete a document.**
   If you want to delete a new document, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, then create a new document. After that, you should input: delete filename to delete the document. If you successfully delete the document, the system will show a hint like: **File xxx deleted successfully**. The screenshot is shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newDoc WIE css Before graduate, WIE is needed.
New Document created successfully.
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.

CVFS Disk\ >> delete WIE
File WIE deleted successfully.
```

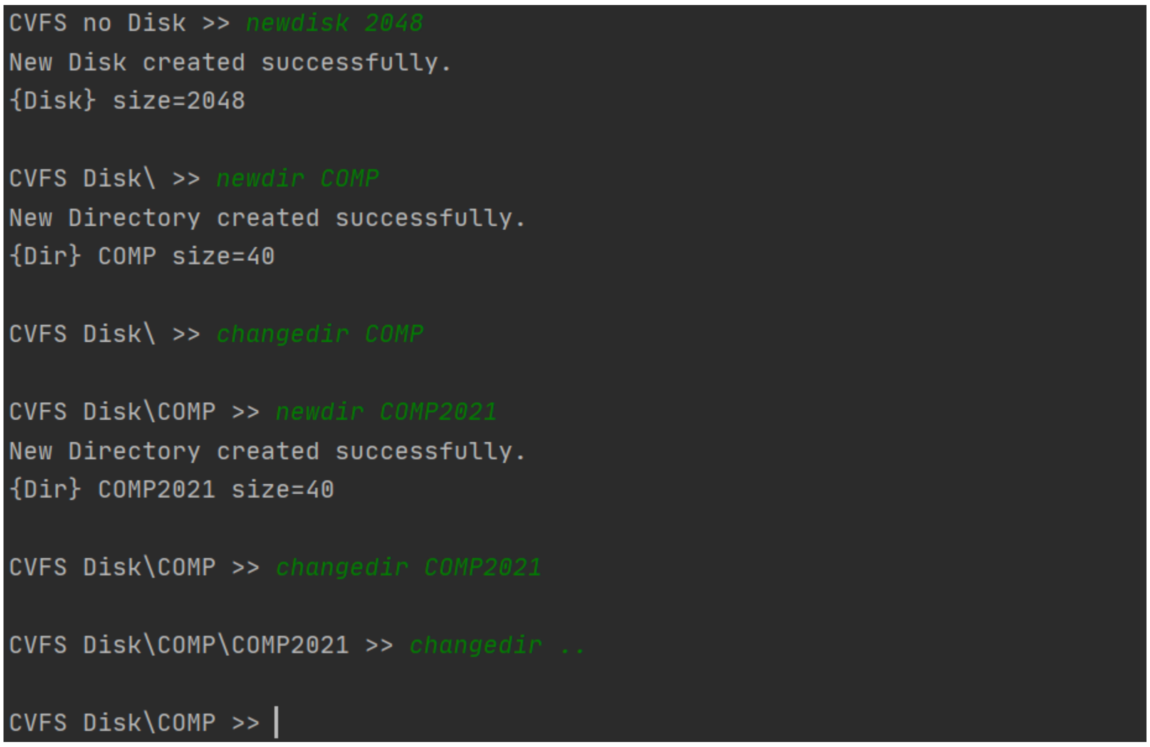
5. **Rename a document.**
   If you want to delete a new document, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, then create a new document. After that, you should input: rename previous_filename changed_filename to rename the document. If you successfully rename

the document, the system will show a hint like: **File xxx has been changed to file yyy**. The screenshot is shown below:



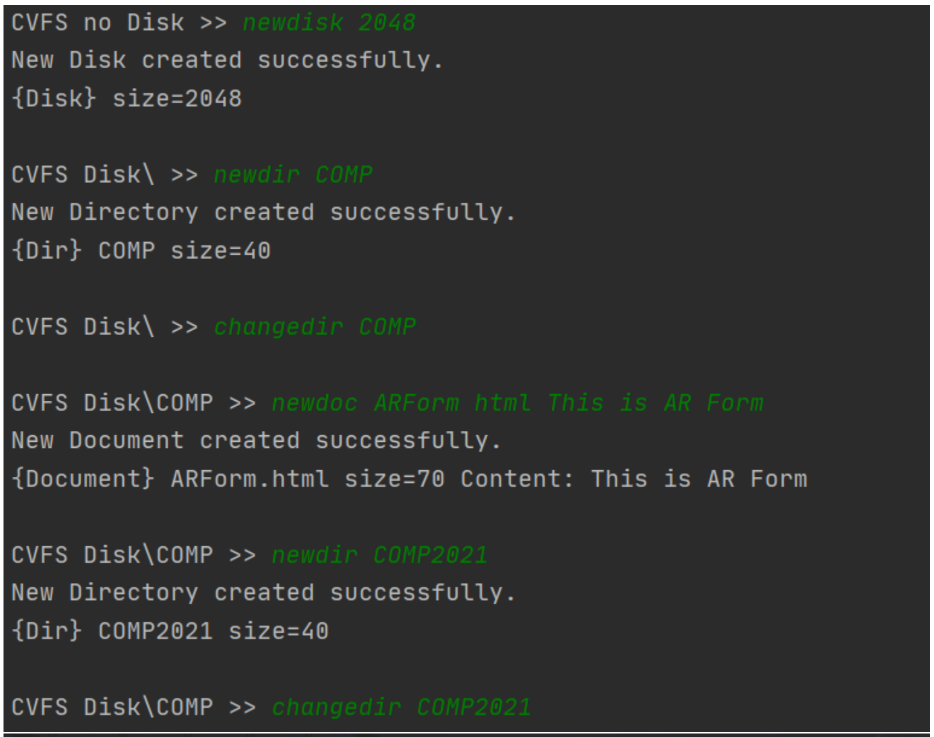
```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdoc WIE css Before graduate, WIE is needed.
New Document created successfully.
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.

