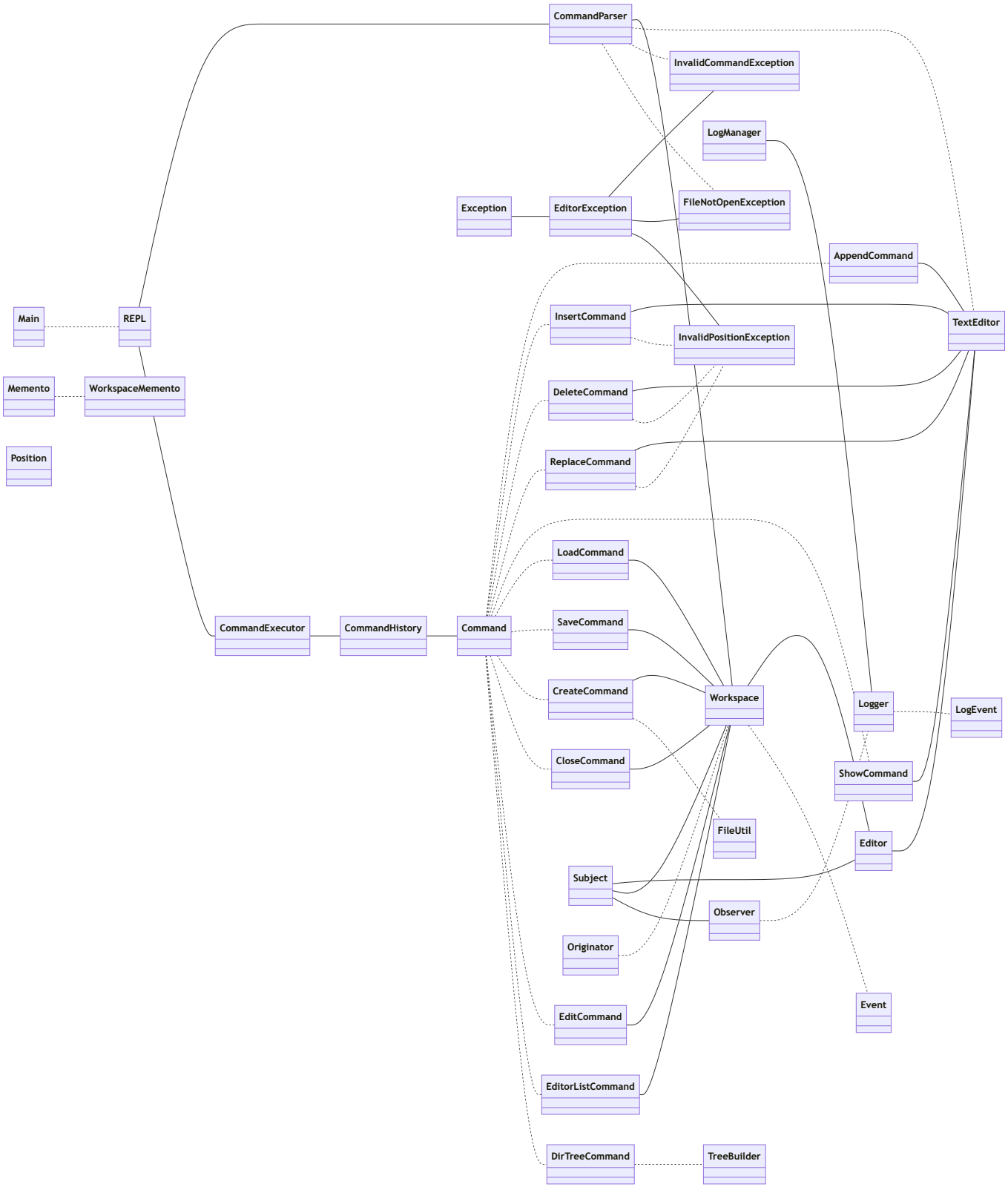
架构设计文档

系统架构

1. 模块划分图

该代码使用三层体系结构。从**App入口**进入命令行输入命令。输入命令后，在**CLI** 模块调用命令行Parser对不同命令进行判断。若出现不合法的输入，则返回"help"。如果出现IO等错误，则调用**Exception**模块。同时配置增加程序鲁棒性的**Utils**模块和记录日志的**Logging**模块。



