Project Report

COMP2021 Object-Oriented Programming (Fall 2024)

Group 19

Members and contribution percentages:

Member Cheung Kei Yau: 25%

Member Chong Zhi Yi: 25%

Member Lau Ming Yin: 25%

Member Lee Yin Lam Elaine: 25%

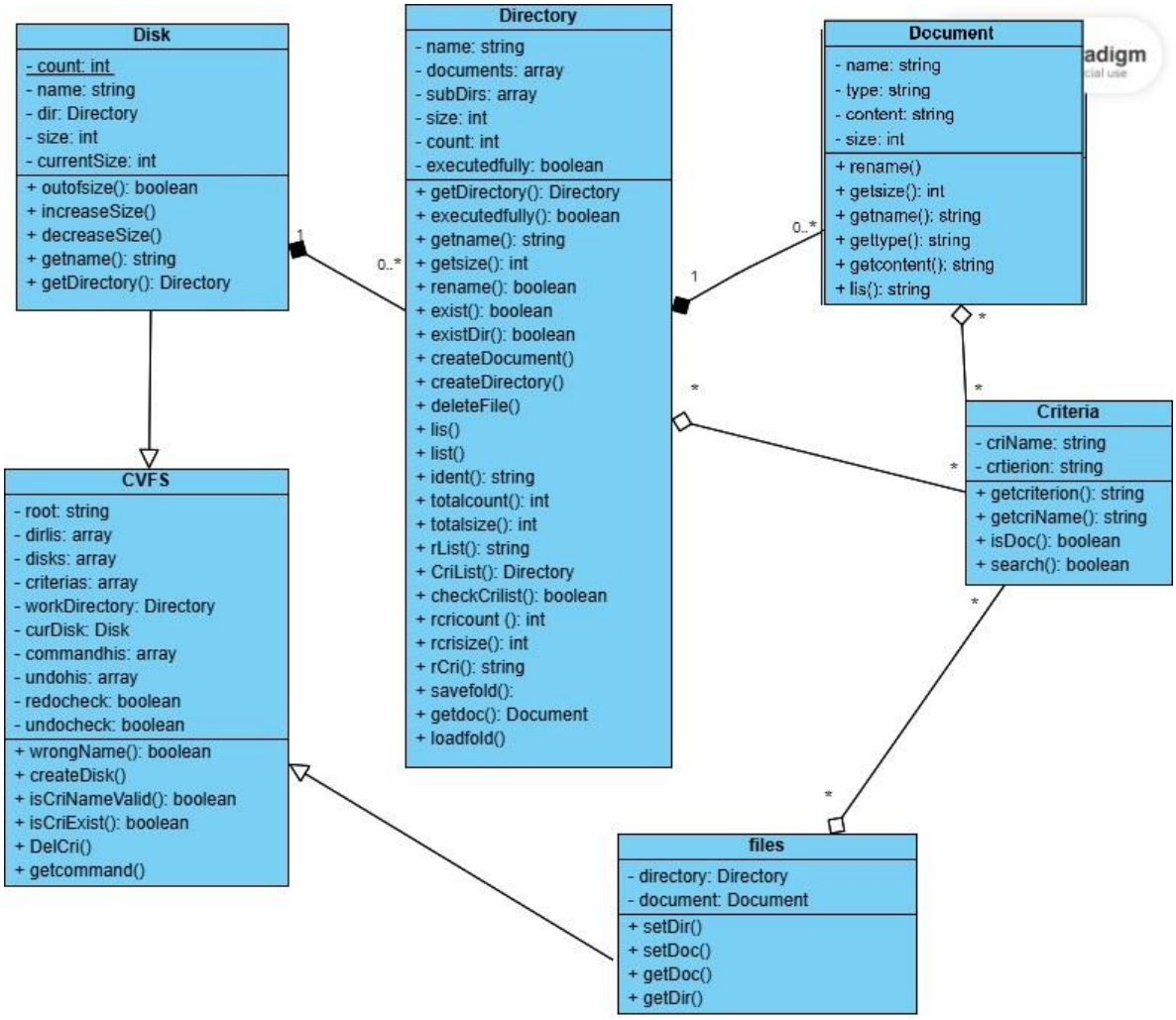
1) Introduction

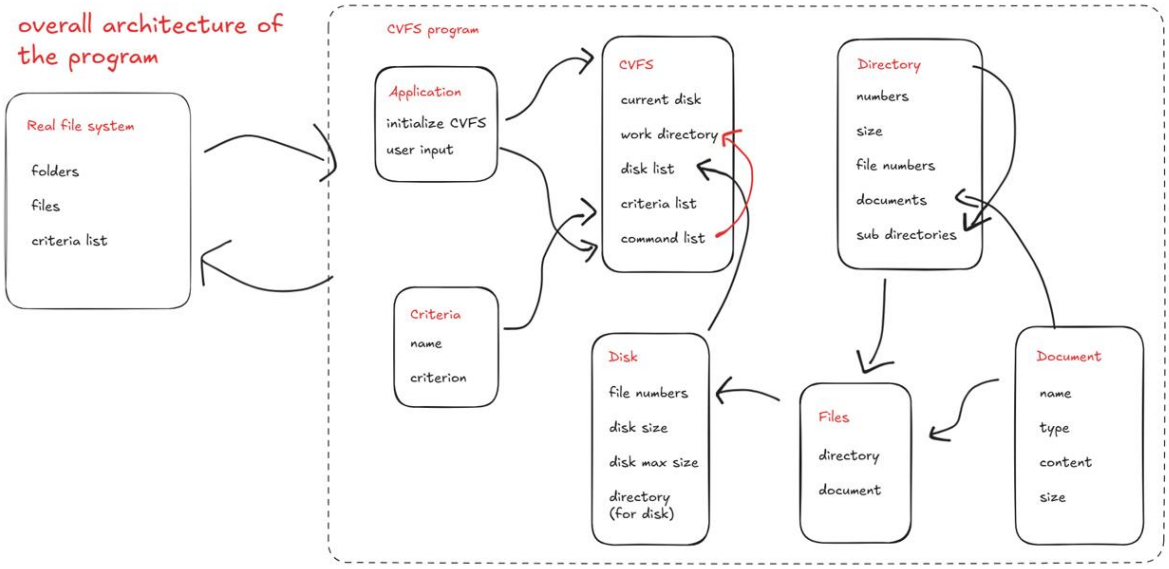
This document describes the design and implementation of the Comp Virtual File System (CVFS) by group 19. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

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


Disk
- count: int
- name: string
- dir: Directory
- size: int
- currentSize: int
+ outofsize(): boolean
+ increaseSize()
+ decreaseSize()
+ getname(): string
+ getDirectory(): Directory
Directory
- name: string
- documents: array
- subDirs: array
- size: int
- count: int
- executedfully: boolean
+ getDirectory(): Directory
+ executedfully(): boolean
+ getname(): string
+ getsize(): int
+ rename(): boolean
+ exist(): boolean
+ existDir(): boolean
+ createDocument()
+ createDirectory()
+ deleteFile()
+ lis()
+ list()
+ ident(): string
+ totalcount(): int
+ totalsize(): int
+ rList(): string
+ CriList(): Directory
+ checkCrilist(): boolean
+ rcricount (): int
+ rcrisize(): int
+ rCri(): string
+ savefold():
+ getdoc(): Document
+ loadfold()
Document
- name: string
- type: string
