Lab1 架构设计文档

姓名: 汪筠 **学号**: 23300120041 **日期**: 2025-11-22

2.1 系统架构

模块划分

本系统基于分层架构设计，代码组织在 `lab1` 包下，主要包含以下四个核心模块：

1. **Model 层 (`lab1.model`)**
    - **职责**: 负责系统的核心数据结构与业务逻辑。
    - **核心类**:
        - `TextEditor`: 数据实体。维护行文本列表 (`List<String>`)，提供底层的 `insert`、`delete`、`append` 等原子操作。
        - `Workspace`: 上下文管理。负责管理多个编辑器实例，维护当前活动文件（Active Editor），处理文件的加载、保存、关闭，以及工作区状态的持久化 (`.editor_workspace`)。
2. **Command 层 (`lab1.command`)**
    - **职责**: 实现命令模式，封装具体的编辑行为，支持撤销与重做。
    - **核心类**:
        - `Command` (接口): 定义统一的 `execute()` 和 `undo()` 标准。
        - `InsertCommand`/`DeleteCommand`/`AppendCommand`: 具体命令实现。每个命令类内部保存了执行前的状态（如被删除的文本片段），以支持精确回滚。
        - `CommandHistory`: 历史记录管理器。维护 `undoStack` 和 `redoStack`，解耦了命令的发起者与执行者。
3. **Utils 层 (`lab1.utils`)**
    - **职责**: 提供通用的辅助功能。
    - **核心类**:
        - `Logger`: 日志工具。负责格式化时间戳，并将操作记录追加写入到 `.log` 文件中。
4. **App 层 (`lab1.app`)**
    - **职责**: 程序入口与交互层。
    - **核心类**:
        - `Main`: 负责解析命令行参数 (CLI)，根据用户输入调度 Workspace 和 Command 模块。

模块依赖关系

- `App` 层作为顶层协调者，依赖 `Model`、`Command` 和 `Utils`。
- `Command` 层依赖 `Model` 层（需要操作具体的 `TextEditor` 对象）。
- `Model` 层作为底层核心，不依赖其他业务模块。

2.2 核心设计

本实验重点应用了以下设计模式以满足功能需求：

1. **命令模式 (Command Pattern)**
    - **应用场景**: 所有的文本编辑操作（插入、删除、追加）。
    - [cite_start]**设计理由**: 作业要求支持无限层级的 Undo/Redo [cite: 20]。通过将请求封装为对象，我们可以轻松地将命令压入栈中，并在需要时弹出执行 `undo()` 方法，无需在业务逻辑中编写复杂的 switch-case 回退逻辑。
2. **备忘录模式 (Memento Pattern) 的应用**
    - [cite_start]**应用场景**: 工作区状态持久化 [cite: 28]。
    - **设计理由**: 程序退出时，`Workspace` 会将当前的“状态快照”（打开的文件列表、修改标记、日志开关）序列化保存到 `.editor_workspace` 文件。下次启动时，通过读取该文件恢复现场，实现了会话的连续性。
3. **观察者模式 (Observer Pattern) 的简化应用**
    - [cite_start]**应用场景**: 操作日志记录 [cite: 29]。
    - **设计理由**: 在 `Main` 的主循环中监听命令执行事件。一旦检测到编辑命令且日志开关 (`logEnabled`) 为真，即触发 Logger 记录。这种设计将“记录日志”与“执行编辑”解耦，符合单一职责原则。

2.3 运行说明

环境依赖

- **JDK 版本**: Java 17 或更高
- **构建工具**: Maven
- **依赖库**: JUnit 5.10.0 (用于自动化测试)

运行步骤

1. **编译**: 在项目根目录下，使用 IDEA 导入 Maven 项目并等待依赖下载完成。
2. **启动**: 运行 `src/main/java/lab1/app/Main.java`。
3. **交互**: 程序启动后进入命令行模式。
    - 输入 `help` 查看所有可用命令。
    - 示例：输入 `init test.txt` 初始化一个新文件。

2.4 测试文档

自动化测试策略

本项目采用 JUnit 5 对核心模块进行了单元测试，测试代码位于 `src/test/java/lab1/TextEditorTest.java`。

测试用例与结果

测试方法	测试内容	预期结果	测试结果
`testInsert`	在指定行列插入文本	文本内容正确变更	☑ 通过

测试方法	测试内容	预期结果	测试结果
`testDelete`	删除指定长度的字符	字符被移除，后续字符前移	☑ 通过
`testInsertUndo`	执行插入后立即撤销	文本恢复到插入前状态	☑ 通过
`testDeleteUndo`	执行删除后立即撤销	被删字符恢复，位置正确	☑ 通过
`testAppendUndo`	追加行后撤销	最后一行被移除	☑ 通过

手动测试覆盖

除了单元测试，还对以下功能进行了手动验证：

- **文件 I/O**: `load`/`save` 功能正常，支持 UTF-8 编码。
- **日志功能**: `log-on` 后，操作能正确写入 `.filename.log` 文件。
- **状态保持**: `exit` 后重启，能通过 `.editor_workspace` 恢复环境（需配合代码实现）。