2. 模块职责说明

1. `App`： 程序的总入口，用于进入程序。
2. `CLI`:提供和用户的命令行交互界面，传递并分类用户指令。

3. `Exception`:对各种异常进行处理，统一置于一个包中。
4. `Utilities`: 包含文件夹树形图生成、标准化文件位置等不同工具
5. `Patterns`: 包含Momento模式的实例`WorkspaceMemento`和Observer模式的实例`Observer`
6. `Logging`:生成记录日志的类
7. `Core`: 核心领域模型与运行时管理层，负责文件与编辑器实例的生命周期、当前活动文件、最近文件等状态管理

3. 模块依赖关系

UML 关系汇总表**

- 泛化(继承) Generalization `<|--`
  - 示例
    - `Editor` 继承 `Subject`
    - `Workspace` 继承 `Subject`
    - `TextEditor` 继承 `Editor`
    - 异常体系：`EditorException` 继承 `Exception`；`InvalidCommandException` / `FileNotOpenException` / `InvalidPositionException` 继承 `EditorException`
  - 要点：空心三角指向父类，表现稳定的 IS-A 继承
- 实现 Realization `<|..`
  - 示例
    - `Logger` 实现 `Observer`
    - `Workspace` 实现 `Originator<WorkspaceMemento>`
    - `WorkspaceMemento` 实现 `Memento`
    - 各具体命令实现 `Command`：如 `AppendCommand` 、 `LoadCommand` 等
  - 要点：虚线三角连接接口，具体类提供行为
- 组合 Composition `*--`
  - 示例
    - `Workspace` 强拥有多个 `Editor`： 构造与加载中创建并加入列表
    - `CommandExecutor` 强拥有 `CommandHistory`
    - `REPL` 强拥有 `CommandParser`、 `CommandExecutor`
    - `LogManager` 强拥有默认 `Logger`

    - 要点：实心菱形在整体端，生命周期绑定，同生共死
- 聚合 Aggregation `o--`
    - 示例
        - `Subject` 聚合 `Observer` 列表
        - `CommandHistory` 聚合 `Command` 栈
    - 要点：空心菱形在整体端，弱拥有，易聚散
- 关联 Association `-->`
    - 示例
        - `CommandParser` 持有 `Workspace`
        - 文本命令持有 `TextEditor`：如 `AppendCommand`
        - 工作区命令持有 `Workspace`：如 `SaveCommand`
- 依赖 Dependency `..>`
    - 示例
        - `Workspace` 依赖事件对象：`notifyObservers(new Event(...))``Logger` 依赖 `LogEvent`：方法内构造写日志，
        - `CreateCommand` 依赖工具类 `FileUtil`
        - `DirTreeCommand` 依赖 `TreeBuilder`
        - `Insert/Replace/DeleteCommand` 依赖 `InvalidPositionException`
    - 要点：临时调用或方法参数级使用，非长期持有

核心设计

1. 设计模式说明

本项目使用了四个设计模式: `Command` 模式、 `Observer` 模式、 `Momento` 模式、 `Singleton` 模式

Command 命令模式

- 目的：将操作封装为对象，支持统一执行、撤销、重做
- 参与者
    - `Command` 接口： `execute` / `undo` / `isUndoable` / `getDescription`
    - 具体实现的命令：文本类命令（ `AppendCommand` 、 `InsertCommand` 、 `DeleteCommand` 、 `ReplaceCommand` 、 `ShowCommand` ）和工作区命令（ `LoadCommand` 、 `SaveCommand` 、

`CreateCommand`、`CloseCommand`、`EditCommand`、`EditorListCommand`、`DirTreeCommand`）
    - `CommandHistory` 撤销/重做栈
    - `CommandExecutor` 执行入口
    - `REPL` + `CommandParser` 充当命令工厂与调度
- 关键点：执行可撤销命令会入撤销栈并清空重做栈；撤销将命令移至重做栈；重做再回到撤销栈

Observer **观察者模式**

- 目的：事件生产者与消费者解耦，支持多订阅者响应
- 参与者
    - `Subject` 维护订阅与通知
    - `Observer` 接口
    - `Event` 事件容器：类型与数据字典
    - 事件源：`Workspace` / `Editor` 继承 `Subject` 观察者：`Logger` 实现 `Observer`，记录事件日志
- 关键点：`Workspace` 在加载/关闭/恢复时广播事件