- content: string
- size: int
+ rename()
+ getsize(): int
+ getname(): string
+ gettype(): string
+ getcontent(): string
+ lis(): string
Criteria
- criName: string
- crtierion: string
+ getcriterion(): string
+ getcriName(): string
+ isDoc(): boolean
+ search(): boolean
CVFS
- root: string
- dirlis: array
- disks: array
- criterias: array
- workDirectory: Directory
- curDisk: Disk
- commandhis: array
- undohis: array
- redocheck: boolean
- undocheck: boolean
+ wrongName(): boolean
+ createDisk()
+ isCriNameValid(): boolean
+ isCriExist(): boolean
+ DelCri()
+ getcommand()
files
- directory: Directory
- document: Document
+ setDir()
+ setDoc()
+ getDoc()
+ getDir()


overall architecture of the program
Real file system
folders
files
criteria list
CVFS program
Application
initialize CVFS
user input
CVFS
current disk
work directory
disk list
criteria list
command list
Directory
numbers
size
file numbers
documents
sub directories
Criteria
name
criterion
Disk
file numbers
disk size
disk max size
directory
(for disk)
Files
directory
document
Document
name
type
content
size

Component Explanation

CVFS Class:

The main controller of the virtual file system. It manages disks, directories, and user commands. It includes methods to create disks, manage documents, and apply criteria for searching and filtering files.