CVFS Disk\ >> rename WIE ABC
File WIE has been changed to file ABC
```

6. **Change working directory.**
   If you want to change working directory, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, then create a new directory. After that, if you want to change to this directory to work, just input: **changedir xxx**, then prompt will change to **CVFS Disk**\ **xxx**>>. If you want to change it to its parent directory, just input: **changedir …**, then then prompt will change to its parent directory like: **CVFS Disk**\ >>. The screenshot is shown below:

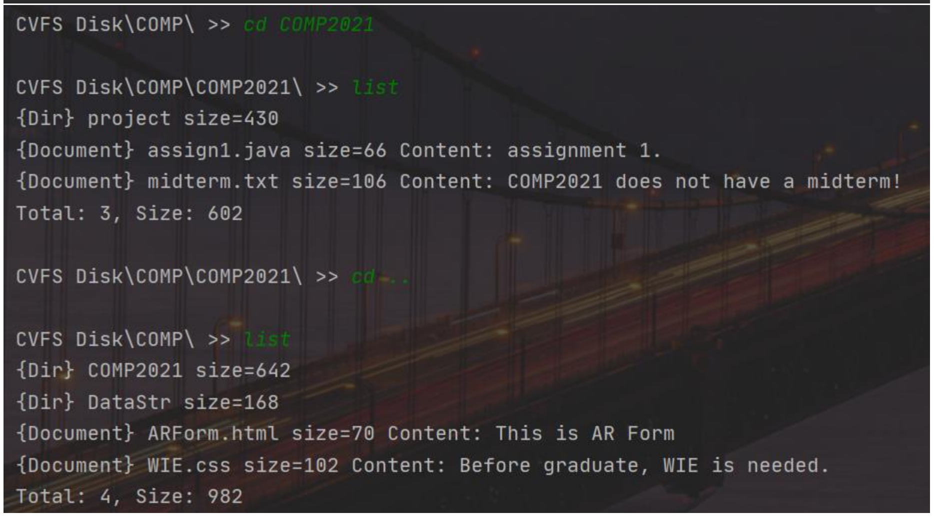


```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP >> changedir COMP2021

CVFS Disk\COMP\COMP2021 >> changedir ..

