Project Report

COMP2021 Object-Oriented Programming (Fall 2024)

Group 30

Members and contribution percentages: (we got special case)

Au Tsz Kin:

Chan Chi Wing:

Chau Yan Tung:

Chen William: 0%

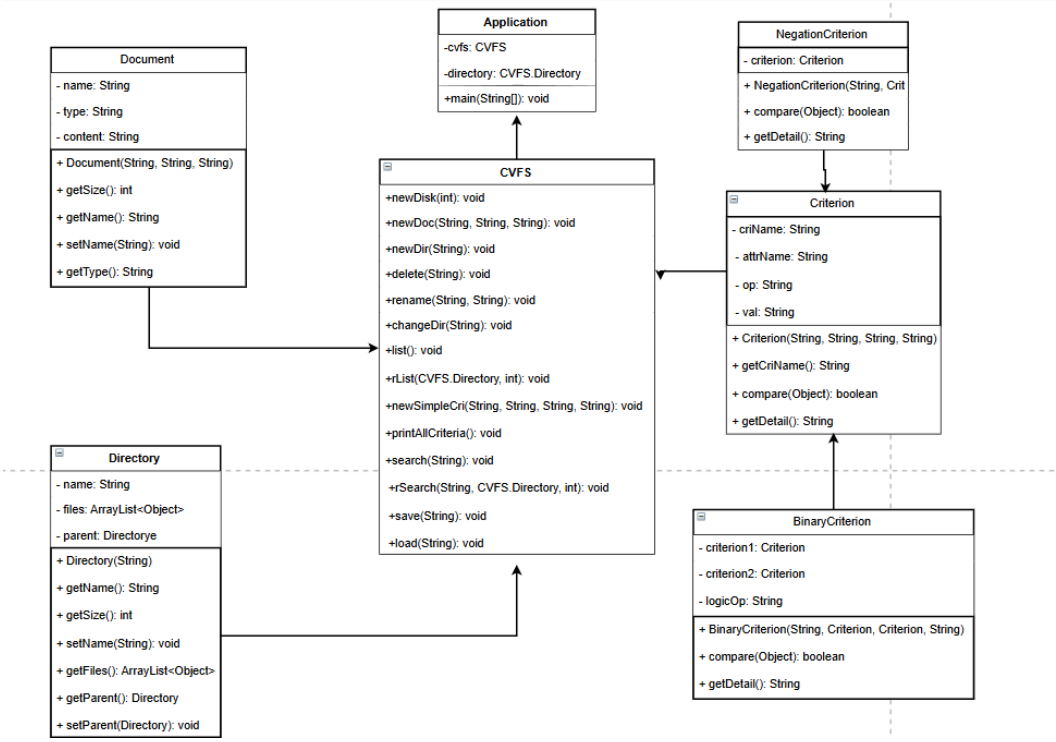
1 Introduction

This document describes the design and implementation of the Comp Virtual File System (CVFS) by group 30. 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



There are a total of 7 classes in our CVFS. The Application class manage the user interaction with the file system, starting with welcome and inputting the commands. The commands of the virtual file system are managed by CVFS class with 3 supporting classes Document, Directory and Criterion, and the left two classes NegationCriterion and BinaryCriterion are the extended classes of Criterion.

2.2 Implementation of Requirements

[REQ1] **newDisk**

1) The requirement is implemented.

2) At start, this is very simple. We decided to use ArrayList to store files (ArrayList<object> because the files can be document or directory). We declared variable “maxSize” to indicate the size of the disk, when we run this function, parameter will work as the maxSize of the disk and clear the ArrayList to make sure the new disk is fill with any files.

3) no error condition

[REQ2] **newDoc**

1) The requirement is implemented.

2) We declared a class “Document” to store documents, class included doc name, type, content, getter of size. We also declared curSize and set it to 0 in **newDesk** to calculate and track the size in the disk. When the function is called, if there are no error input, doc will be added to files, doc.getSize() will be added to curSize.

3) First, we have to ensure the file name is unique, we ask GenAI for help and it tell us the **filesexist** helper method to scan all the files in the desk and see there are same name, so we add getter getName() in doc for implement filesexist. Next, we ensure only digits and English letters are allowed in file names and only documents of types of txt, java, html, and css are allowed in the system. Finally, when curSize after adding the new doc exceeds maxSize, error message will be shown, and doc can’t be added.

