**Project Report COMP2021**

**Object-Oriented Programming (Nov 2024) Group 32   
  
Members and contribution percentages:**

**Chun Yip LEUNG: 40%   
Wing Ho LEUNG: 30%   
Siu Man LIU: 30%   
Tsz Fung LIU: 0%  
  
1. Introduction**

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

**2. The Comp Virtual File System (CVFS)**

**2.1 Design**

The CVFS is designed to manage virtual disks and their associated documents and directories. The system utilizes object-oriented principles and design patterns, such as the Composite pattern for managing directories and documents, and the Command pattern for encapsulating operations like creating, deleting, and searching. Below is a class diagram illustrating the relationships between key components:

* **CVFS**: Main class handling operations.
* **VirtualDisk**: Represents a virtual disk with a root directory.
* **Directory**: Represents directories within the disk.
* **Document**: Represents files stored in the directories.
* **Criterion**: Represents search criteria for documents.

**2.2 Implementation of Requirements**

**[REQ1] Create a New Disk**

1. **Implementation**: The newDisk(int size) method creates a new virtual disk of specified size and sets it as the current working disk.
2. **Error Handling**: If a disk is already created, it automatically closes the previous disk.

**[REQ2] Create a New Document**

1. **Implementation**: The newDoc(String name, String type, String content) method creates a new document in the current working directory.
2. **Error Handling**: Checks for duplicate names (not implemented in the code).

**[REQ3] Create a New Directory**

1. **Implementation**: The newDir (String name) method allows the creation of a new directory in the current working directory.
2. **Error Handling**: Verifies if the disk is initialized before creating a directory.

**[REQ4] Delete an Existing File**

1. **Implementation**: The delete (String name) method deletes a document or directory by name from the current working directory.
2. **Error Handling**: Prints a message if the document or directory is not found.

**[REQ5] Rename an Existing File**

1. **Implementation**: The rename (String oldName, String newName) method renames an existing file in the current directory.
2. **Error Handling**: Checks if the old file exists before renaming.

**[REQ6] Change Working Directory**

1. **Implementation**: The changeDir (String name) method changes the current working directory.
2. **Error Handling**: Handles cases for parent directory navigation and non-existent directories.

**[REQ7] List Files in Working Directory**

1. **Implementation**: The listFiles () method lists all files and directories within the current working directory.
2. **Error Handling**: Checks if a disk is created before listing.

**[REQ8] Recursively List Files**

1. **Implementation**: The rList () and rListHelper() methods provide recursive listing of all files in the current directory and its subdirectories.
2. **Error Handling**: Similar checks as in REQ7.

**[REQ9] Construct Simple Criteria**

1. **Implementation**: The newSimpleCri(String criName, String attrName, String operation, String value) method constructs a new simple criterion.
2. **Error Handling**: Validates criterion names and operations.

**[REQ10] Check if a File is a Document**

1. **Implementation**: The isDocument checks if a document meets the criteria.
2. **Error Handling**: Returns false for non-existing documents.

**[REQ11] Construct Composite Criteria**

1. **Implementation**: The newNegation (String criName1, String criName2) and newBinaryCri (String criName1, String criName3, String logicOp, String criName4) methods allow for the creation of negated and binary composite criteria.
2. **Error Handling**: Validates existing criteria before creating new ones.

**[REQ12] Print All Defined Criteria**

1. **Implementation**: The printAllCriteria () method prints all defined criteria in a readable format.
2. **Error Handling**: None needed.

**[REQ13] Search Files Based on Criteria**

1. **Implementation**: The search (String criName) method lists files that satisfy the given criterion.
2. **Error Handling**: Reports if the criterion is not recognized.

**[REQ14] Recursively Search Files Based on Criteria**

1. **Implementation**: The rSearch (String criName) method recursively searches for files based on the provided criterion.
2. **Error Handling**: Similar to REQ13.

**[REQ15] Save the Working Virtual Disk**

1. **Implementation**: The save(String filePath) method saves the current disk state to a file.
2. **Error Handling**: Catches IO exceptions during the save process.

**[REQ16] Load a Virtual Disk**

1. **Implementation**: The load(String filePath) method loads a virtual disk from a specified file.
2. **Error Handling**: Handles exceptions during loading.

**[REQ17] Terminate Execution**

1. **Implementation**: A simple command to terminate the execution of the system.
2. **Error Handling**: None needed.

**Bonus Features**

**[BON1] Saving and Loading Search Criteria**

1. **Implementation**: The criteria are saved alongside the virtual disk state when the disk is saved, and loaded when the disk is loaded.
2. **Error Handling**: Exceptions during file operations are handled gracefully.

**[BON2] Undo and Redo Functionality**

1. **Implementation**: The system maintains a history for undoing and redoing specific commands such as newDoc, newDir, delete, etc.
2. **Error Handling**: Ensures the history stack does not exceed a certain limit, and provides feedback when undo/redo actions cannot be performed.

**3. Reflection on My Learning-to-Learn Experience**

At first, we were quite daunted by the intricate requirements, especially those related to advanced OOP concepts like design patterns, criteria evaluation, and recursive algorithms. To address this, we established clear learning goals, pinpointing specific areas where we needed to enhance our knowledge. We turned to various online resources, such as tutorials, documentation, and coding forums, to deepen our understanding.

We immersed ourselves in hands-on coding, which enabled us to put theoretical concepts into practice. For example, while implementing composite criteria and conducting recursive file searches, we experimented with various methods and progressively debugged our code. This experience not only reinforced our grasp of the material but also underscored the value of persistence in solving problems.

Looking ahead, we plan to enhance our time management skills by dedicating regular time slots for learning and coding. This routine will help us remain committed to our self-directed learning journey and ensure consistent progress.

By adopting these strategies, we aspire to become more effective self-learners, equipped to confront complex programming challenges with confidence and resilience.