CVFS Disk\COMP >> |
```

7. **List all files in the working directory.**
   If you want to change working directory, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, then create a new directory and create a new file. After that, if you want to list all files in the working directory, just input: **list**, the system will show the number of documents and all the document like: **{Document} xxx.xx size=xx Content: xxxxx**. The screenshot is shown below:

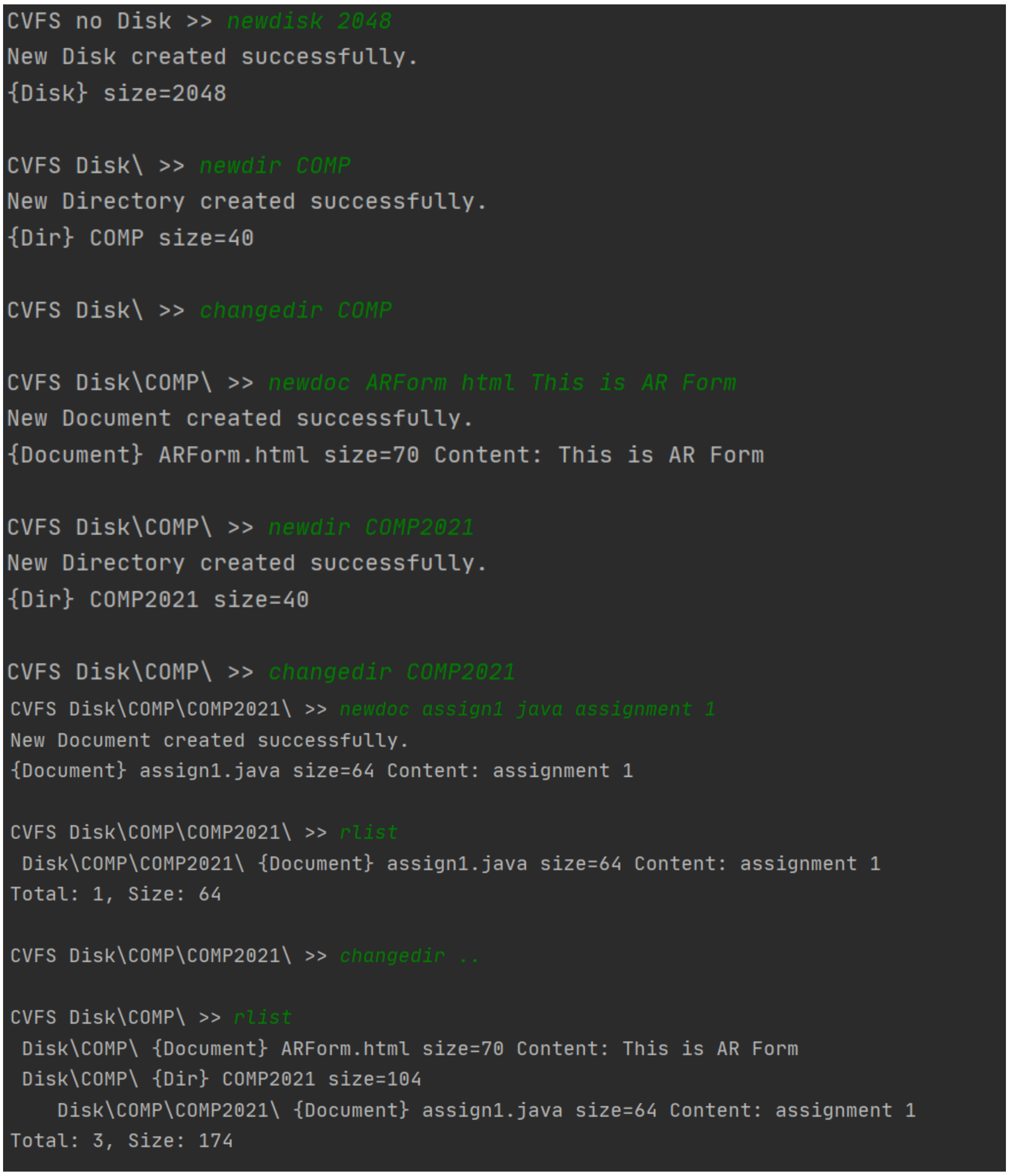


```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP >> newdoc ARForm html This is AR Form
New Document created successfully.
{Document} ARForm.html size=70 Content: This is AR Form

