Project Report

Group 18
COMP2021 Object-Oriented Programming (Fall 2020)
LAM Chi Yuen (19052423D)
LAM Kwan Lok (19051723D)
WONG Tsz Fung (19059855D)

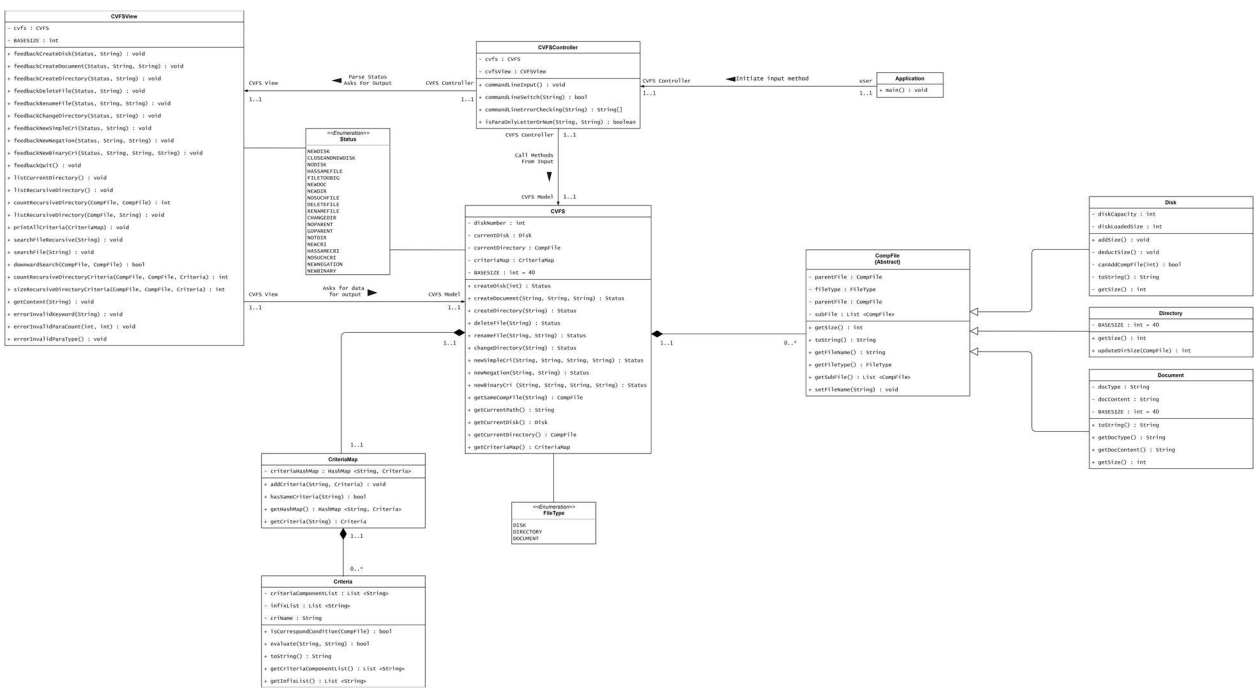
1. Introduction

This report describes the design and implementation of the Comp Virtual File System (CVFS) by group 18. This project is part of the course COMP2021 Object-Oriented Programming at PolyU. The report will be divided into three parts – design, requirements and user manual.

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 - Class Diagram



CompFile

`CompFile` is an abstract class that represents a basic file in the CVFS. Document, directory or even disk all inherits from `CompFile`, each with their specific fields and methods.

CVFS Model

The model is built around `CompFile` and is dedicated to logic error checking and `CompFile` manipulation. Each method (of file manipulation) contains a series of logic error (e.g. searching with a criterion that is not created) checking condition and return the corresponding status to the controller. Should a command be valid, each method will carry out operation on the files in the current disk.

Criteria

`Criteria` is a class that represents a simple or composite criterion. It contains an `ArrayList` that stores attribute names, logic operators and specific values in reverse Polish notation. It also contains methods to evaluate whether a `CompFile` satisfies the criterion itself.

`CriteriaMap` is a class that stores each `Criteria` object created in the model. It pairs the criteria name with the `Criteria` object.

CVFS Controller

The controller serves as a bridge between the model and the view. It receives user input and does input error checking (i.e. too few arguments or wrong keywords). It then passes the input to the model. The controller passes the status to the view for output to console. The controller itself is not visible to the model and view. It is only responsible for passing the commands.

CVFS View

For the view, it is responsible for the output to console. It either receives a status, or a method call from the controller. For the former, the feedback method prints out statements according to the status. For the latter, these calls are exclusively for listing and searching methods. These listing and searching methods ask for data from the model and print out the result. Originally, such is not allowed in MVC pattern (view and model cannot communicate with each other). However, to simplify the operation (of passing the content from model to controller, and controller to view), it is decided that the view should be able to retrieve data from the model directly.

Enumeration

To facilitate understanding and coding efficiency, enumerations are used for naming file types and status. `FileType` indicate the type of files, such as documents and directories. `Status` refers to the condition of file operation, whether it succeeds or fails due to input or logic errors.

Demonstration

To better demonstrate the design, we can take the command `newSimpleCri aa size > 40` and `rSearch aa` as an example.

When the user enters the former, the command will be checked be the controller and then passes to the model. The model then creates new `Criteria` accordingly and stores the `Criteria` in `CriteriaMap` and returns a status of `NEWCRI` to the view, which means a simple criterion is created successfully. The view prints a message according to `NEWCRI`.

The latter command will again be checked in the controller. The controller then calls the method in the view. In the end, by using the data in the model, the result of a recursive search done according to `Criteria` aa will be printed.

2.2 Requirements

All the requirements, bar the bonus features, are implemented. The following is the implementation and error handling details. In implementation details, it is assumed that the input is valid and free from input and logic error.

[REQ1] 1) The requirement is implemented.

2) Implementation Details
- `createDisk` creates an object of class `Disk` with the specified size.
  - In the constructor of `Disk`, since it inherits from `CompFile`, it uses the `super` constructor with the given parameters.
  - Fields exclusive to `Disk`, such as `diskCapacity` and `diskLoadedSize` are assigned in the constructor of Disk.
- Should there be no disk being operated, set the current disk and directory to the newly created disk. The method then returns a status `NEWDISK` to the view.
- Otherwise, replace the disk that is being operated currently. Set the current directory to the newly created disk. The method then returns a status `CLOSEANDNEWDISK` to the view.
- The method `feedbackCreateDisk` in the view then prints the disk creation message according to the status.

3) Error Conditions and Handling

Input Error
- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array, it must be `newDisk`.
- If the keyword is correct, it checks for the length of the array, which has to be 2 (1 keyword + 1 parameter).
- If the number of parameters is correct, it checks for the type (composition) of the parameter. Since creating a disk requires a size, the parameter must be a numeric value smaller than the maximum size of an integer in Java.
- Whether a string contains only integer, English letters or both is checked by a method called `isParaLetterOrNum`.
- Only if all the above is correct, the controller will call and pass the disk size to the method `createDisk` in the model.
- Otherwise, if any input error is found, the methods in view (shown below) will print error message accordingly.

```
errorInvalidKeyword
errorInvalidParaCount
errorInvalidParaType
```

[REQ2] 1) The requirement is implemented.

2) Implementation Details
- Method `createDoc` creates an object of class `Document` in the current directory, with a given name, document type and content.
  - In the constructor of `Document`, since it inherits from `CompFile`, it uses the `super` constructor.

    - Fields exclusive to `Document` like document type and content are assigned in the constructor of `Document`.
- The newly created document is add added to the current directory by adding it to the `ArrayList` of the current directory.
- The total size of the document, that is 40 + content length * 2 is calculated. It is then added to the size of the current directory and disk.
- In the end, `createDoc` returns a status `NEWDOC` to view and the method `feedbackCreateDocument` prints the document creation message.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `newDoc`.
- If the keyword is correct, it checks for the length of array, which must be 4 (1 keyword + 3 parameters)
- If the number of parameters is correct, it checks for the type of the parameter. For the second element (document name), it must consist of only digits and English letters. For the third element (document type), it must be either `css`, `html`, `java` or `txt` (all in lowercase). For the content, it can be anything (even empty).
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, the method `createDoc` will check for logic error.
- If there is no disk being operated, it returns a status `NODISK` to the view and the document creation is aborted.
- If there is a file with same name in the current directory, it returns a status `HASSAMEFILE` to the view, and the document creation is aborted.
- If the current disk cannot hold the current document, it returns a status `FILETOOBIG` to the view, and the document creation is aborted.
- The method `feedbackCreateDocument` in the view then prints the error message according to the status.

[REQ3] 1) The requirement is implemented.

2) Implementation Details

- Method `createDir` creates an object of class `Directory` in the current directory, with a given name.
    - In the constructor of `Directory`, since it inherits from `CompFile`, it uses the `super` constructor.
- The newly created directory added to the current directory by adding the document to the `ArrayList` of the current directory.
- The total size of the directory, that is 40, is then added to the size of the current directory and disk.
- The size of a directory is calculated dynamically. Whenever a call to `getSize` of `Directory` is made, it will evaluate the size of the directory by counting the files inside.

