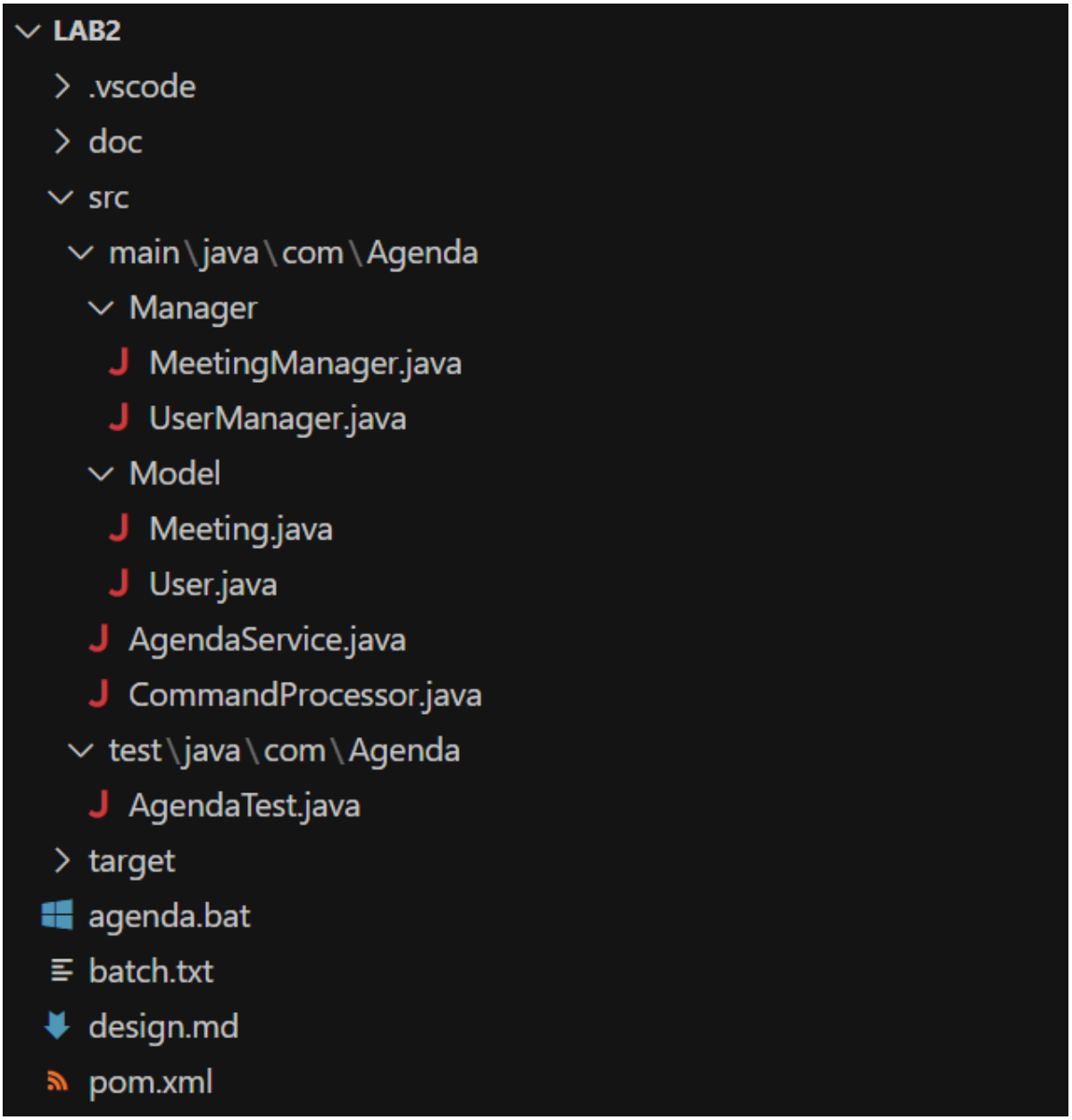
Agenda设计说明

——22336073傅小桐

一、文件结构



doc文件夹里是生成好的javadoc

src\main文件夹里是Agenda的实现。其中AgendaService是程序的入口，CommandProcessor负责处理输入的命令行，并给出反馈；Manager文件夹里是MeetingManager和UserManager，这两个类提供管理User和Meeting的接口；Model文件夹里是User和Meeting，这两个类提供了User和Meeting的基本属性。