CVFS Disk\COMP >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP >> changedir COMP2021
```



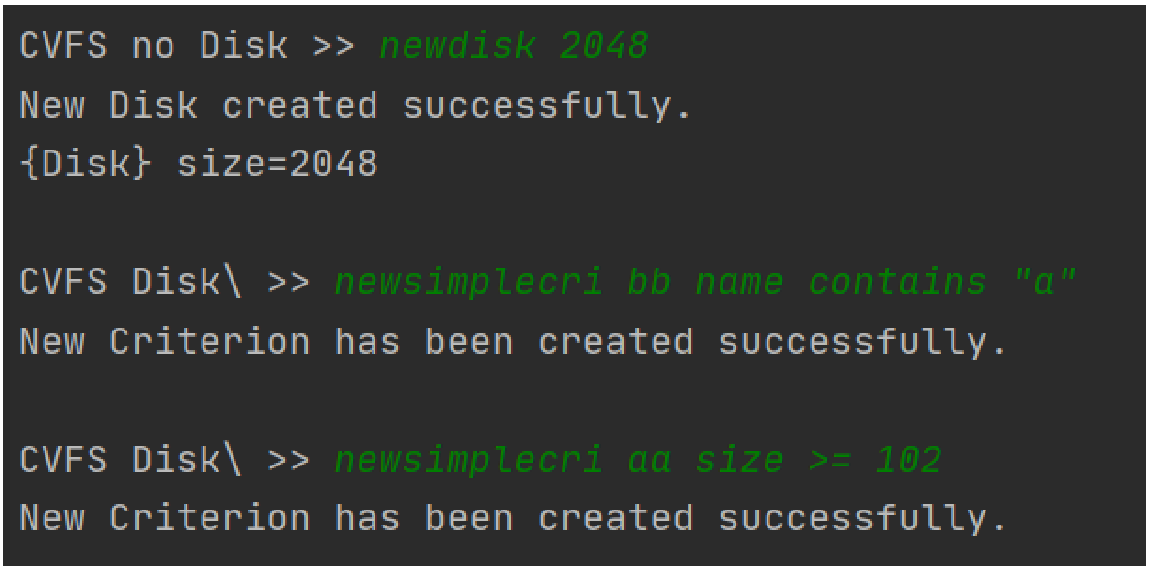
```
CVFS Disk\COMP\ >> cd COMP2021

CVFS Disk\COMP\COMP2021\ >> list
{Dir} project size=430
{Document} assign1.java size=66 Content: assignment 1.
{Document} midterm.txt size=106 Content: COMP2021 does not have a midterm!
Total: 3, Size: 602

CVFS Disk\COMP\COMP2021\ >> cd ..

CVFS Disk\COMP\ >> list
{Dir} COMP2021 size=642
{Dir} DataStr size=168
{Document} ARForm.html size=70 Content: This is AR Form
{Document} WIE.css size=102 Content: Before graduate, WIE is needed.
Total: 4, Size: 982
```

8. **List recursively all files in the working directory.**
   If you want to change working directory, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>, then create a new directory and create a new file. Then create a sub-directory and a sub-file. After that, if you want to list recursively all files in the working directory, just input: **rlist**, the system will show the number of documents and all the document like: **{Document} xxx.xx size=xx Content: xxxxx**. The screenshot is shown below:



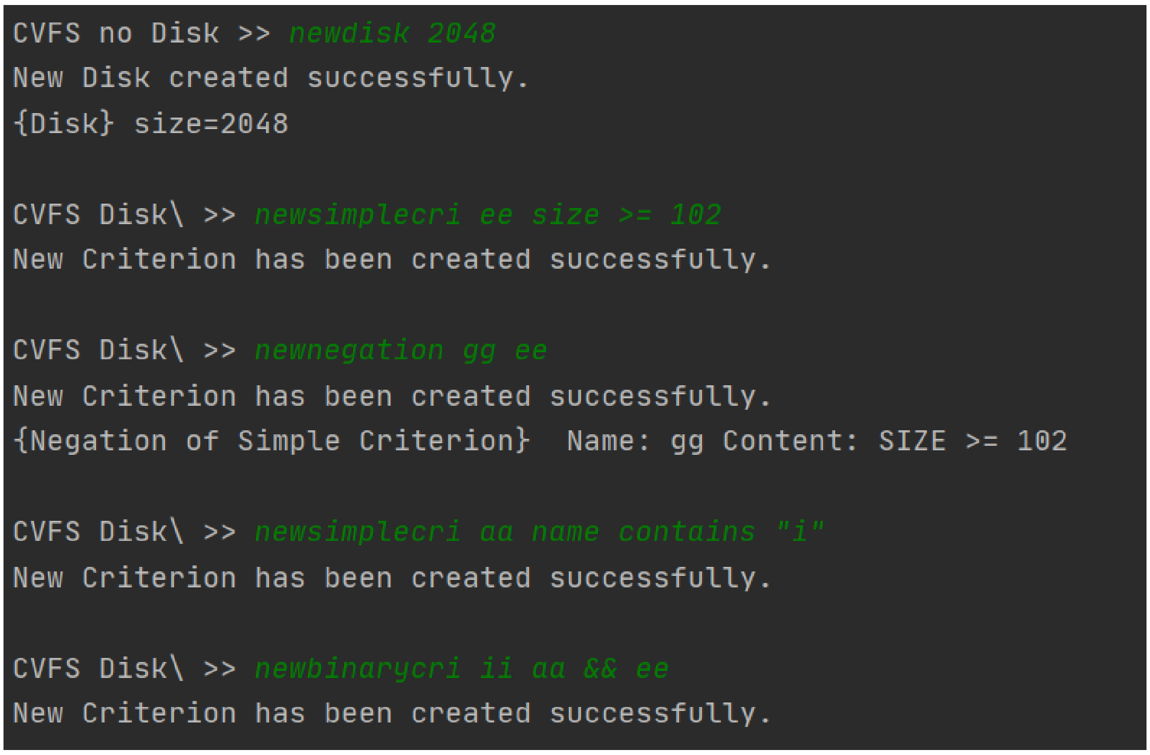
```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newdir COMP
New Directory created successfully.
{Dir} COMP size=40