- In the end, `createDir` returns a status `NEWDIR` to the view and the view prints the corresponding directory creation message.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `newDir`.
- If the keyword is correct, it checks for the length of array, which has to be 2 (1 keyword + 1 parameters)
- If the number of parameters is correct, it checks for the type of the parameter. For the second element (the directory name) of the array, it must consist of only digits and English letters.
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `createDir` will check for logic error.
- If there is no disk being operated, it returns a status `NODISK` to the view and the directory creation is aborted.
- If there is a file with same name in the current directory, it returns a status `HASSAMEFILE` to the view and directory creation is aborted.
- If the disk cannot hold the directory, it returns a status `FILETOOBIG` to the view and directory creation is aborted.
- The method `feedbackCreateDirectory` in the view then prints the error message according to the status.

[REQ4] 1) The requirement is implemented.

2) Implementation Details

- Method `deleteFile` deletes the reference to the file in the current directory, given a file name.
- This is done by `removeIf` and a lambda function.
- The size of loaded file in the disk is then deducted by the size of the removed file.
- In the end, `deleteFile` then returns a status `DELETEFILE` to the view and the view prints the corresponding directory creation message.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `delete`.
- If the keyword is correct, it checks for the length of array, which must be 2 (1 keyword + 1 parameters)
- If the number of parameters is correct, it checks for the type of the parameter. For the second element (the name of the file to be deleted) of the array, it must consist of only digits and English letters.
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `deleteFile` checks for logic error.
- If there is no disk being operated, it returns a status `NODISK` to the view, and the deletion is aborted.
- If there is no file with the name same as the given one, it returns a status `NOSUCHFILE` to the view, and the deletion is aborted.
- Method `feedbackDeleteFile` in the view then prints error message according to the status.

[REQ5] 1) The requirement is implemented.

2) Implementation Details
   - Method `renameFile` first finds the target file to be renamed via method `getSameCompFile`.
   - Then it uses the setter `setFileName` to set the file name.
   - In the end, `renameFile` return a status `RENAMEFILE` to the view and the view prints corresponding message.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `rename`.
- If the keyword is correct, it checks for the length of array, which has to be 3 (1 keyword + 2 parameters)
- If the number of parameters is correct, it checks for the type of the parameter. For the second and the third element (the name of the file to be renamed and the new name) of the array, it must consist of only digits and English letters.
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `renameFile` checks for logic errors
- If there is no disk being operated, it returns a status `NODISK` and the renaming is aborted.
- If there is no file with name same as the "old name", it returns `NOSUCHFILE` and the renaming is aborted.
- If there is a file with name same as the "new name", it returns `HASSAMEFILE` and the renaming is aborted.
- Method `feedbackRenameFile` in the view then prints error message according to the status.

[REQ6] 1) The requirement is implemented.

2) Implementation Details
   - Method `changeDirectory` checks for the target directory. If it is `..`, it will change the current directory to the parent directory of the original current directory, using `parentFile` field in `CompFile`.
   - If this is the case, `changeDirectory` returns `GOPARENT` to the view and the view prints corresponding message.
   - Otherwise, if it changes to a specific directory, it changes the current directory to the target directory.

- Then, `changeDirectory` returns `CHANGEDIR` to the view and the view prints corresponding message.

3) Error Condition and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `changeDir`.
- If the keyword is correct, it checks for the length of array, which has to be 2 (1 keyword + 1 parameters)
- If the number of parameters is correct, it checks for the type of the parameter. For the second (directory name) of the array, it must consist of only digits and English letters, or be equal to `..`.
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `changeDirectory` checks for logic errors
- If there is no disk being operated, it returns a status `NODISK` to the view and the directory change is aborted.
- If the target directory is not found, it returns a status `NOSUCHFILE` to the view and the directory change is aborted.
- If the target directory is a document, it returns a status `NOTDIR` to the view and the directory change is aborted.
- If the target directory is `..` but there is no parent directory, it returns `NOPARENT`, and the directory change is aborted.
- Method `feedbackChangeDirectory` prints the message according to the status

[REQ7] 1) The requirement is implemented.

2) Implementation Details

- Method `listCurrentDirectory` in the view iterates through and show every file found (directly) in the current directory.
- During the iteration, the number and total size of file is also calculated.
- In the end, it will print the number and total size of file.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameters.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `list`.
- If the keyword is correct, it checks for the length of array, which must be 1 (1 keyword + 0 parameter)
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `listCurrentDirectory` checks for logic error.
- If there is no disk being operated, it prints a message of it.

[REQ8] 2) This requirement is implemented.

3) Implementation Details
- Method `listRecursiveDirectory` lists every file in the current directory by recursion.
- It first iterates through each file in the current directory.
- If it finds a document, the document is printed.
- If it finds a directory, the directory is printed. It will also enter that directory and iterate through every file in that directory, and print document and directory in the same way.
- The total size of the file is calculated by iterating through the current directory and adding the size from method `getSize`.
- The total number of the file is calculated separately in a method called `countRecursiveDirectory`.
- In the end, the total size and number of files are printed.

4) Error Conditions and Handling

Input Error
- The controller checks for three types of input error: keyword, number of parameters.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array, it must be `rList`.
- If the keyword is correct, it checks for the length of array, which must be 1 (1 keyword + 0 parameter)
- If any input error is found, the methods in view will print error message accordingly.

Logic Error
- If all the above is correct, method `listRecursiveDirectory` checks for logic error.
- If there is no disk being operated, it prints a message of it.

[REQ9] 1) This requirement is implemented.

2) Implementation Details
- Method `newSimpleCri` creates an object of class `Criteria`.
  - There are overloading of constructors in the class `Criteria`. In `newSimpleCri`, the one with 4 parameters is used.
- In such constructor, the criterion is stored in two `ArrayList`, each with one specific way. The first being in reverse Polish notation, which can be evaluated easily without the use of brackets.
- The second being in infix notation with brackets, which is for showing the criterion to the user.
- For example, in reverse Polish notation `ArrayList`, the content will be: `type, “css”, equals`. Yet for infix notation `ArrayList`, the content will be `(type, equals, “css”)`
- The object of class `Criteria` is then added to the `CriteriaMap`, which is essentially a HashMap. The key is the criterion name and the value is the `Criteria` object.
- In the end, `newSimpleCri` returns a status `NEWCRI` to the view and the view prints the criterion creation message accordingly.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `newSimpleCri`.
- If the keyword is correct, it checks for the length of array, which has to be 5 (1 keyword + 4 parameters)
- If the number of parameters is correct, it checks for the type of the parameter.
  For the second element (criterion name) of the array, it must consist of only English letters, with length being exactly 2.
  For the third element (attribute name) of the array, it must be either `name`, `size` or `type`.
  If attribute name is name, the fourth element (logic operator) of the array must be `contains` and the fifth element must be a string in double quotation mark.
  If the attribute name is size, the fourth element must be one of the equality signs (`!`, `!=`) or one of the inequality signs (`>`, `<`, `>=`, `<=`) and the fifth element must be a non-negative numeric value (checked by having only digits).
  If the attribute name is equals, the fourth element of the array must be `equals` and the fifth element should a document type in double quotation mark (e.g. "`css`", "`html`").
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, `newSimpleCri` checks for logic error.
- If there is already a criterion with the same name, it returns `HASSAMECRI` to the view and the criterion creation is aborted.
- In the end, `feedbackNewSimpleCri` will print the error message according to the status.

[REQ10] 1) This requirement is implemented.

2) Implementation Details
   - In the constructor of the CVFS model, an object of class `Criteria` is created by using a special constructor with no parameter given. Such constructor is only used for creating the `IsDocument` criterion.
   - To facilitate comparison, `IsDocument` is defined as `FileType equals Document`.
   - Such criterion is stored into `CriteriaMap`, with key being `IsDocument` and value being a `Criteria` object.

3) Error Conditions and Handling
   - Since the storage of criterion `IsDocument` is done internally without requiring any user input, there is no need for error checking.

[REQ11] 1) This requirement is implemented.

2) Implementation Details
   - Method `newNegation` creates a `Criteria` object that is the negated version of an existing one.
   - Such method uses another constructor of `Criteria` with 2 parameters to generate the negated version.
   - In such constructor, the reverse Polish notation `ArrayList` of the new Criteria object has the same content, except a negation sign `!` is added in the end.

- For the infix notation `ArrayList`, the negation sign is added in the front and the content is bracketed in the end.
- For example, for `size > 40`, the negated version in reverse Polish notation `ArrayList` will be `size, 40, >, !`.
- Yet for the infix notation `ArrayList` will be `(, !, (, size,  >,  40, ), )`.
- In the end, method `newNegation` returns a status `NEWNEGATION` to the view and the view prints the message of negating criterion according to the status.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `newNegation`.
- For the second element (negated criterion name) of the array, it must be two English letters.
- For the third element (existing criterion name) of the array, it must be either two English letters or `IsDocument`.
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, `newNegation` checks for logic errors.
- If the negated criterion has a name that is used, it returns a status `HASSAMECRI` to the view and the negation is aborted.
- If the "existing" criterion is not found, it returns a status `NOSUCHCRI` to the view and the negation is aborted.
- In the end, `feedbackNewNegation` will print the error message according to the status.

[REQ11] 1) This requirement is implemented.