Memento **备忘录模式**

- 目的：捕获并外化对象内部状态用于后续恢复，避免破坏封装
- 参与者
    - `Originator<T>`：创建/恢复备忘录接口
    - `Memento` 标记接口
    - `WorkspaceMemento` 工作区快照（打开文件、活动文件、最近文件队列）并支持 JSON 序列化
    - `Workspace` 作为 `Originator` 创建/恢复快照
- 关键点：与命令模式互补（命令处理操作序列，备忘录处理整体状态）；支持会话持久化与恢复

Singleton **单例模式**

- 目的：全局唯一访问点，集中管理日志
- 参与者
    - `LogManager` 单例持有 `Logger` 并提供静态便捷方法
- 关键点：`getInstance()` 懒汉式同步获取；默认构造内部创建 `Logger`

2. **其他说明**

本版块将阐明部分除单个设计模式外、为实现方便而做的的其他设计

- 简单工厂(Eg:CLI包)
  - 目的：集中将用户输入映射到具体命令实例
  - 参与者
    - `CommandParser` 根据命令字符串创建不同命令对象
  - 关键点：虽然未显式命名为工厂模式，但其职责与实现等同于“简单工厂”
- 抽象基类
  - 目的：抽象出编辑器共性接口，具体实现负责细节
  - 参与者,Eg:
    - `Editor` 抽象基类定义 `load` / `save` / `getContent` 等接口
    - `TextEditor` 具体实现行级文本操作

运行说明

语言及版本

语言: Java

版本：11

依赖安装步骤

- 前置条件
  - 安装并配置 `JDK 11+`（设置 `JAVA_HOME`，`PATH` 包含 `JAVA_HOME\bin`）
  - 安装并配置 `Apache Maven 3.6+`（设置 `MAVEN_HOME`，`PATH` 包含 `MAVEN_HOME\bin`）
  - 验证安装：`mvn -v`、`java -version`
- 获取项目依赖（库）
  - 在项目根目录执行：`mvn clean package`
  - Maven 会自动下载 `pom.xml` 中声明的依赖：`junit-jupiter`、`gson`、`commons-io`
- 说明
  - 若本机尚未安装 Maven，则需要先安装 Maven

程序运行步骤

- 构建可运行 JAR
  - `mvn -DskipTests clean package`

- 运行入口（REPL 交互）
  - `java -jar target/text-editor-1.0.0.jar`
- 程序入口类：`com.editor.Main`
- 启动后常用命令示例
  - `help` 查看命令列表
  - `create <filepath>` 创建并打开文件
  - `append <text>` 在末尾追加一行
  - `save` 保存当前文件
  - `quit` 退出

测试步骤

- 运行全部测试
  - `mvn test`
- 构建与测试一起运行
  - `mvn clean verify`
- 测试框架与插件
  - JUnit 5
  - Maven Surefire Plugin

测试环境配置

- 测试环境
  - Windows、JDK 11+、Maven 3.6+，项目根执行 Maven 命令
- 测试范围
  - 当前仓库提供 `TextEditor` 的单元测试，覆盖行级操作与文件读写路径
- 执行步骤
  - 在项目根：`mvn test`
  - 查看 `target/surefire-reports` 中的报告与控制台摘要
- 通过/失败判定
  - 控制台应出现类似摘要：`Tests run: 4, Failures: 0, Errors: 0, Skipped: 0`

测试用例列表

- `appendAndCount` 用例
  - 场景：追加两行并统计行数
  - 断言：行数为 2；第 1、2 行分别为 `"a"`、`"b"`；编辑器状态为已修改
- `insertAndDelete` 用例
  - 场景：在第 2 行前插入 `"z"`，然后删除第 1 行

    - 断言：插入后行序列为 `["x","z","y"]`；删除后为 `["z","y"]`
- `setAndGetLine` 用例
    - 场景：设置第 2 行内容为 `"bb"`，读取越界行
    - 断言：第 2 行为 `"bb"`；越界行返回 `null`
- `saveAndLoad` 用例
    - 场景：写入两行保存，再新建编辑器加载同文件
    - 断言：保存后 `isModified` 为 `false`；加载后的内容为 `"one \n two"`，行数为 2

测试结果