# Project Report

COMP2021 Object-Oriented Programming (Fall 2024)

Group 4

Members and contribution percentages:

Siu Yau Shing: 60%

Chow Kwan Ho: 40%

1 Introduction

This document describes the design and implementation of the Comp Virtual File System (CVFS)

by group 4. 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.

The Directory of the CVFS is as follows:

cvfs/

├── Application.java

├── controller/

│ ├── Command.java

│ ├── CommandProcessor.java

│ ├── NewDirCommand.java

│ ├── NewDocCommand.java

│ ├── ChangeDirCommand.java

│ ├── DeleteCommand.java

│ ├── NewNegationCommand.java

│ ├── NewSimpleCriCommand.java

│ ├── RenameCommand.java

├── exceptions/

│ ├── DiskSpaceException.java

│ ├── InvalidCommandException.java

│ ├── InvalidFileNameException.java

├── model/

│ ├── VirtualDisk.java

│ ├── Document.java

│ ├── Directory.java

│ ├── Criterion.java

│ ├── SimpleCriterion.java

│ ├── CompositeCriterion.java

└── view/

├── CLIView.java

The Design of the CVFS follows the Model-View-Control Architecture (MVC) and the three parts are shown here:

Model:

* Represents the objects, logic and data of the application
* Includes classes like VirtualDisk, Document, Directory and Criterion
* Handles core functionality and data management of the VFS

View:

* CLIView handles the user interface using the console
* Enters user input and shows if errors are outputted during the use of the CFVS.
* Also Displays other information

Controller:

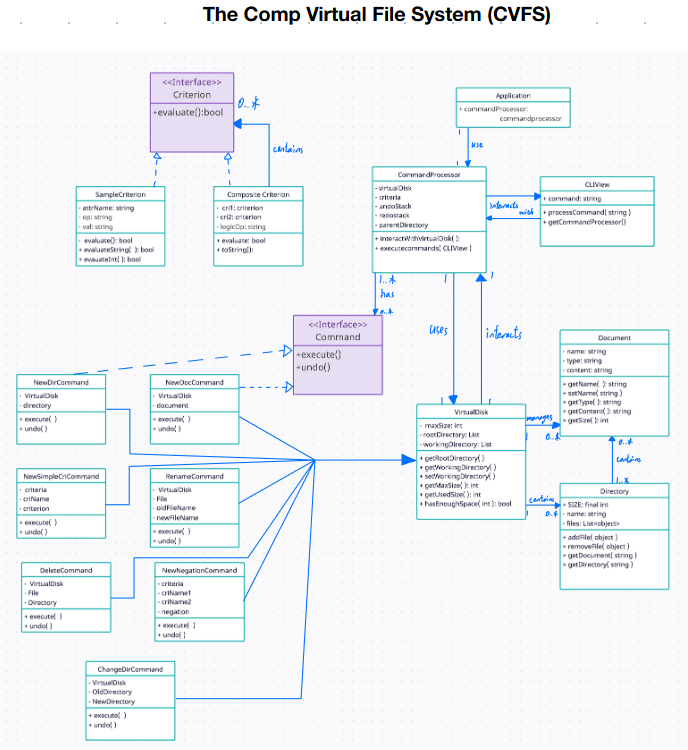
* acts as an intermediary between the Model and the View Folders
* Processes the commands from CLI View and sends them to the Model and updates them. The CommandProcessor and Commands belong to this section
* The Model interprets the user input and performs the commands on the Model

2.1 Design

Use class diagram(s) to give an overview of the system design and explain how different components

fit together. Feel free to elaborate on the used design patterns and/or anything else that might help

others understand your design.



The commands which use the interface Command are used to implement undo and redo function and their respective commands as well.

Switch are used in CLIview to access Commands from the processor

Application is used to create a new instance of CLIview when run

The application is a model view control application where CLIview is view, CommandProcessor is control, and model is virtual disk, document, directory and the criteria

Simple Criterion and Composite Criterion user criterion interface

2.2 Implementation of Requirements

For each (compulsory and bonus) requirement, describe 1) whether it is implemented and, when yes,

2) how you implemented the requirement as well as 3) how you handled various error conditions.

A file called CommandProcessor is implemented for the logic of using the commands below , with another file called CLIView used to retrieve commands.