2) Implementation Details

- Method `newBinaryCri` creates a `Criteria` object that is the combination of two existing `Criteria` objects.
- Such method uses a constructor in `Criteria` with 4 parameters.
- In such constructor, the reverse Polish notation `ArrayList` will contain the content of the first and second `Criteria`, and the logic operator is added in the end.
- For the infix notation `ArrayList`, the logic operator is placed between the content of the first and second Criteria.
- For example, for two criteria being `size > 169` and `name contains "oldlam"`, and the logic operator being &&, the reverse Polish notation `ArrayList` will be
  `size 169 > name "oldlam" contains &&`
- Yet for infix notation `ArrayList`, it will be
  `((size > 169) && (name contains "oldlam"))`.
- In the end, `newBinaryCri` returns a status `NEWBINARYCRI` to the view and the view prints the corresponding message.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameter and types of parameter.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `newBinaryCri`.
- For the second element (criterion that going to be created), it must be two English letters.
- For the third and the fifth element (criterion name) of the array, it must be two English letters or `IsDocument`.
- For the third element of the array, it must be a logic operator, which is either && or ||

Logic Error

- If all the above is correct, `newBinaryCri` checks for logic errors.
- If either of the criteria in the command does not exist, it returns a status `NOSUCHCRI`
- If the newly created binary criterion has name that same as the existing criteria, it returns a status `HASSAMECRI`.
- In the end, `feedbackNewBinaryCri` prints the error message according to the status

[REQ12] 1) This requirement is implemented.

2) Implementation Details
   - Method `printAllCriteria` in the view iterates through the key set of the `HashMap` in `CriteriaMap`, and prints the key and values each time.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameters.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `printAllCriteria`.
- If the keyword is correct, it checks for the length of array, which must be 1 (1 keyword + 0 parameter)
- If any input error is found, the methods in view will print error message accordingly.

[REQ13] 1) This requirement is implemented.

2) Implementation Details
   - Method `searchFile` in the view searches the files that satisfy the given criterion.
   - It simply iterates through the files in the current directory and prints the files which satisfy the criterion.
   - Total size and number of files are calculated in the same manner with method list, except with criterion used.
   - The `Criteria` evaluation is done based on the reverse Polish notation `ArrayList`.
   - For each element in that `ArrayList`, push that element into a stack.
   - When the element is an operator, pop the stack for three times and evaluate the result by using `evaluate`. Push the evaluation result into the stack.
   - When the iteration is completed, pop the final element and return as `boolean`

- For example, the `ArrayList` contains {`size`, `30`, >}
- Push `size`, `30`, > into the stack
- Pop the stack for three times, and suppose the evaluation is `true`, push `true`.
- Since the iteration is done, return `true`
- There might be situation that there are only two items in the stack. If that is the case, pop twice only.
- For example, for the negation of a criterion, the stack might only have two items.
- A special condition and method is made subsequently to pop the stack for two times and evaluate that two items.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameters.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array , it must be `search`.
- If the keyword is correct, it checks for the length of array, which must be 2 (1 keyword + 1 parameter)
- If any input error is found, the methods in view will print error message accordingly.

Logic Error

- If all the above is correct, method `search` checks for logic error.
- If there is no disk being operated, it prints the condition of it.

[REQ14] 1) This requirement is implemented.

2) Implementation Details

- Method `searchFileRecursive` in the view searches a file that satisfy the criterion in the current directory recursively.
- For printing the files that satisfy the criterion, it is done in the same way as `rList`, except we need to handle situation when the parent file does not satisfy the criterion, but the child file does satisfy the criterion (both of them need to be printed). A method `downwardSearch` is used to check this situation
- For counting the number of files that satisfy the criterion, method `countRecursiveDirectoryCriteria` also counts the number in the same manner with method `countRecursiveDirectory`, except the former checks if the file satisfies the criterion.
- Yet for counting the size of the files, for a directory, it needs to consider whether the sub-directories or files are included, to prevent repeated counting of files. As a result, a `Boolean` variable `topper` is passed each time to indicate whether the parent folder has already been counted.
- In the end, the total size and number of files are shown.

3) Error Conditions and Handling

Input Error

- The controller checks for three types of input error: keyword, number of parameters.
- The command (in String) is split into an array, with delimiter being space.
- For the first element (keyword) of the array, it must be `rSearch`.

- If the keyword is correct, it checks for the length of array, which must be 2 (1 keyword + 1 parameter)
- If any input error is found, the methods in view will print error message accordingly.

- **Logic Error**
- If all the above is correct, method r`Search` checks for logic error.
- If there is no disk being operated, it prints the condition of it.

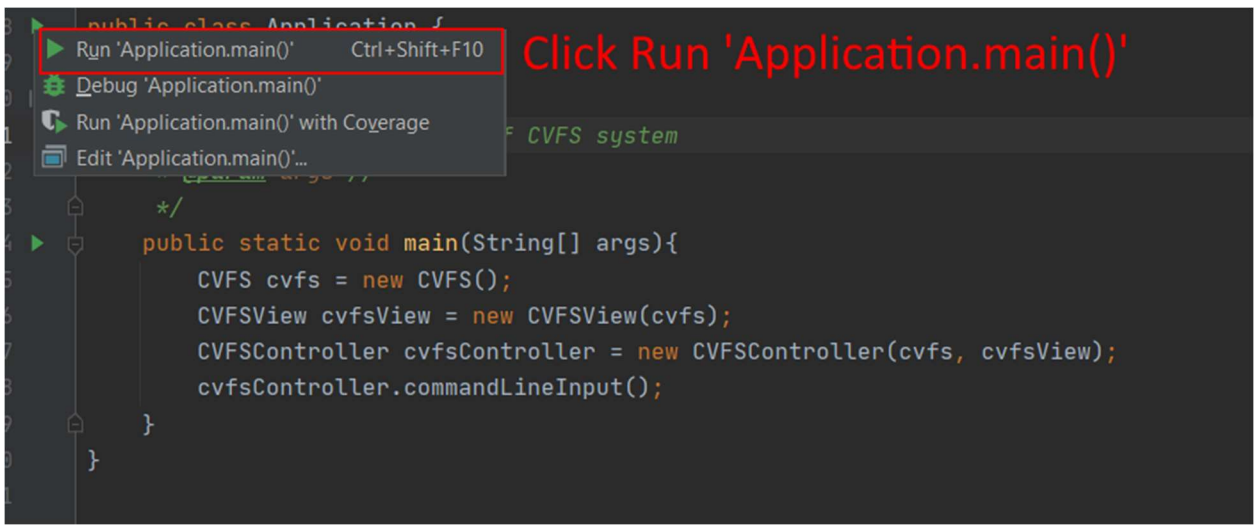
3. User Manual

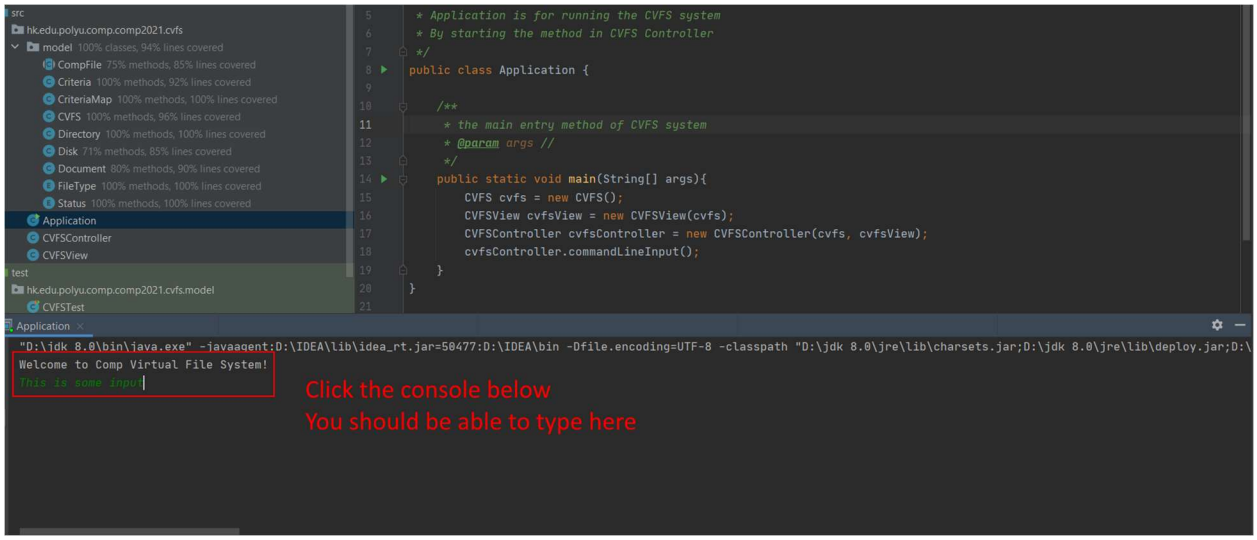
In this section, we explain how the CVFS works from a user's perspective.

Starting the CVFS

Run Application.class after compiling every .java file in Command Prompt, or run Application.java in any IDE directly. The CVFS then can receive input.

For example, in Intelij IDEA, press the green arrow button after opening Application.java.





