Real-Time Log Viewer

1. Introduction

This project is a **Real-Time Log Viewer** application designed to display and manage logs from multiple files in real-time. It provides users with the ability to view logs as they are updated, search through the logs with custom regular expressions, and highlight specific keywords for easier identification. The application also offers features for capturing logs into a file and filtering log data based on user-defined criteria.

2. Key Features

- **Real-Time Log Display**: The application reads and displays logs from one or more log files in real-time.
- **Keyword Highlighting**: Supports highlighting of keywords in the logs using regular expressions.
- **File Watching**: Monitors log files for changes and updates the display accordingly.
- **Log Filtering**: Allows users to filter logs based on user-defined regular expressions.
- **Multiple Log Files**: Displays logs from multiple files side by side.
- **Log Capturing**: Users can capture and view logs from selected files.

3. Motivation

The motivation behind developing this project was to create a tool that simplifies the task of monitoring and analyzing log files in real-time. Logs are crucial for understanding the behavior of systems and applications, but manually sifting through them can be time-consuming and tedious. By creating this log viewer, we aimed to enhance productivity by providing users with an easy-to-use interface for filtering and highlighting key information in logs. The tool is designed to meet the needs of developers, system administrators, and anyone who needs to monitor log files regularly.

4. Role of Every Team Member

- **Shrujal Tailor**: Shrujal was responsible for the **coding part** of the project, including implementing the core functionality of the log viewer, file watching, regex-based filtering, and log capturing.
- **Lokesh Kumar Thalluru:** Lokesh Kumar handled the **documentation part**, providing comprehensive documentation for the project, including **creation of the diagrams**, like the UML class diagram and other visual aids to explain the architecture of the project.
- **Gavin Yentis**: Gavin provided significant help in the both the parts, coding and documentation.

5. Detailed Analysis of the Project

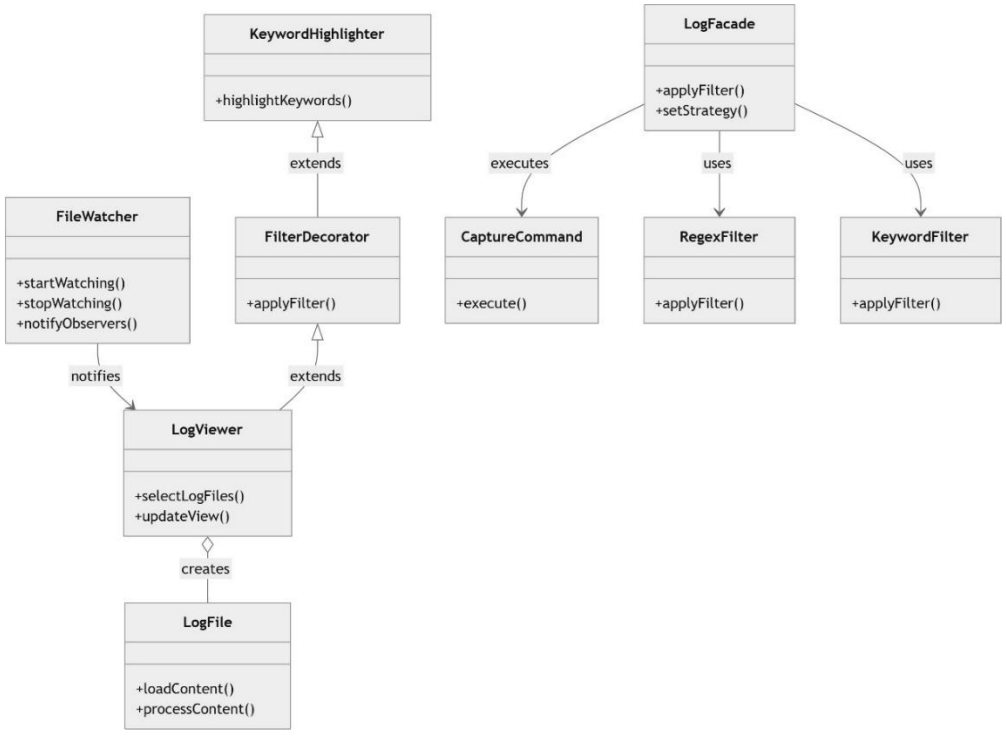
Identification of Classes and Relationships Between Classes

The project consists of the following main classes, each responsible for a specific part of the functionality:

1. **CaptureCommand**:
    - Purpose: To capture logs into a file by writing the log text to log.txt.
    - Key Methods: execute()
    - Relationship: This class interacts with file IO operations and serves as a command to capture logs.
2. **FileWatcher**:
    - Purpose: Monitors the specified log files for changes and notifies observers when the files are updated.
    - Key Methods: start()
    - Relationship: Works with LogViewer to notify updates when logs change.
3. **FilterDecorator**:
    - Purpose: A decorator class that applies a regular expression filter to the log viewer and highlights matched keywords.
    - Key Methods: highlightMatches()
    - Relationship: This class decorates the LogViewer by enhancing its functionality with filtering and highlighting.