[REQ3] **newDir**

1) The requirement is implemented.
2) We declared a class “Directory” to store documents, class included doc name, ArrayList for files and getter of size. The getter is pretty hard because directory can be recursive. Most of When the function is called, if there are no error input, directory will be added to files, dir.getSize() will be added to curSize
3) Most of them are the same as **newDoc,** including **filesexists**, this helper method checks both document and directory files.

[REQ4] **delete**
1) The requirement is implemented.
2) Added **getFile()** using the same concept as **filesexist()** for us the get files from the disk, then simply remove the files from cvfs.files and minus the size of the files from curSize
3) To delete a file, we need to ensure the file exists, filesexists is used to check whether the file exists.

[REQ5] **rename**
1) The requirement is implemented.
2) Added setter setName in class Document and Directory to help us rename the file.
3) First use **filesexist()** check whether the oldFileName is exists, if no then error. Then check whether the new file name has already existed, if yes then error.

[REQ6] **changeDir**
1) The requirement is implemented.
2) This function changes our code a lot, at first when we create class Directory and function like **newDisk**, **newDir,** we do not consider the work directory, root directory, parent directory etc.

So, we made a big change to our code. First, we declared parent in class Directory to indicate the parent of **this** directory, also implemented setter and getter of parent. Also, we deleted the first thought of use ArrayList in class CVFS to store files, instead, we see intial disk as a Directory “root” to store files.

Second, we declared root in class CVFS to indicate the root of the disk, curDir to keep

track of the working directory. Added **root = new Directory("root")** and **curDir = root** in function **newDisk** to set the inital disk be the root directory and current working directory. We modified newDoc, newDir adding method due to those changes, getter **GetFiles** in class Directory is impkemented for us to obtain ArrayList files from the curDir (i.e all **files** are replaced by **curDir.getFiles()**). In **newDir,** when we create a new Directory, we set the parent directory of the new dir to the current working directory**.** CVFS contructor is now empty because we have got **newDisk** to initialize everything.

Finally, is the implementation of **changeDir**, when ".." is the input, the curDir change to the parent directory of the current working directory by using the getter **getParent** that we just implemented. If input is not "..", we check the dirName by getFile, and set the curDir to targetDir.

Honestly this is a big change of our program, so we were a little overwhelmed, so we took the help of GenAI, to help us reorganized the code by providing our own thoughts.

3) When ".." is the input, if the current working directory is the root directory, error occur so we check this by **curDir == roo**t. For the non ".." input, we make sure targetDir is not null and it is a directory file by **getFile.**

[REQ7] **list**
1) The requirement is implemented.

2) Set totalSize and totalFiles to 0, use for loop to loop all the files in curDir, keep tracking the files count, print them all and add all their size by **getName** and **getSize**, implemented getter **getType()** for printing the type of doc.

3) no error condition

[REQ8] **rList**
1) The requirement is implemented.

2) most of them are same as **list,** but in this we added parameter **level** to indicate indentation, adding **" ".repeat(level)** in front of each printing statement. When we

call this function, we initialize the level = 0 (i.e. in application, we call **cvfs.rList(0)**) We add **rList(dir, level + 1)** recursion under directory part so it will search the files in that directory until no more directory is found.

3) For output the total files and size, as it will only output once so from prevent it output every time recursion of **rList** is called we set that only print when level is equal to 0.

[REQ9] **newSimpleCri**

1) The requirement is implemented.

2) First, we have to create a class Criterion, including criName, attrName, op and val, Implemented getter **getCriName**. In class CVFS, we declared ArrayList<**criterion>** to store criterion data. Initialize criterion crit in cvfs constructor, and clear when **newDisk** is called.

When the **newSimpleCri** is called, if there is no input error, new Criterion will be created and add to ArrayList crit.

3) There are lots of input limit, first we make sure that a criName contains exactly two English letters. Second, is the attrName, op and val input checking, separated into 3 case by || . For the size integer checking, we ask GenAI for help, learned how to use try-catch block and **Integer.parseInt()** for the checking.

[REQ10] **IsDocument**

1) The requirement is implemented.

2) Since this is the default criterion, we just simply add this to the ArrayList crit when the CVFS is initialized. We add compare function (which help us if a file meets criteria a file in **Search/rSearch** function) in Criterion class, when the criName is equal to “IsDocument”, it will evaluate if a file is document (it is really similar to our filesexist method).