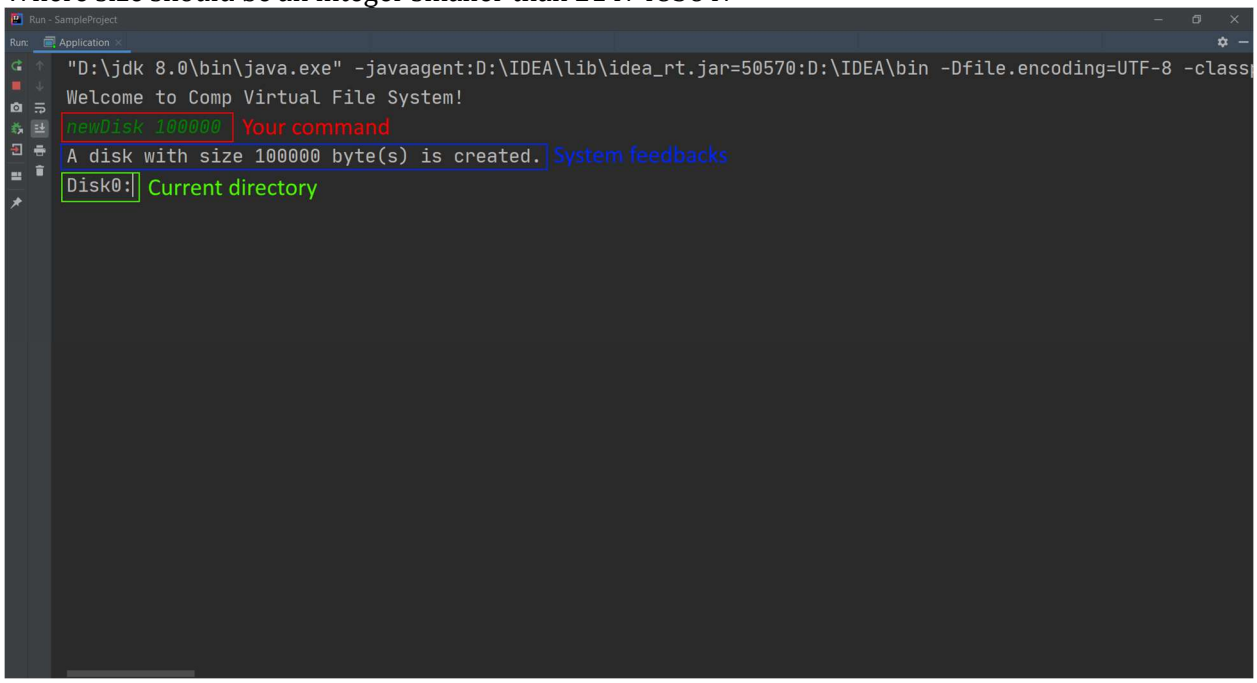
Creating a Disk

Most of the commands require a disk to work.

You can create a disk by typing the following command:

```
newDisk size
```

Where size should be an integer smaller than 2147483647

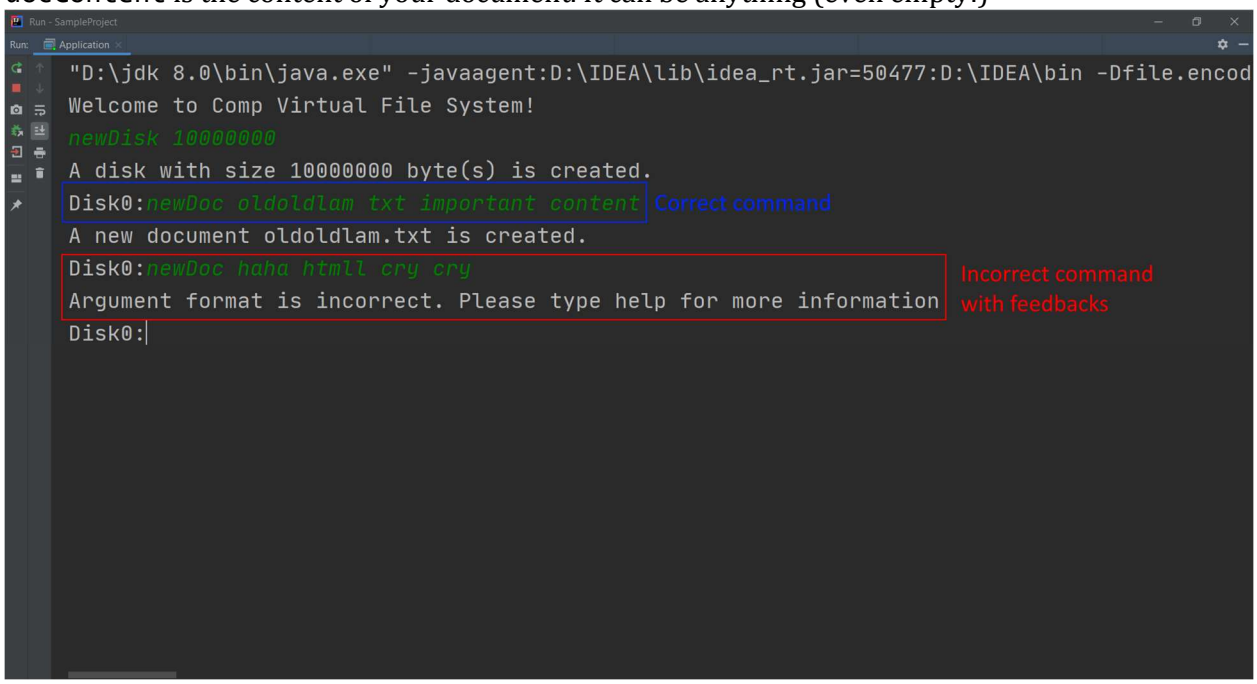


Creating and Accessing a Document

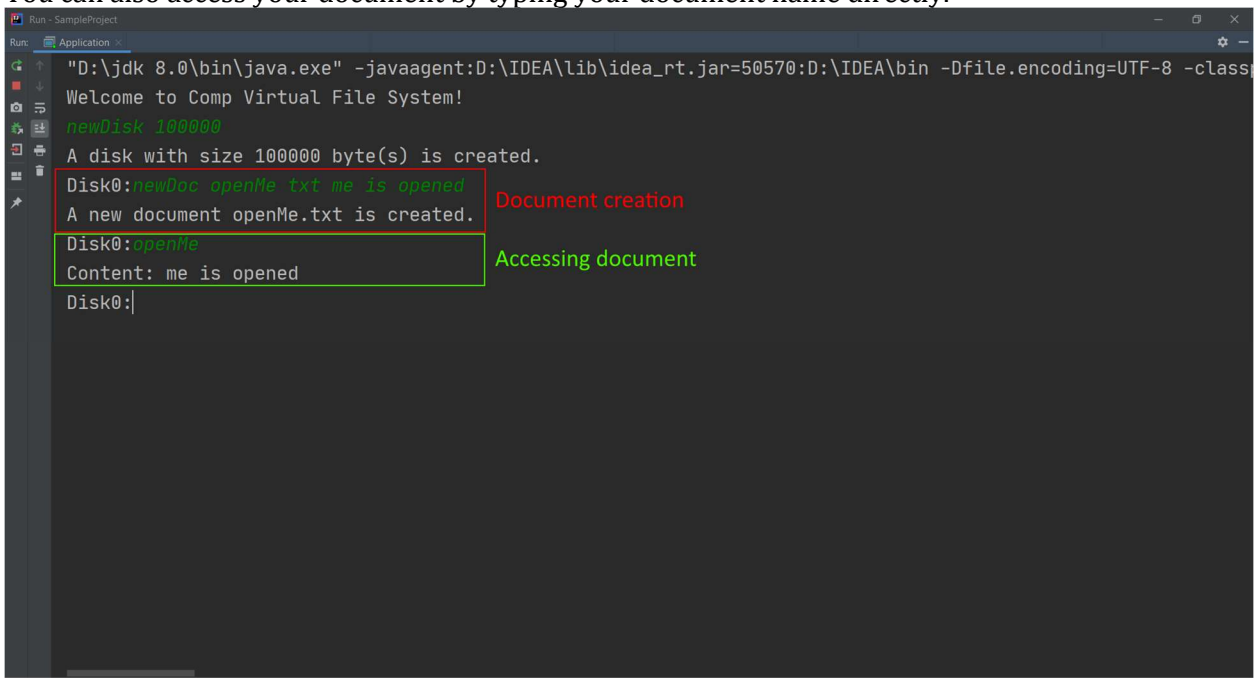
With a disk operating in the system, you can now create a document by typing the following command:

```
newDoc docName docType docContent
```

Where `docName` is your document name. It can contain 10 or less digits or English letters
`docType` is your document type, which can be css, java, html or txt.
`docContent` is the content of your document. It can be anything (even empty!)



You can also access your document by typing your document name directly.