Disk Class:

Represents a disk in the virtual file system. It has attributes for storing its name, size, and the root directory. Methods include functionality to check for size limits and manage the disk's capacity.

Directory Class:

Represents a directory that can contain documents and subdirectories. It includes methods to create new documents, create subdirectories, delete files, and list contents.

Document Class:

Represents a file within a directory. It contains attributes for the document's name, type, and content, along with methods to access these properties.

Criteria Class:

Encapsulates the criteria used for searching and filtering files within the system. It includes methods for evaluating whether a file meets specified conditions (e.g., name, type, size).

Files class:

Acts as a utility class to manage the relationship between directories and

documents, holding references to both.

Application class:

The class is responsible for initializing the CVFS system, handling user input, and coordinating interactions between the user and the underlying components. The class also have error-handling features that guarantee user input is valid and offer feedback in case of issues.

Design Patterns Used

Command Pattern:

The getcommand method in the CVFS class implements a command pattern to handle user inputs flexibly. Each command (e.g., newDoc, delete, changeDir) is processed in a unified manner, allowing for a modular approach to command handling.

Composite Pattern:

The Directory class can contain both Document and other Directory instances, allowing for a tree-like structure. This pattern facilitates operations on individual files and directories uniformly.

Strategy Pattern:

The Criteria class acts as a strategy for searching files. Different criteria can be applied dynamically, allowing for flexible searching mechanisms without altering the the main system structure.

2.2) Implementation of Requirements

[REQ1]

1) The requirement is implemented.
2) Implementation details.

   The implementation checks for a command to create a new disk. It parses the disk size from the input, handling potential NumberFormatException for non-integer values. If the input is insufficient (only if " " would cause this exception), it catches ArrayIndexOutOfBoundsException. Successful disk creation sets the new disk as the working disk, and the working directory is set to be the root directory of the disk. For input command ("newDisk 1000"), the system outputs a successful message: Disk created! For disk size with any integer (e.g. 123), the system outputs "Disk created!" as successfully executed. The previous working disk, if any, is closed.
3) Error conditions and error handling.

   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   When non-integer (e.g. "abc", "fifty") or "null" values are inputted, the system throws NumberFormatException to remind users must input integer.

[REQ2]

1) The requirement is implemented.
2) Implementation details.

   The newDoc command allows users to create a document in the working directory. It validates the document name, constructs the document with specified name, type, and content, and checks if the current disk has enough space. Upon success, it updates the disk size and manages command history.
3) Error conditions and error handling.

   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

When user input a document name already exists in the disk, the system displays an error message, and the new document will not be created.

Documents without name or type, having non-digits and non-letters, name longer than 10 units, or document type are not “txt/java/html/css” are not allowed in the system. The system returns an error message to user of how a document is accepted.

When the disk is out of space, the system would not create any new document and remind user the disk is out of space to add the file.

[REQ3]

1) The requirement is implemented.
2) Implementation details.
   The newDir command enables the creation of a new directory in the working directory. It first checks the validity of the directory name using wrongName(). If valid, it creates a Directory object and verifies available disk space. Upon successful creation, it updates the disk size and manages command history.
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   Directories without name, having non-digits and non-letters, name longer than 10 units, are not allowed in the system. The system returns an error message to user of how a directory is accepted. Having same directory name is also not allowed in the system.

   When the disk is out of space, the system would not create any new directory and return "The disk is out of space to add this file".

[REQ4]

1) The requirement is implemented.
2) Implementation details.
   The delete command allows users to remove an existing file from the working directory. It first checks if the specified file exists. If found, it retrieves the file's details for potential undo purposes. Upon successful deletion, it updates the disk size and lists the remaining files in the directory. It also supports undo (to be mentioned in a later section).
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   When deleting a file does not exist, the system displays error message “File not found!”.

[REQ5]

1) The requirement is implemented.
2) Implementation details.
   The rename command enables users to rename an existing file in the working directory. It first checks if the new file name is valid using wrongName(), then verifies the existence of the specified old file. If both checks pass, it proceeds to rename the file and updates the file list accordingly.
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   If the old file name is not found, or the new file name is invalid, the system would not execute the command and display an error message.

[REQ6]

1) The requirement is implemented.

