一、系统架构

1.1 模块划分图

```
com.editor
├── App.java                    // 程序入口
├── command/                    // 命令模块
│   ├── Command.java
│   ├── CommandParser.java
│   └── ......                  // 具体命令实现
├── editor/                     // 编辑器模块
│   └── TextEditor.java
├── logging/                    // 日志模块
│   └── Logger.java
├── memento/                    // 备忘录模式
│   └── Memento.java
├── observer/                   // 观察者模式
│   ├── Event.java
│   ├── EventType.java
│   └── Observer.java
└── workspace/                  // 工作区模块
    └── Workspace.java
```

1.2 模块职责说明

模块	职责描述
App	程序入口，初始化工作区、日志和命令解析器，处理用户输入循环
command	命令解析与执行，包含所有具体命令的实现
editor	文本编辑功能实现，支持基本编辑操作和撤销重做
logging	日志记录功能，支持日志开关、查看和持久化
memento	实现工作区状态的保存与恢复（备忘录模式）
observer	事件通知机制，支持日志模块监听命令执行
workspace	管理所有打开的文件、当前活动文件、日志开关状态

1.3 模块依赖关系

```
App → command, workspace, logging
command → workspace, editor, logging
workspace → editor, memento, observer
logging → observer
editor → memento (用于撤销重做状态保存)
```

二、核心设计

2.1 设计模式应用说明

（1）命令模式（Command Pattern）

- **位置**: `command/` 包
- **实现**:
  - `Command` 抽象类定义 `execute()`，`undo()`，`redo()` 方法
  - 每个具体命令（如 `AppendCommand`，`InsertCommand`）继承 `Command`
  - `CommandParser` 负责解析输入并创建对应命令对象
- **优势**: 支持撤销重做，易于扩展新命令

（2）备忘录模式（Memento Pattern）

- **位置**: `memento/` 包
- **实现**:
  - `WorkspaceMemento` 保存工作区状态（打开文件列表、活动文件、修改状态）
  - `Workspace` 提供 `createMemento()` 和 `restoreFromMemento()` 方法
  - 程序退出时保存到 `.editor_workspace`，启动时恢复
- **优势**: 实现状态持久化，支持会话恢复

（3）观察者模式（Observer Pattern）

- **位置**: `observer/` 包
- **实现**:
  - `Observer` 接口定义 `update()` 方法
  - `Logger` 实现 `Observer` 接口，监听命令执行事件
  - `Workspace` 作为被观察者，维护观察者列表并通知事件

- **优势**: 解耦日志记录与命令执行，支持事件驱动架构

（4）装饰器模式（Decorator Pattern）

- **位置**: `editor/` 包（预留扩展）
- **实现**: 可用于扩展 `Editor` 功能，如日志装饰器、只读装饰器等

2.2 其他设计说明

- **文本存储结构**: 使用 `List<String>` 存储文本，每行为一个元素
- **错误处理**: 所有命令均进行参数校验和异常捕获，提供友好提示
- **日志自动开启**: 若文件首行为 `# log`，则自动启用日志记录
- **状态管理**: 每个 `Editor` 维护独立的修改状态和撤销栈
- **文件编码**: 统一使用 UTF-8 编码

三、运行说明

3.1 编程语言及版本

- **语言**: Java
- **版本**: JDK 8 或以上

3.2 安装依赖步骤

本项目为纯 Java 实现，无第三方依赖。

四、测试文档

4.1 测试用例列表

测试类	测试方法	说明
`TextEditorTest`	`testAppend`	测试追加文本功能
`TextEditorTest`	`testInsert`	测试插入文本功能

测试类	测试方法	说明
TextEditorTest	testInsertWithNewline	测试插入含换行符的文本
TextEditorTest	testDelete	测试删除文本功能
TextEditorTest	testReplace	测试替换文本功能
TextEditorTest	testShow	测试显示文本功能
TextEditorTest	testUndoRedo	测试撤销重做功能
WorkspaceTest	testLoadSaveClose	测试文件加载、保存、关闭
WorkspaceTest	testSwitchEditor	测试切换活动文件
WorkspaceTest	testEditorList	测试文件列表显示
WorkspaceTest	testMemento	测试状态保存恢复
LoggerTest	testLogOnOff	测试日志开关
LoggerTest	testLogShow	测试日志显示
LoggerTest	testAutoLog	测试自动日志开启
CommandParserTest	testParseAndExecute	测试命令解析与执行
CommandParserTest	testInvalidCommand	测试无效命令处理

4.2 测试执行结果

所有测试用例通过，覆盖以下功能：

- ✅ **工作区命令** (10个): load, save, init, close, edit, editor-list, dir-tree, undo, redo, exit
- ✅ **文本编辑命令** (5个): append, insert, delete, replace, show
- ✅ **日志命令** (3个): log-on, log-off, log-show
- ✅ **核心机制**: 撤销重做、状态持久化、事件通知、错误处理

五、功能验证清单

5.1 工作区命令验证

- ✅ `load <file>` - 加载文件，自动检测日志标记
- ✅ `save [file|all]` - 保存文件，支持指定文件或全部保存
- ✅ `init <file> [with-log]` - 创建新缓冲区，可选日志标记
- ✅ `close [file]` - 关闭文件，提示未保存修改
- ✅ `edit <file>` - 切换活动文件
- ✅ `editor-list` - 显示文件列表和状态
- ✅ `dir-tree [path]` - 显示目录树结构
- ✅ `undo / redo` - 撤销重做操作
- ✅ `exit` - 退出程序，保存工作区状态

5.2 文本编辑命令验证

- ✅ `append "text"` - 在文件末尾追加文本
- ✅ `insert <line:col> "text"` - 在指定位置插入文本
- ✅ `delete <line:col> <len>` - 删除指定长度字符
- ✅ `replace <line:col> <len> "text"` - 替换指定文本
- ✅ `show [start:end]` - 显示文本内容

5.3 日志命令验证

- ✅ `log-on [file]` - 启用日志记录
- ✅ `log-off [file]` - 关闭日志记录
- ✅ `log-show [file]` - 显示日志内容

六、总结

本实验基于模块化架构，实现了一个功能完整的命令行文本编辑器。系统采用清晰的分层设计，合理应用了命令模式、备忘录模式和观察者模式，实现了所有18个必需命

令，具备完整的编辑、撤销重做、日志记录和状态持久化功能。