Creating and Accessing a Directory

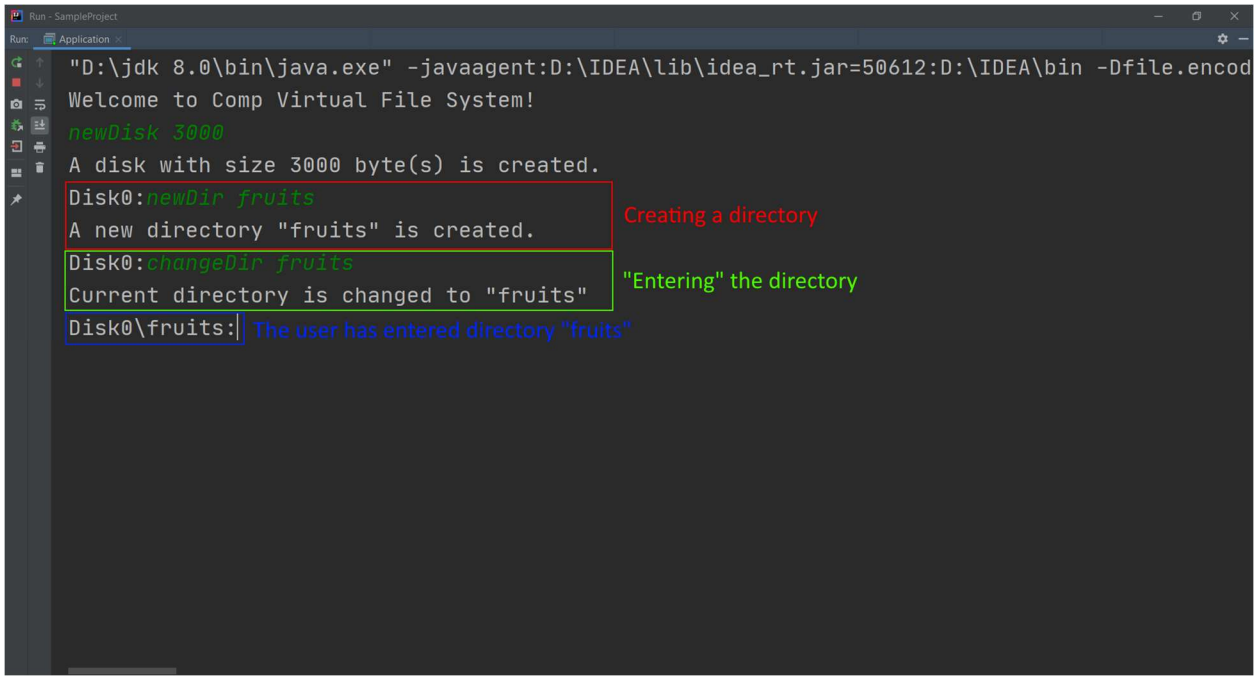
Too many documents can be a hassle to manage. You can try to create directories and place files inside for a much clearer view.

```
newDir dirName
```

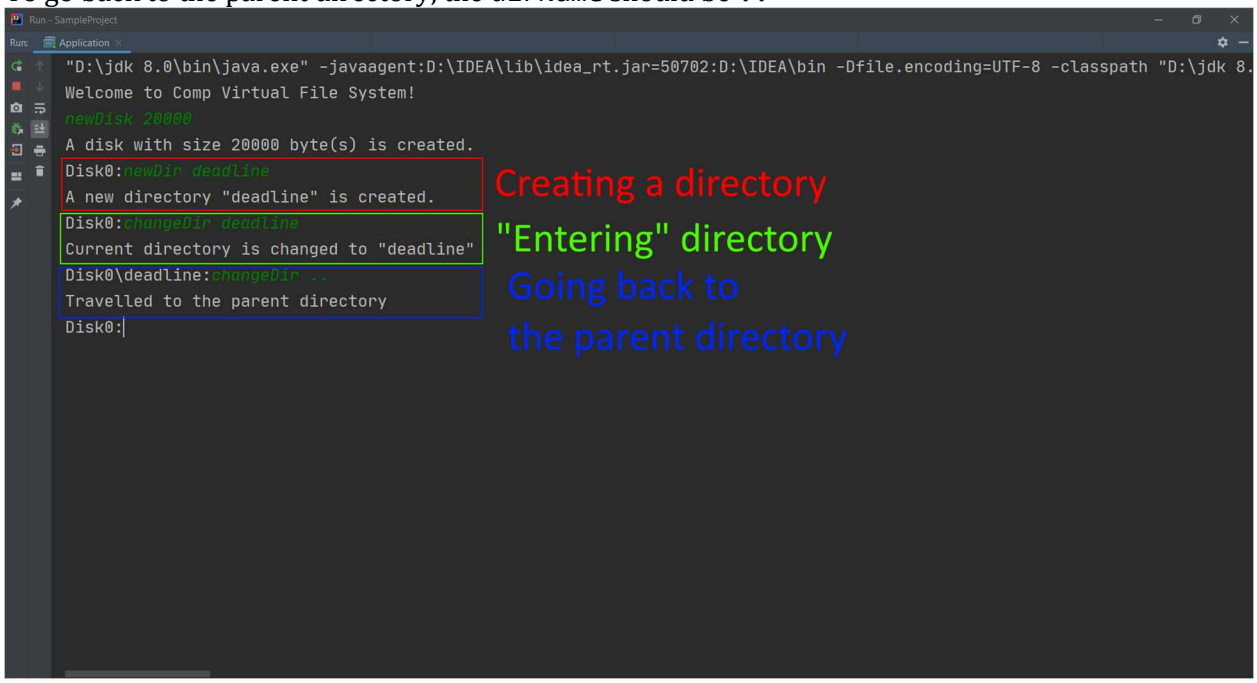
Where `dirName` is the directory name

You can also change the directory by the following command:

```
changeDir dirName
```

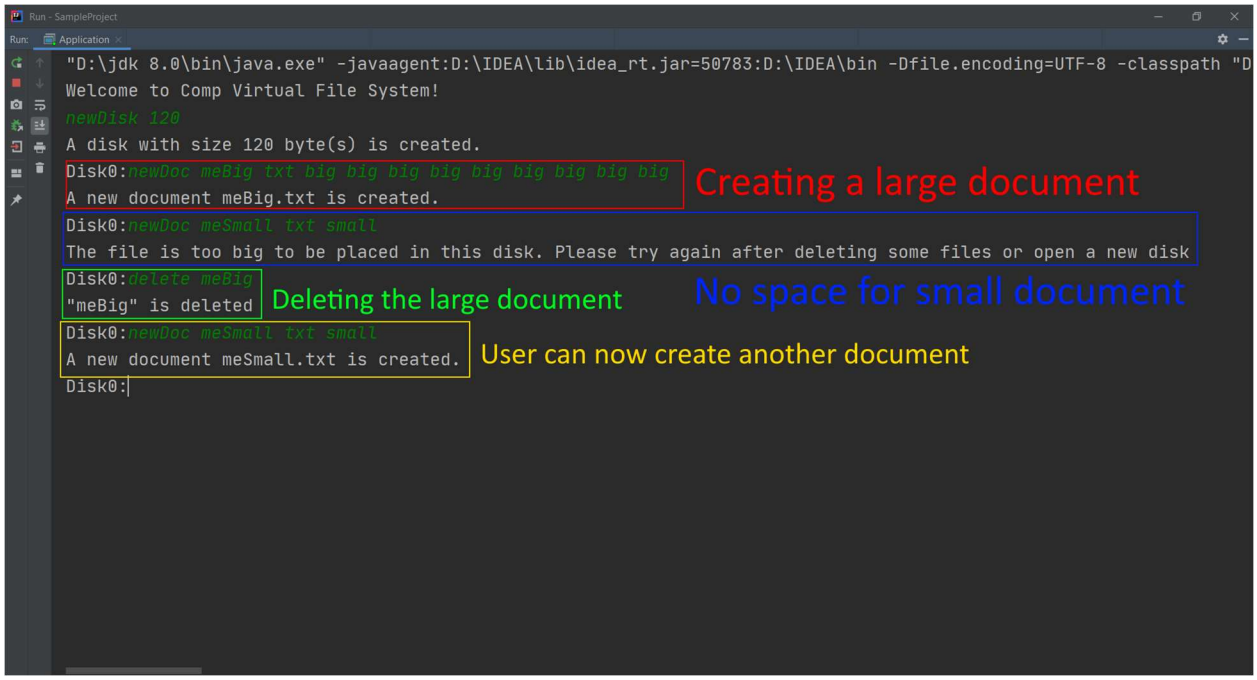


To go back to the parent directory, the `dirName` should be `..`



Deleting a Document or Directory

A disk might not be able to store all the files you would like to place. You can try delete some files (document or directory) to make room for the new file.