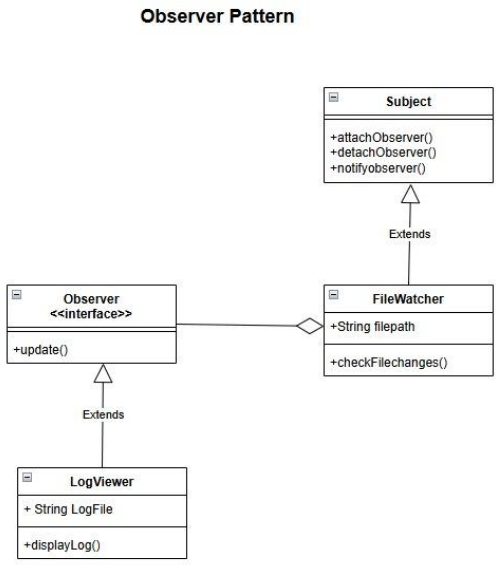
4. **LogFacade**:
    - Purpose: Provides an abstraction for reading and filtering logs using a regular expression.
    - Key Methods: start()
    - Relationship: Acts as an intermediary between FileWatcher and FilterDecorator, managing how log data is filtered.
5. **LogViewer**:
    - Purpose: The main UI class for displaying logs, allowing users to interact with the logs, apply regex filters, and select log files.
    - Key Methods: selectLogFiles(), captureLogs(), applyRegex(), onFileUpdated(), pollLogFiles()
    - Relationship: It coordinates with other components, including FileWatcher and FilterDecorator, to manage the display and filtering of logs.