3) no error condition.

[REQ11] **newNegation newBinaryCri**
(This is the last req we implemented for the criteria part so please skip and back after rSearch)

1) The requirement is implemented.

2) For the negation, we create a class for negationCriterion which is inherit from class Criterion. We receive criterion from the original criterion, override criterion.compare method to return **!criterion.compare(file) which** will return a negative result of the original criterion. We also need to override **getDetail** for **printAllCriteria. ~ is used for indicate NOT**

For the binary, we also create a class for binaryCriterion which is inherited from class Criterion. Override compare method, its logicOp is either && and || so we split it into two cases, return crit1 && crit2 or crit1 || crit2 according to the logicOp.

For both **newNegation** and **newBinaryCri** they are similar to the implementation of **newSimpleCri**

3) check criName(criName2 for **newNegation**, criName3 and 4 for **newBinaryCri)** if it is existed criterion. Check logicOp if it is either "&&" or "||".

[REQ12] **printAllCriteria**

1) The requirement is implemented.

2) Loop the ArrayList crit, implement **getDetail()** to output all attribute (critName, AttrName, op and val) of a criterion.

3) **IsDocument** is a criterion without op and val, so this is a special case we have to check if it is IsDocument and it just simply output "IsDocument" in **getDetail()**

[REQ13] **search**

1) The requirement is implemented.

2). Implemented help method getCriterion to get criterion from a String criName. This method loops the list crit and compares if it Criterion criName is equal to parameter criName, return null if criterion is not found.

The challenge of this fuction is the **compare (Object files)** in class Criterion, because format of **search** is really similar to **list** we just did before. The only difference is that we have to compare the files (doc or dir) to the criterion.

We use switch(attrName) to seperate to 3 cases, but we found that files getter can't be used in type Object (but both doc and dir can match the criterion), we can't think other way, so we decided to use instanceof in every case to identify the Object is document or directory. (Yes, that makes this method become so long)

3) If the criterion is null (i.e. the criterion not found), an error message will be sent.

[REQ14] **rSeach**

1) The requirement is implemented.

2) This function is a combination of **rList and search**, rSearch has got 3 parameters. In application code, the command is received, **rList("command", curDir, 0)** should be called

3) If the criterion is null (i.e. the criterion not found), an error message will be sent.

[REQ15]

1) The requirement is implemented.

[REQ16]

1) The requirement is implemented.

[REQ17]

1) The requirement is implemented.

2) by turning the boolean variable “running” in Application to false when the method is called.

3) no error condition.

3 Reflection on My Learning-to-Learn Experience

In looking back at the project's learning process, tackling new Java and object-oriented programming concepts was tough but valuable. When doing this project, quite a lot of problems occur. To satisfy all the requirements of CVFS, we found that some of them are not taught from the lecture. For example, in REQ2, we know that the name of the file should be unique but what method can be helped was not sure. REQ6 is tough to do too. At first, we only created a class called Directory but did not consider the work directory, root directory and parent directory. We need to make a big change to our code. We know roughly how the whole thing works, but there are so many steps, which makes us feel a little overwhelmed as to how to write this part of the code perfectly. Also, REQ9, we found that this part of the code has quite a lot of input limitations, like the criName should contains exactly two English letters, the attrName, op and val should be matched. Some of them are easy to reach, but we stuck on the val of size at first, which should be an integer, and we don’t know what method can be used for the checking. We not only encounter the above-mentioned problems, but also face difficulties in various other aspects. However, we also know that there are many internet sources that can be used to help us to solve those problems. Generative AI is one of the tools that help us so much. For REQ2, we ask GenAI, and it tells us **filesexist** can be used, and we search this method on the internet and learned how to use it. The problem from REQ9 also uses the same way to solve. And to solve the problem from REQ6, we also seek help from GenAI. We provide our thoughts on this part to GenAI and let our code being more well-organized. It really enhances our problem-solving skills.
However, we still have some problems that cannot be solved well, like REQ13. We

can’t find a good way to identify whether the object is document or directory and just simply use instanceof in every case. It is the place that we need to improve. We are planning to do more practice with the coding exercise related to object-oriented programming and Java. Also seeing other groups’ code and their design pattern to gain more valuable insights in the future design review section. We would like to explore online courses about the advanced topics of object-oriented programming and Java to deepen our knowledge and understanding in the future. This project is a great learning experience for us.