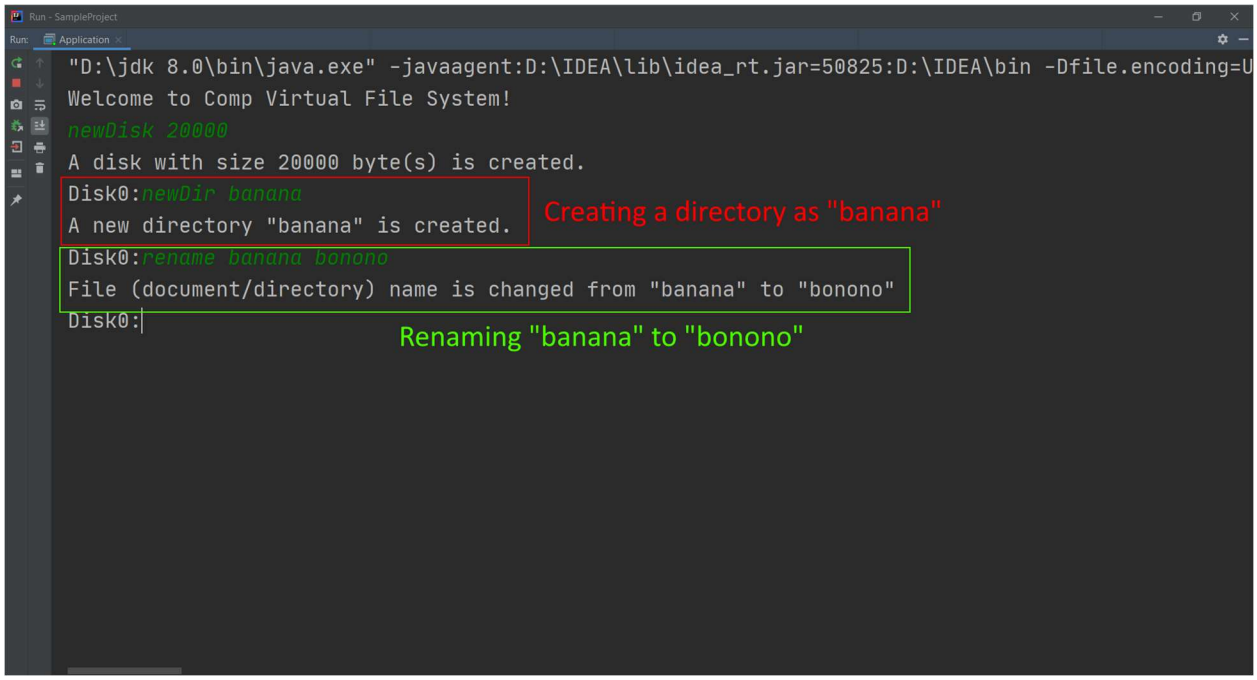
Renaming a Document or Directory

You can rename your document or directory by the following command:

```
rename oldName newName
```

Where `oldName` refers to the original name of the file.

`newName` refers to the new name of the file



Listing and Recursive Listing

To view all the files in the current directory, you can try the command:

```
list
```

```
A new directory "apple" is created.
Disk0:changeDir banana
Current directory is changed to "banana"
Disk0\banana:newDir bonono
A new directory "bonono" is created.
Disk0\banana:newDir benene
A new directory "benene" is created.
Disk0\banana:newDoc wao txt wow
A new document wao.txt is created.
Disk0\banana:changeDir ..
Travelled to the parent directory
Disk0:list
 ├── DIRECTORY banana - Size: 166
 └── DIRECTORY apple - Size: 40
Total number of file(s): 2
Total size: 206
Disk0:
```

Listing the current directory

To view all the files, including the files in the directories, in the current directory, you can try the command:

```
rList
```

```
A new document wao.txt is created.
Disk0\banana:changeDir ..
Travelled to the parent directory
Disk0:list
 ├── DIRECTORY banana - Size: 166
 └── DIRECTORY apple - Size: 40
Total number of file(s): 2
Total size: 206
Disk0:rList
 ├── DIRECTORY banana - Size: 166
 |       ├── DIRECTORY bonono - Size: 40
 |       ├── DIRECTORY benene - Size: 40
 |       └── DOCUMENT wao.txt - Size: 46
 └── DIRECTORY apple - Size: 40
Total number of file(s): 5
Total size: 206
Disk0:
```

Non-recursive listing

Recursive listing

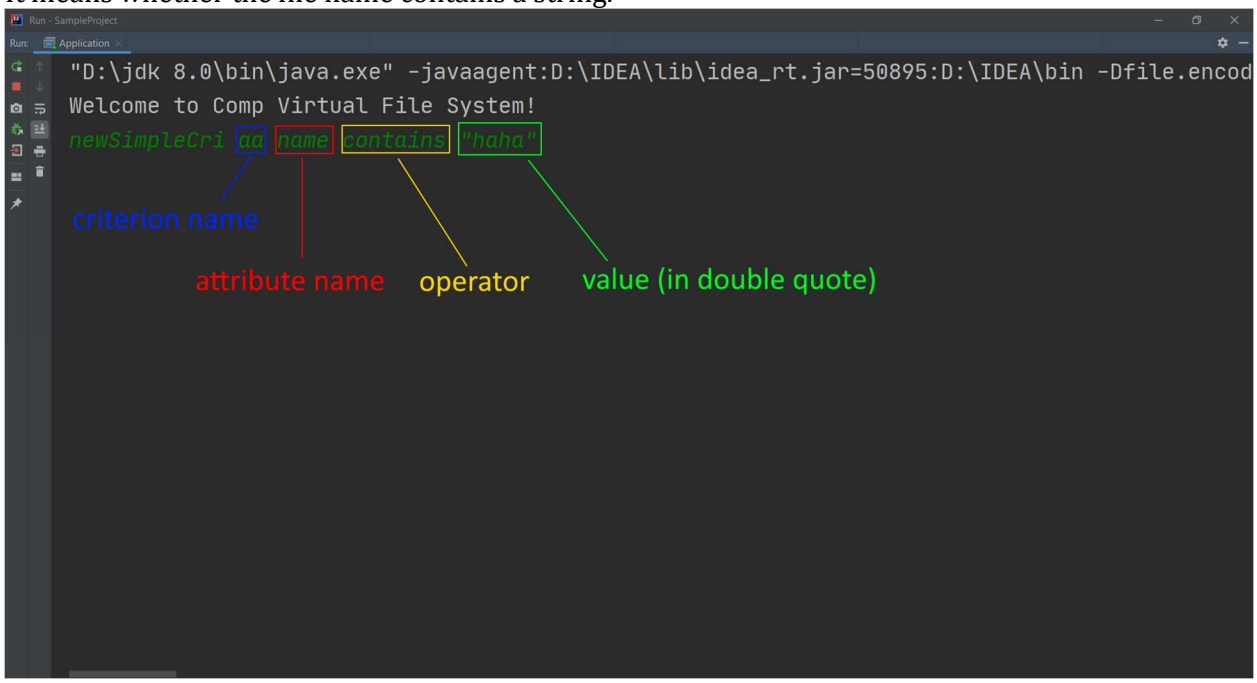
Criteria and Searching

Having trouble finding your files? You can set some criteria to filter out the unwanted result.
Try creating a simple criterion with the following command:

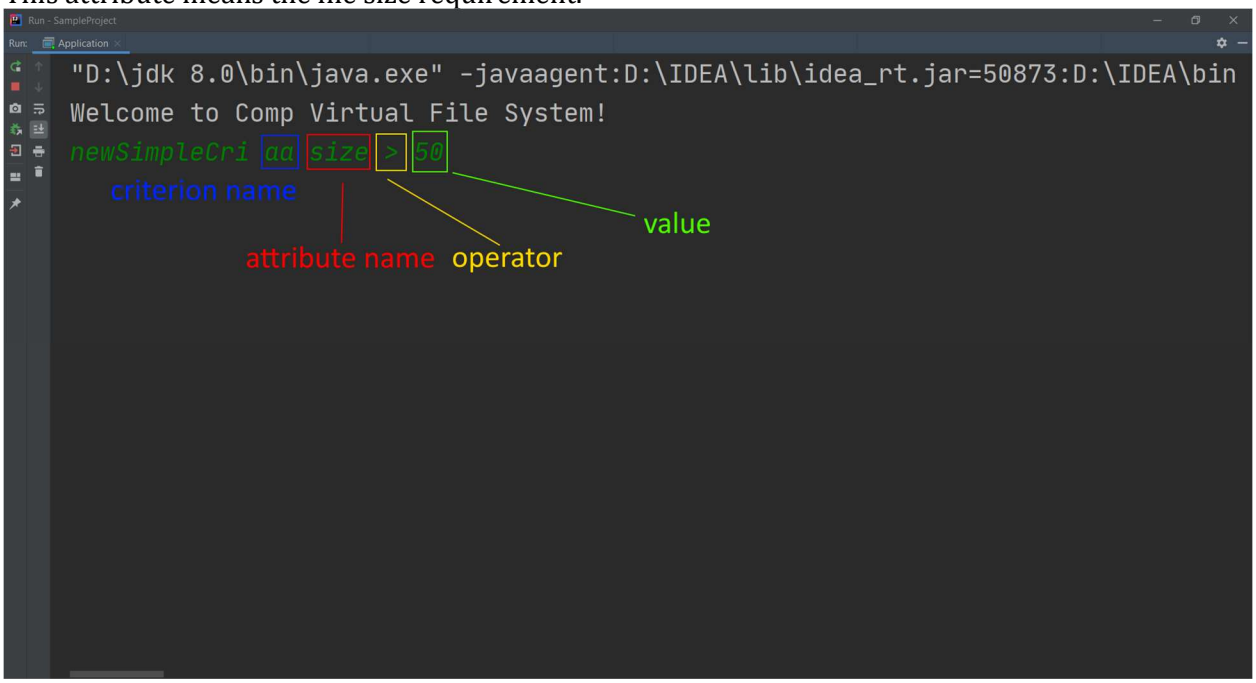
```
newSimpleCri criName attrName op val
```

`criName` stands for the criteria name, which can only contain two English letters.
`attrName` stands for the attribute name, which can only be `name`, `type` or `size`.