2) Implementation details.
The changeDir command allows users to navigate between directories in the working directory. It verifies if the specified directory exists. If the input is "..", it goes to the parent directory. It first checks if the current directory is already the root; if not, it moves to the parent directory. Upon successful change, it updates the working directory and displays the new directory contents. The command also manages the command history for potential undo operations.

3) Error conditions and error handling.

When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

When changing a directory with the directory name not found, the system returns an error message that the directory is not found.

When using “..” to change the directory, and the current directory is in the root directory, the system displays "This is already the root directory!" To remind user that the directory is already set to the root.

[REQ7]

1) The requirement is implemented.
2) Implementation details.
   The list command displays all files and directories in the working directory. It counts and presents each document's name, type, and size, along with each directory's name and size. If there are no documents or directories, it indicates that the directory is empty. Finally, it reports the total number of

items and their cumulative size.

[REQ8]

1) The requirement is implemented.
2) Implementation details.
   The rList command provides a recursive listing of all files and directories within the working directory. It formats the output with indentation to indicate the hierarchy level of each item. Each document displays its name, type, and size, while directories show their name and size. If no documents or directories exist, it indicates that the directory is empty. Finally, it reports the total number of files and their cumulative size, providing a comprehensive view of the directory structure.

[REQ9]

1) The requirement is implemented.
2) Implementation details.
   The newSimpleCri command creates a simple criterion for filtering files. It validates the criterion name, attribute, operator, and value based on restrictions for name, type, and size. Successful validation adds the criterion to the list, while errors prompt appropriate messages, managing history for potential undo.
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   When user input a criteria name already exists in the disk, the system displays an error message, and the new criteria will not be created.

   For invalid criteria name, attribute name or type, the system would not create any new criteria and return an error message.

[REQ10]

1) The requirement is implemented.
2) Implementation details.
   The IsDocument tool can implement a simple criterion named IsDocument to evaluate whether a file is a document. This criterion checks if the specified file's type matches recognized document formats (java, txt, html, css). If it matches, the criterion evaluates to true; otherwise, it evaluates to false.
3) Error conditions and error handling.

Not doc: false

[REQ11]

1) The requirement is implemented.
2) Implementation details.
   The newNegation command creates a composite criterion that negates an existing criterion, while newBinaryCri constructs a criterion combining two existing criteria using a logical operator (&& or ||). Both commands ensure the validity of criterion names and existence before adding the new criteria to the list.
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   For invalid criteria name, or invalid logic operator, the system would not create any new criteria and return an error message.

   If the new negation or new binary criteria is constructed without creating a new simple criterion, the system would not accept such creation.

   When user input a criteria name already exists in the disk, the system

displays an error message, and the new criteria will not be created.

[REQ12]

1) The requirement is implemented.
2) Implementation details.
   The printAllCriteria command displays all defined criteria in a structured format. It iterates through the list of criteria, printing each criterion's name alongside its components (attribute name, operator, value, logical operator, or IsDocument). This provides a clear overview of all criteria available for filtering.

[REQ13]

1) The requirement is implemented.
2) Implementation details.
   The search command allows users to find files in the working directory based on an existing criterion. It first checks if the specified criterion exists. If found, it retrieves the corresponding criterion and uses it to filter files in the working directory. The command then lists all matching files and reports the total number and size, providing a clear result of the search operation.
3) Error conditions and error handling.
   When user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   When criteria not found, the system returns “Criteria not found!” that there are no criteria found.

[REQ14]

1) The requirement is implemented.
2) Implementation details.

The rSearch command enables recursive searching for files in the working directory based on an existing criterion. It verifies the existence of the specified criterion before retrieving it. The command then calls a method to perform the recursive search, listing all matching files and reporting the total count and size of the results.

3) Error conditions and error handling.
   When the user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

   When criteria not found, the system returns “Criteria not found!” that there are no criteria found.

[REQ15]

1) The requirement is implemented.
2) Implementation details.
   The save command allows users to save the current working virtual disk, including its files, to a specified file path on the local file system. It first checks if a search criterion is applied. If so, it retrieves the relevant files based on the criterion and saves them to a temporary directory before writing to the specified path. If no criteria are active, it directly saves the entire working directory.
3) Error conditions and error handling.
   When the user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

Test cases:

[REQ16]

1) The requirement is implemented.
2) Implementation details.
   The load command allows users to load a virtual disk from a specified file path. It uses the loadfold method to recursively read the directory structure, adding subdirectories and documents. Each document is created by reading its content and type, enabling users to restore their previous work effectively.
3) Error conditions and error handling.
   When the user does not input the number of commands required, the system throws ArrayIndexOutOfBoundsException and an error message to remind users to input sufficient amounts of commands.