UML Class Diagram

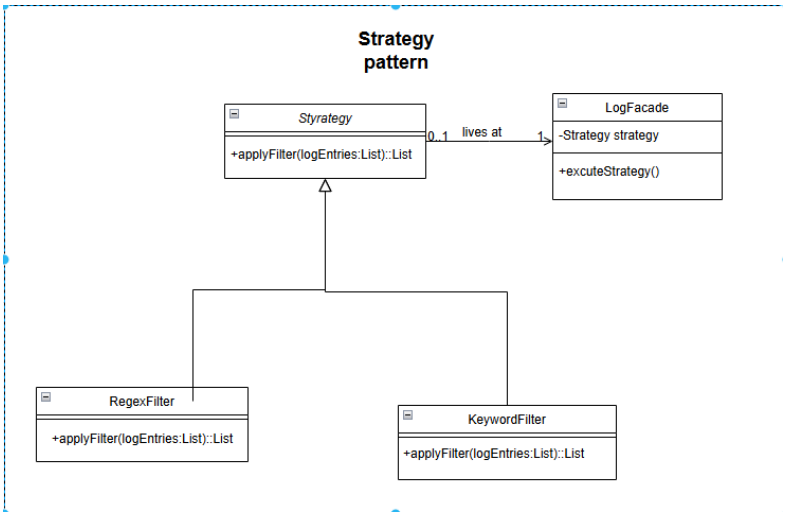


6. Use of Design Patterns and How They Solve the Original Problem

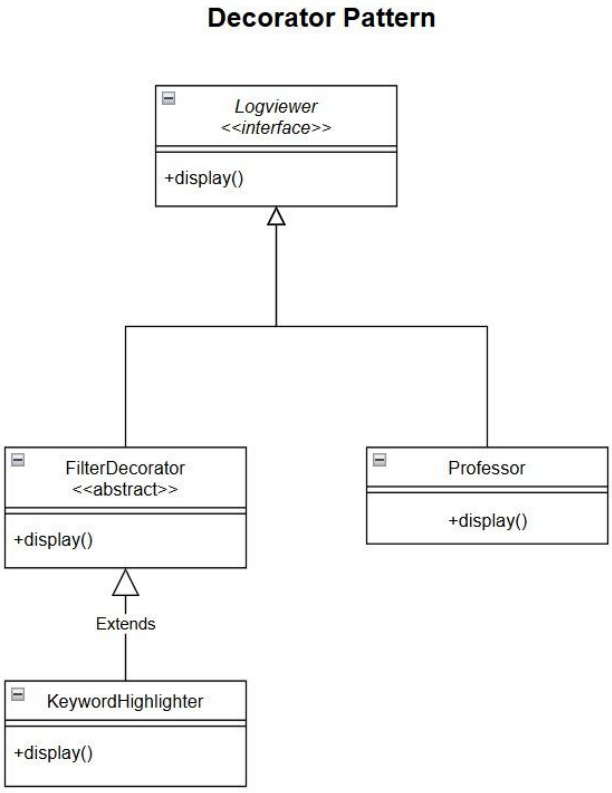
1. Observer Pattern



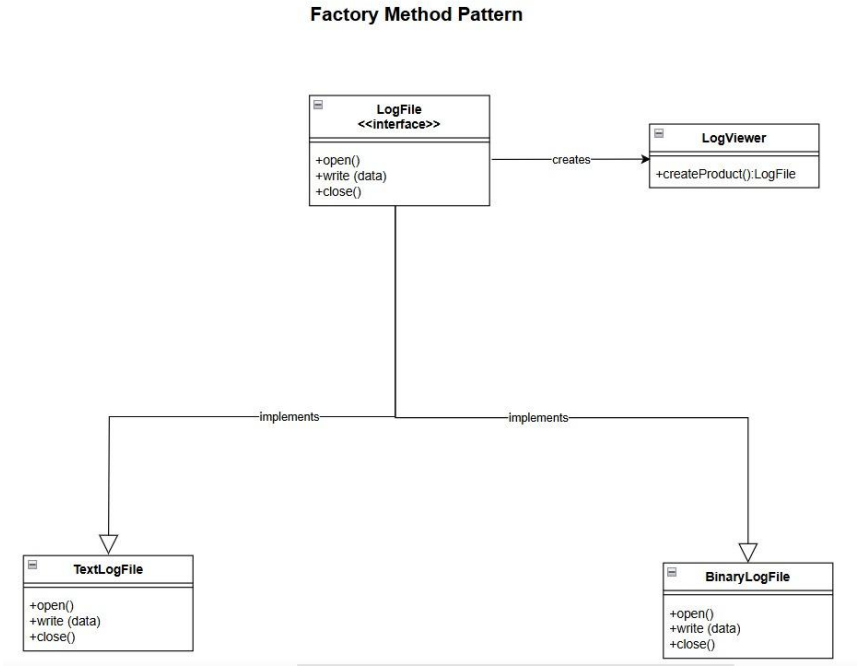
2. Strategy Pattern



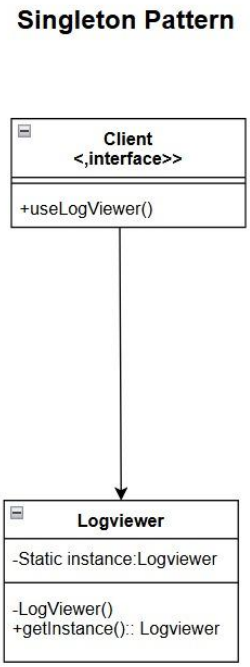
3. Decorator Pattern



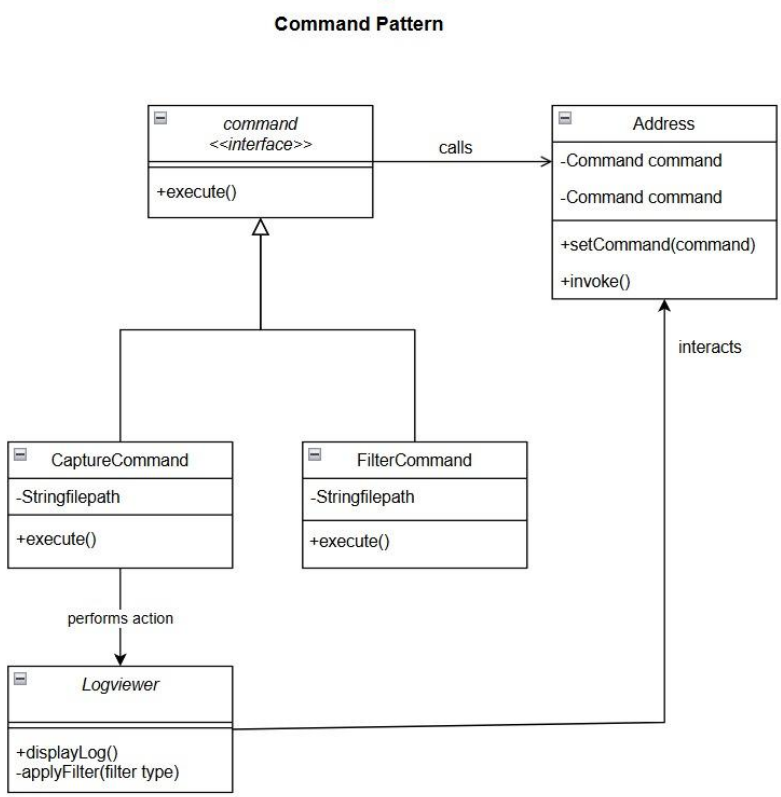
4. Factory Method Pattern



5. Singleton Pattern



6. Command Pattern



Video Link: **Demo Video**

Code Link: GitHub Link