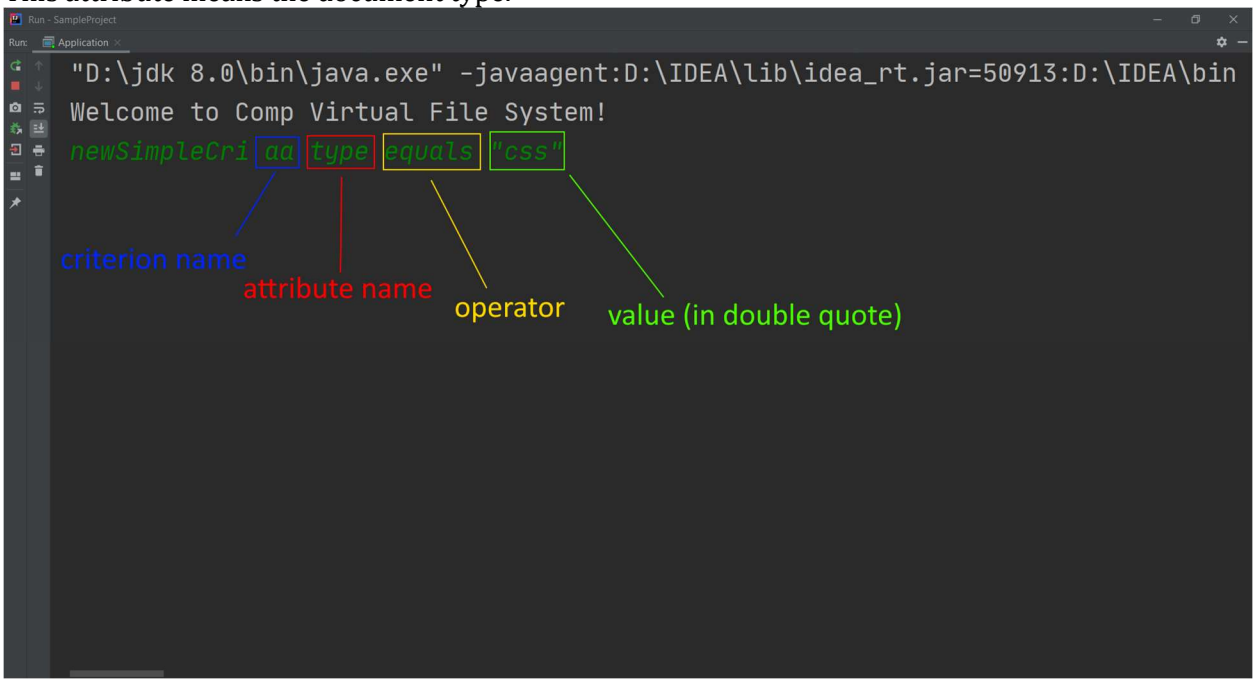
If the attribute name is `name`, `op` (operator) has to be `contains`, and `val` should a string enclosed in double quotation mark
It means whether the file name contains a string.



If the attribute name is `size`, `op` has to be either >, <, >=, <=, == or `!=`. `val` has to be a non-negative numeric value
This attribute means the file size requirement.



If the attribute name is `type`, `op` has to be `equals`. `val` has to be one of the four document types, enclosed in double quotation marks.
This attribute means the document type.

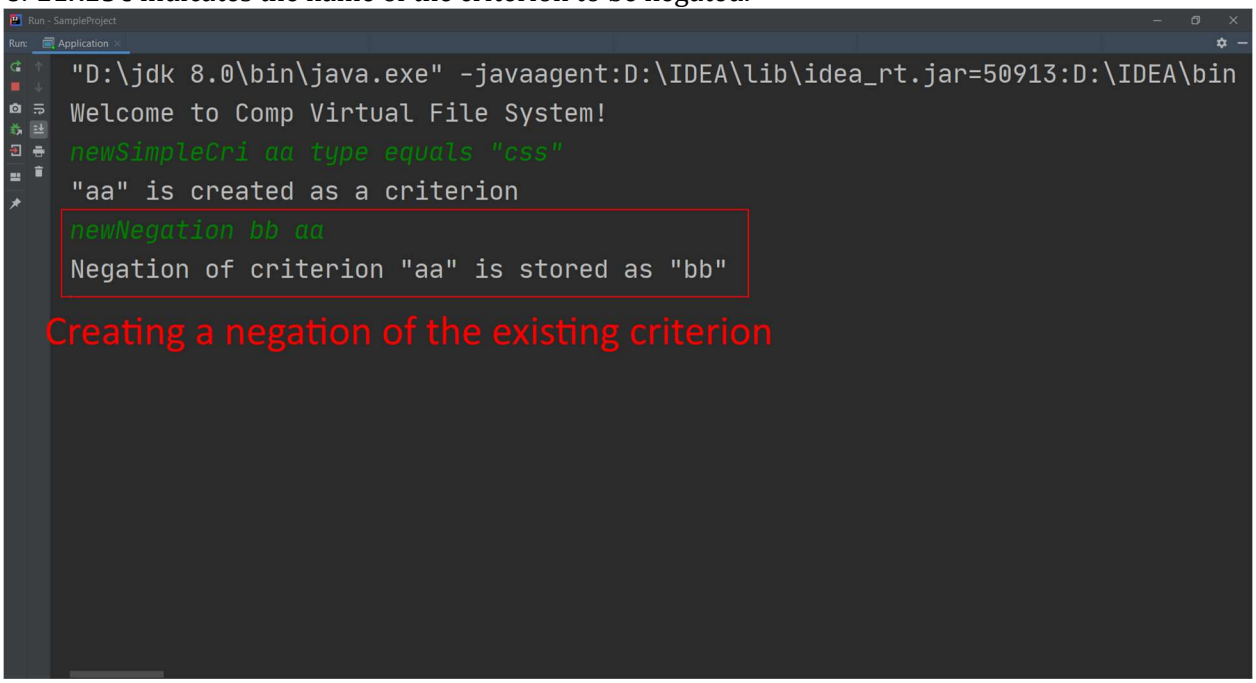


In addition to user-defined criteria, the system also comes with a pre-defined criterion namely `IsDocument`. It determines whether the file is a document.

To create negated criterion, use the follow command:

```
newNegation criNew criExist
```

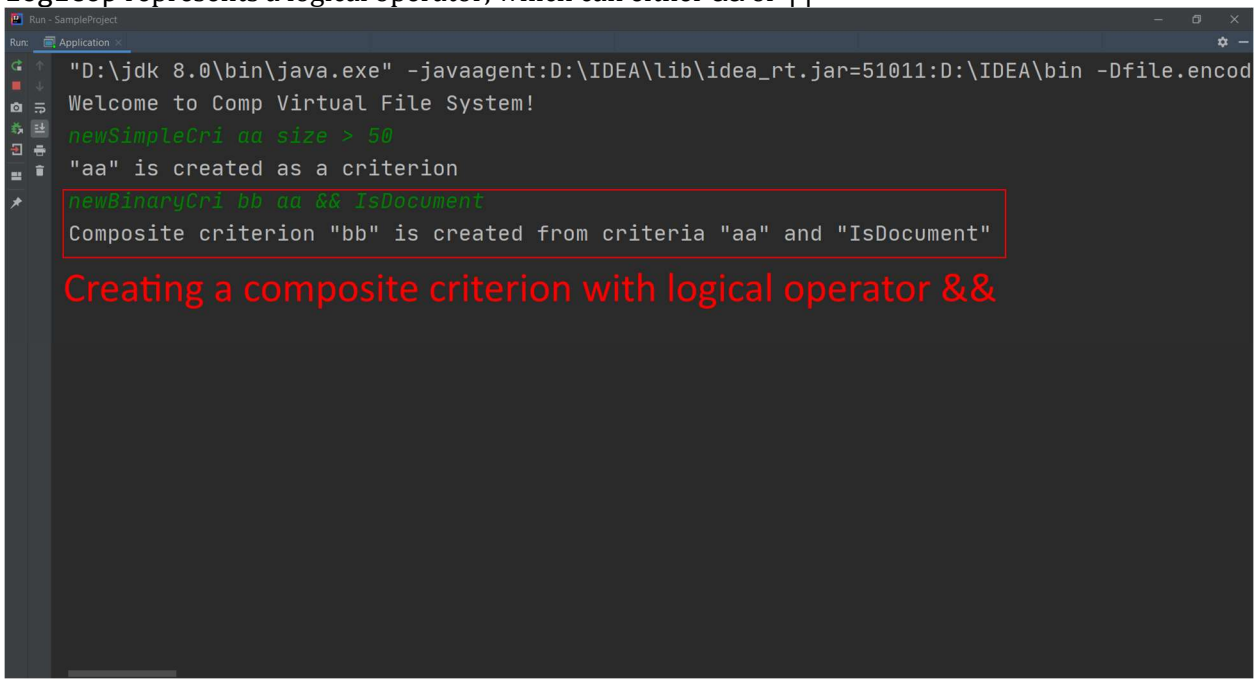
Where `criNew` indicates the name of the negated criterion to be created
`criExist` indicates the name of the criterion to be negated.