CVFS Disk\ >> changedir COMP

CVFS Disk\COMP\ >> newdoc ARForm html This is AR Form
New Document created successfully.
{Document} ARForm.html size=70 Content: This is AR Form

CVFS Disk\COMP\ >> newdir COMP2021
New Directory created successfully.
{Dir} COMP2021 size=40

CVFS Disk\COMP\ >> changedir COMP2021
CVFS Disk\COMP\COMP2021\ >> newdoc assign1 java assignment 1
New Document created successfully.
{Document} assign1.java size=64 Content: assignment 1

CVFS Disk\COMP\COMP2021\ >> rlist
 Disk\COMP\COMP2021\ {Document} assign1.java size=64 Content: assignment 1
Total: 1, Size: 64

CVFS Disk\COMP\COMP2021\ >> changedir ..

CVFS Disk\COMP\ >> rlist
 Disk\COMP\ {Document} ARForm.html size=70 Content: This is AR Form
 Disk\COMP\ {Dir} COMP2021 size=104
    Disk\COMP\COMP2021\ {Document} assign1.java size=64 Content: assignment 1
Total: 3, Size: 174
```

9. **Create a new criteria.**
   If you want to create a new criteria, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk\** >>. After that, if you want to create a new criteria, just input: **newsimplecri criname attrname op val**, the system will show a hint like: **New Criterion created successfully**. The screenshot is shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newsimplecri bb name contains "a"
New Criterion has been created successfully.

CVFS Disk\ >> newsimplecri aa size >= 102
New Criterion has been created successfully.
```

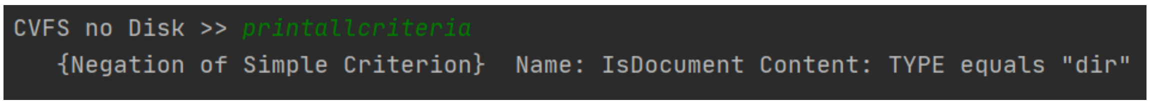
10. **Create a new composite criteria.**
    If you want to create a new composite criteria, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>. Then create a new criteria. After that, if you want to create a new composite criteria, just input: **newnegation criname1 criname2** or **newnegation criname1 criname3 logicOp criname4**, the system will show a hint like: **New Criteria created successfully**. And show your document and its name and content like: **{Negation of Simple Criterion} Name: xx Content: xxx**. The screenshot is shown below:



```
CVFS no Disk >> newdisk 2048
New Disk created successfully.
{Disk} size=2048

CVFS Disk\ >> newsimplecri ee size >= 102
New Criterion has been created successfully.

CVFS Disk\ >> newnegation gg ee
New Criterion has been created successfully.
{Negation of Simple Criterion}  Name: gg Content: SIZE >= 102

CVFS Disk\ >> newsimplecri aa name contains "i"
New Criterion has been created successfully.

CVFS Disk\ >> newbinarycri ii aa && ee
New Criterion has been created successfully.
```