**REQ1**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Created a new class called VirtualDisk with an attribute for Max Size.
   * Created another class called Directory with attributes for size (initial size), name, and a list for files.
   * VirtualDisk has a root directory and a working directory, both with getters and setters for initialization and retrieval.
3. **Error Handling:**
   * Set up tests for:
     + Disk size setup
     + Invalid commands
     + Space availability
     + Initializing the root directory as the working directory
     + Retrieving remaining size of the virtual disk

**REQ2**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Created a new class called Document with attributes for size (initial size), name, type, and content.
   * Added a function in Directory called addFile to add files to a directory.
3. **Error Handling:**
   * Tested initialization of a new document and retrieval of its size.
   * Validated document name.
   * Implemented error handling for:
     + Insufficient disk space
     + No disk loaded

**REQ3**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Initialized Directory with attributes for initial size, name, and a list, including getters and setters.
3. **Error Handling:**
   * Implemented checks for:
     + Valid directory name
     + Insufficient disk space
     + No disk loaded

**REQ4**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Added a function to delete a file from the directory.
3. **Error Handling:**
   * Managed errors during file addition and removal.
   * Handled cases where no file matches the criteria.

**REQ5**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Renaming a document is done using the name setter.
3. **Error Handling:**
   * Tested renaming functionality and validated name and command.

**REQ6**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Implemented directory changing in VirtualDisk using a setter.
   * Added functionality for navigating to the parent directory using "..".
3. **Error Handling:**
   * Handled errors for:
     + No disk loaded
     + Directory not found

**REQ7**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Lists all files and directories in the working directory, showing their type and size.
3. **Error Handling:**
   * Handled error for no disk loaded.

**REQ8**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Used recursion to list files and directories, including their type and size.
   * Utilized a string builder with an indent function for formatting.
3. **Error Handling:**
   * Handled error for no disk loaded.

**REQ9**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Developed a Criterion interface.
   * Created SimpleCriterion class with attributes for name, operator, and value, including validation.
3. **Error Handling:**
   * Validation for character size and attributes.

**REQ10**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Implemented search functionality for documents within the disk.
3. **Error Handling:**
   * Handled errors for:
     + No disk loaded
     + Document not found

**REQ11**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Accessed pre-made criteria for operations.
3. **Error Handling:**
   * Handled errors when one or two criteria cannot be found.

**REQ12**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Added a set to store all criteria.
3. **Error Handling:**
   * Handled errors when no criteria are stored.

**REQ13**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Search function initializes file count and size to zero.
   * Uses the directory array to accumulate files and subdirectory sizes.
3. **Error Handling:**
   * Handled errors for:
     + No disk loaded
     + Duplicates

**REQ14**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Conducted a recursive search on the working directory, tracking files and sizes, with a final sum.
3. **Error Handling:**
   * Handled error for no disk loaded.

**REQ15**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Saved disk alongside its files to a specified path.
3. **Error Handling:**
   * Handled error for no disk loaded.

**REQ16**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Loaded disk from the specified path.
3. **Error Handling:**
   * Handled errors for:
     + Path not found
     + Class not found

**REQ17**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Quits the application using System.exit.
3. **Error Handling:**
   * Confirmed functionality works as intended.

**BONUS 1**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Saved the criteria alongside the disk.
3. **Error Handling:**
   * Implemented alongside REQ 15 and 16.

**BONUS 2**

1. **Requirement:** Implemented.
2. **Implementation Details:**
   * Utilized a stack to manage commands for undo and redo functionality.
3. **Error Handling:**
   * Tested to ensure functionality works correctly.

## Reflection on learning-to-learn experience

Throughout this project, we have learnt a lot about how applications are built, particularly its architecture, debugging and test coverage. Learning about these concepts have been rewarding as we have learnt a lot about how to create specific errors handling for our use case and learn how to use a while loop to make a continuous application for the use of repeating user input. Without using learning-to-learn, we would not have learnt how to use these concepts in making the CVFS and how Software Engineers use this in Mac, Linux and Windows to create a file system. We have also learnt a lot about how to use java documentation to learn about how to implement features that we want in our client and also use data structures such as stack to implement an undo and redo feature.

## Plan to improve your self-learning experience

To improve our self-learning experience in the future, we would do more project- based learning in order to learn more concepts about programming and its best practices, creating our own projects to create a situation where new knowledge about how to implement our concepts into code can be done. We would also learn from looking at how other people create projects to learn about best practices in the industry and learn the logic between each object and how they relate to each other.

## Use of GenAI

GenAi has been used to help us plan and draft out the directory for the MVC system, alongside debugging and clarification of requirements. It also gave ideas on how to proceed when we were stuck. Also helped us format the code.