[REQ17]

1) The requirement is implemented.
2) Implementation details.
   The quit command allows users to terminate the system's execution. Upon invoking this command, the program prints a message indicating that it is exiting and then calls System.exit(0), effectively shutting down the application. This ensures a clean exit from the tool.

   When the user inputs the word “quit” in any letter cases, the system terminates the execution of the system. Valid inputs including “quit”, “qUiT”, “QUIT”.

[BON1]

1) The requirement is implemented.
2) Implementation details.
   The implementation supports saving and loading the defined search criteria when a virtual disk is saved to or loaded from the local file system. When it is implemented, the criteria created before the implementation is

saved. When it is reopened, it loads the past records of criteria into the local virtual disk.

[BON2]

1) The requirement is implemented.
2) Implementation details.
   The implementation maintains two lists: commandHistory for undo operations and redoHistory for redoing actions. Each command (e.g., newDoc, delete) pushes its details onto commandHistory. Undo methods reverse the command effects, while redo methods reapply them, ensuring users can navigate their actions effectively within the system.
3) Error conditions and error handling.
   When undoing or redoing unspecified commands, the system prints "Previous step cannot be undo." and "Cannot redo without undo." respectively.

   When number of redo is larger than undo, the system prints "Cannot redo without undo." to prevent the action.

3) Reflection on My Learning-to-Learn Experience

Cheung Kei Yau:

Throughout the project, I encountered several aspects of Java and object-oriented programming that went beyond the standard curriculum covered in lectures. I found that learning the theories and implementing the techniques, such as design patterns, are quite different. These difficulties motivated me to find resources on the Internet, such as online tutorials, to practice more on those topics.

I found that hands-on coding experience significantly strengthened my

understanding. Writing tests for the system has required me to apply what I had learned and ensuring the code that we wrote are logically correct. Peer discussions also assisted me in improving my strategy and clearing up misunderstandings. The process of receiving feedback and making adjustments was important for my growth. In order to improve myself, I would do more coding practice from different coding platforms so that I can reflect on what I have learned in the lecture, the challenges I faced and how to overcome them.

Chong Zhi Yi:

This project was challenging enough to enrich our programming skills and problem-solving skills. Throughout the process, I encountered numerous of bugs and problems. By applying the knowledge and skills learned during lectures and lab, I slowly breakthrough the problem one by one. This process consolidates my programming knowledge about JavaScript and object-oriented programming.

Moreover, when I encounter each difficulty, I would do research online and ask my groupmates for guidance and support. These helps me to better understand my weaknesses and improve myself. In addition, when we try to tackle the problems, we try to effectively communicate with each other to clear our misunderstanding or logical errors.

Lau Ming Yin:

After doing the project, I learned more about object-oriented programming and Java. I learned more about how to utilize class in Java and found out how great the class can be as it helped a lot with reusability. At the beginning of creating the code, I had no idea how to use the class and did not know the direction of creating it. I learned a lot on the Internet about how to use the class and found some ways that can help me to make a good foundation of the code. After doing the first few requirements, I realized that with the help of making multiple classes in the model and linking them together can make a good foundation to

complete the rest of the requirements. However, in the criteria part, it took me a whole day on how should I design the criteria class to have the rest of the required commands executed successfully as the "search" and "rSearch" parts need the criteria to have real functions. Moreover, I originally thought that the tests would not be as time-consuming as creating the main code, but the tests actually require nearly the same amount of time to create the main code. Overall, this project helped me a lot in learning more about Java and object-oriented programming.

Lee Yin Lam Elaine:
During the development of our project, I have encountered quite a lot opportunities to learn about new methods and java classes which were not covered in lectures. At first, I felt frustrated and was puzzled about if there was even a solution for the issue. However, through dozens of trial and error, I learnt that every error I encountered is a crucial step to achieve success. I have learnt to refer to different sources on the Internet and discuss with my peers in order to achieve a better solution. From now on, I would need to do more coding exercises, actively engage in peer discussions so that I could become a better developer.