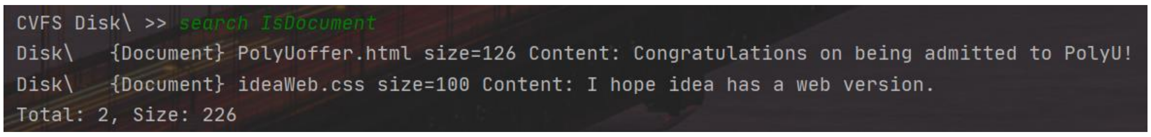
11. **Print all the criteria.**
    If you want to print all the criteria, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>. Then create a new criteria. After that, if you want to print all the criteria, just input: **printallcriteria**, the system will show a hint like: **New Criteria created successfully**. And show your document and its name and content like: **{Negation of Simple**

Criterion} Name: xx Content: xxx. The screenshot is shown below:



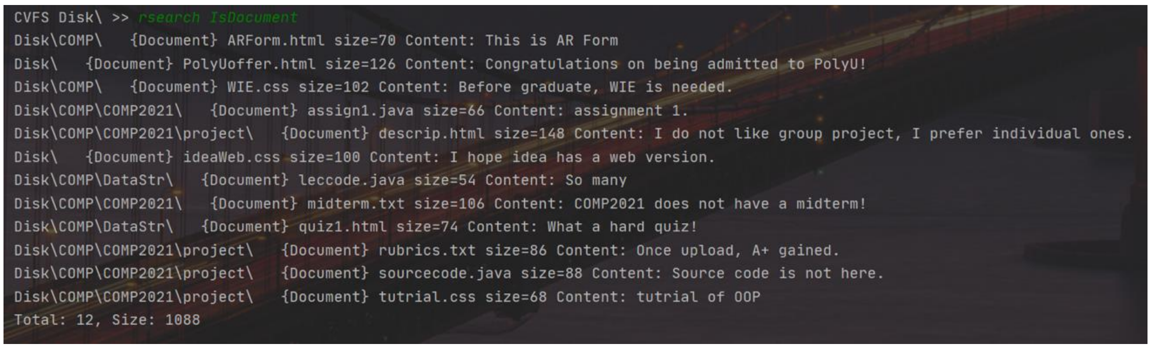
```
CVFS no Disk >> printallcriteria
   {Negation of Simple Criterion}  Name: IsDocument Content: TYPE equals "dir"
```

12. **Search certain document.**
    If you want to search certain document, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>. Then create a new file. After that, if you want to search certain document, just input: **search IsDocument**, the system will show your document and its name and content like: **{Document} Name: xx Content: xxx**. The screenshot is shown below:



```
CVFS Disk\ >> search IsDocument
Disk\   {Document} PolyUoffer.html size=126 Content: Congratulations on being admitted to PolyU!
Disk\   {Document} ideaWeb.css size=100 Content: I hope idea has a web version.
Total: 2, Size: 226
```

13. **Rearch certain document recursively.**
    If you want to search certain document recursively, just create a disk at first (or the disk has existed). Then the prompt will change to **CVFS Disk**\ >>. Then create a new file. After that, if you want to search certain document recursively, just input: **rsearch IsDocument**, the system will show all document you search recursively and its name and content like: **{Document} Name: xx Content: xxx**. The screenshot is shown below:



```
CVFS Disk\ >> rsearch IsDocument
Disk\COMP\   {Document} ARForm.html size=70 Content: This is AR Form
Disk\   {Document} PolyUoffer.html size=126 Content: Congratulations on being admitted to PolyU!
Disk\COMP\   {Document} WIE.css size=102 Content: Before graduate, WIE is needed.
Disk\COMP\COMP2021\   {Document} assign1.java size=66 Content: assignment 1.
Disk\COMP\COMP2021\project\   {Document} descrip.html size=148 Content: I do not like group project, I prefer individual ones.
Disk\   {Document} ideaWeb.css size=100 Content: I hope idea has a web version.
Disk\COMP\DataStr\   {Document} leccode.java size=54 Content: So many
Disk\COMP\COMP2021\   {Document} midterm.txt size=106 Content: COMP2021 does not have a midterm!
Disk\COMP\DataStr\   {Document} quiz1.html size=74 Content: What a hard quiz!
Disk\COMP\COMP2021\project\   {Document} rubrics.txt size=86 Content: Once upload, A+ gained.
Disk\COMP\COMP2021\project\   {Document} sourcecode.java size=88 Content: Source code is not here.
Disk\COMP\COMP2021\project\   {Document} tutrial.css size=68 Content: tutrial of OOP
Total: 12, Size: 1088
```