To combine two existing criteria, use the follow command:

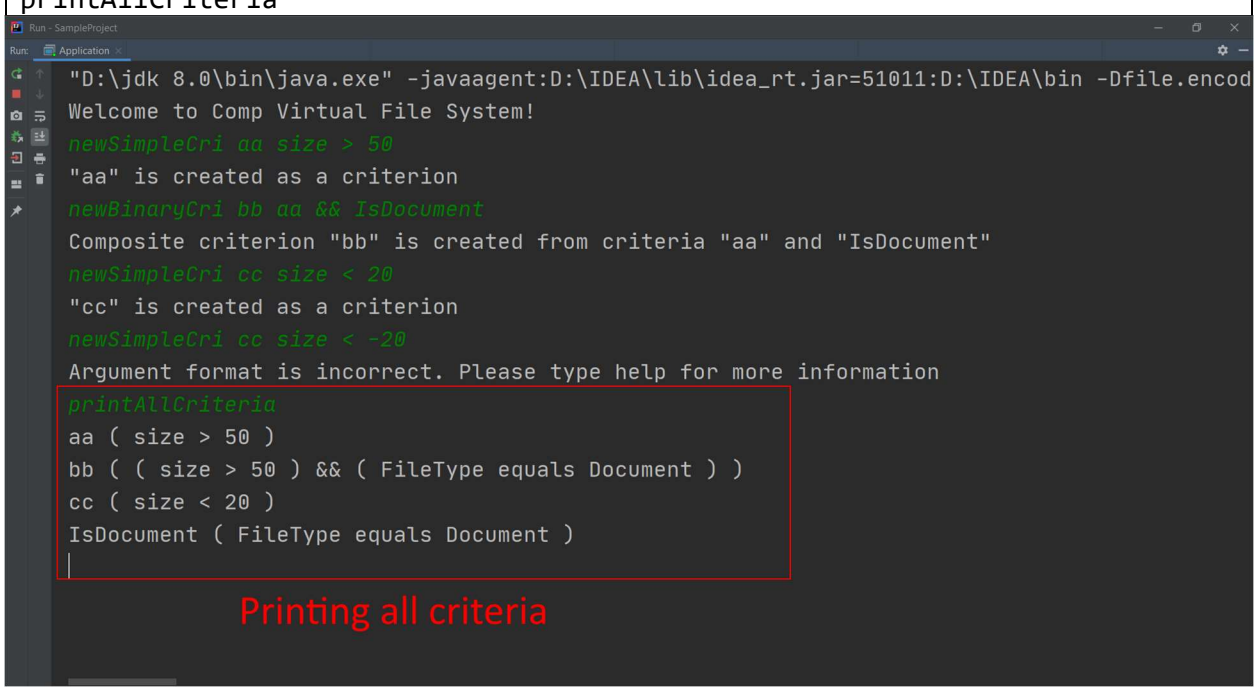
```
newBinaryCri criNew criExist1 logicOp criExist2
```

Where `criNew` is the name of the composite criterion to be created
`criExist1` and `criExist2` mean the name of the two existing criteria
`logicOp` represents a logical operator, which can either && or ||



To view all existing criteria, you can type:

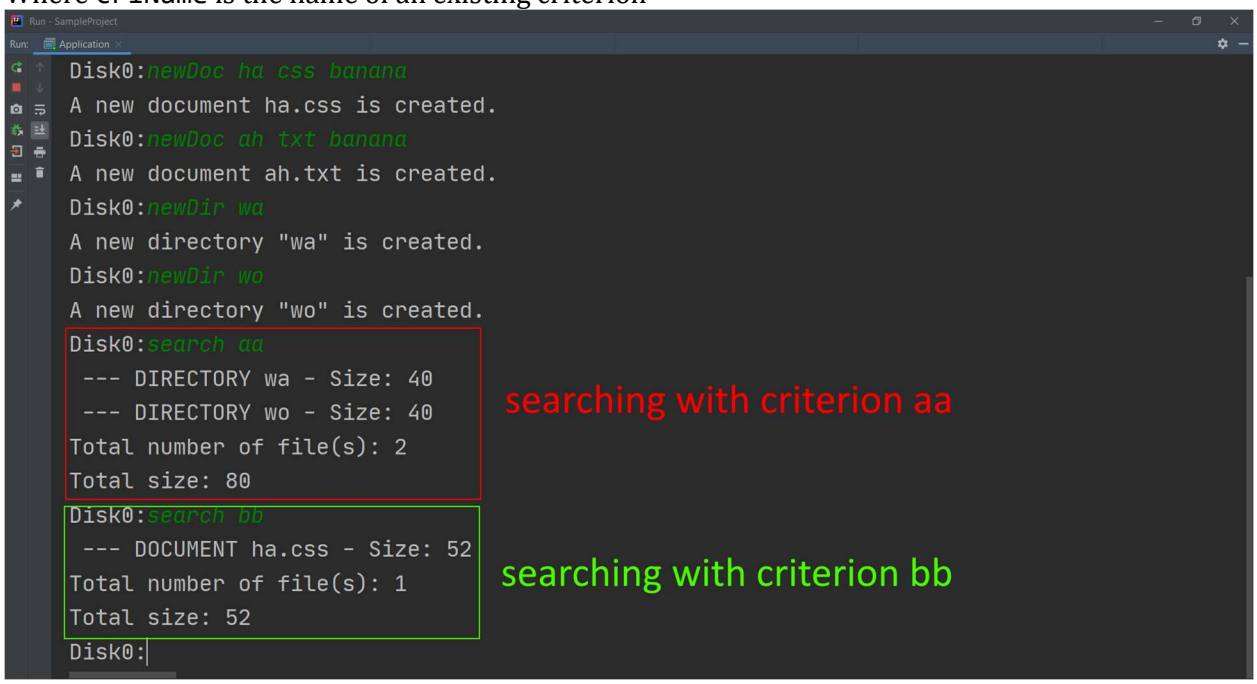
```
printAllCriteria
```



To search the required files directly located in the current directory (given an existing criterion), you can type the following command:

```
search criName
```

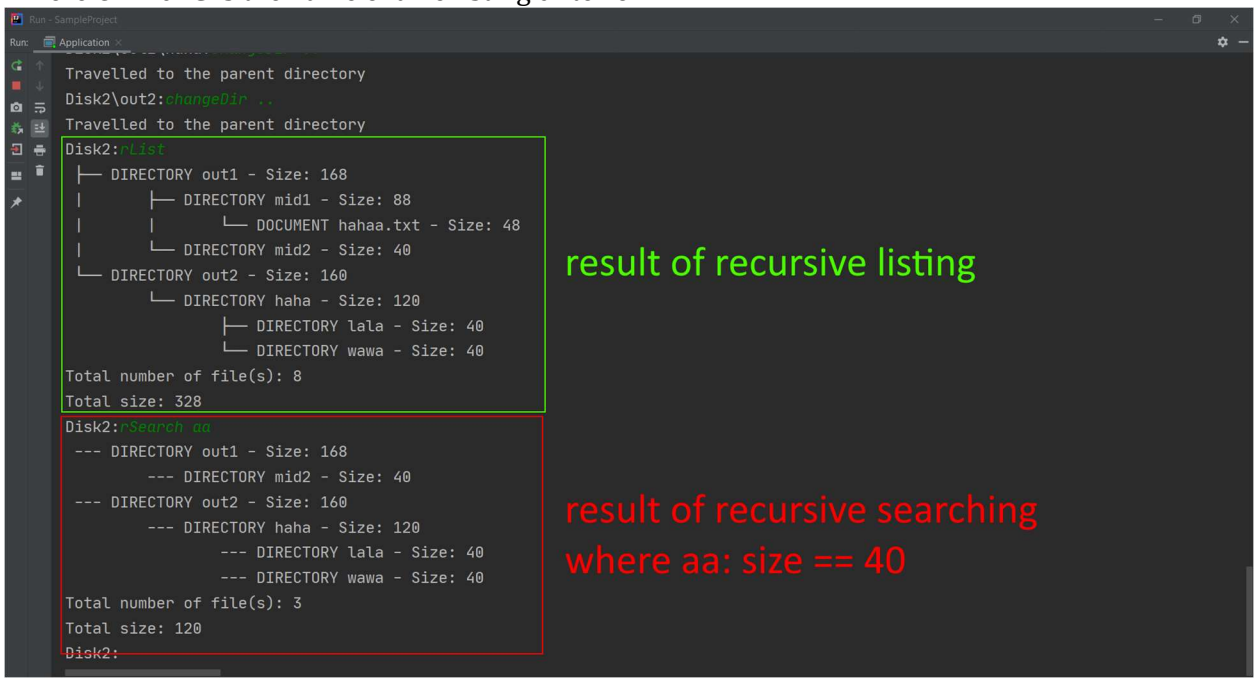
Where `criName` is the name of an existing criterion



To search recursively (given an existing criterion), you can type the following command:

```
rSearch criName
```

Where `criName` is the name of an existing criterion



To quit the current CVFS session, type:

```
quit
```

```
Run - SampleProject
Run: Application
 ├── DIRECTORY out1 - Size: 168
 |       ├── DIRECTORY mid1 - Size: 88
 |       |       └── DOCUMENT hahaa.txt - Size: 48
 |       └── DIRECTORY mid2 - Size: 40
 └── DIRECTORY out2 - Size: 160
         └── DIRECTORY haha - Size: 120
                 ├── DIRECTORY lala - Size: 40
                 └── DIRECTORY wawa - Size: 40
Total number of file(s): 8
Total size: 328
Disk2:rSearch aa
 --- DIRECTORY out1 - Size: 168
         --- DIRECTORY mid2 - Size: 40
 --- DIRECTORY out2 - Size: 160
         --- DIRECTORY haha - Size: 120
                 --- DIRECTORY lala - Size: 40
                 --- DIRECTORY wawa - Size: 40
Total number of file(s): 3
Total size: 120
Disk2:quit
You have quitted the Comp Virtual File System

Process finished with exit code 0
```

Quitting the CVFS session