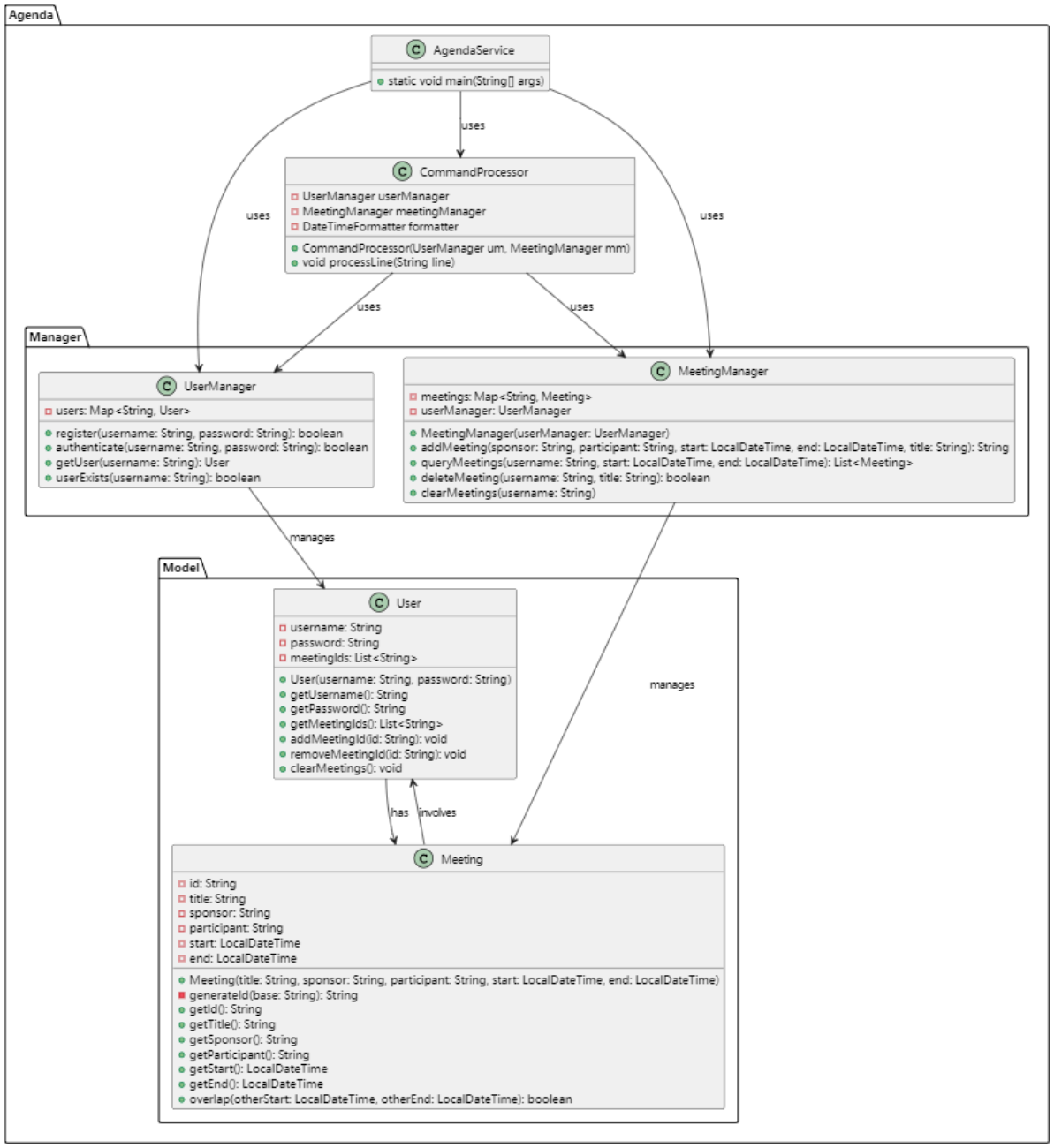
src\test文件夹是测试板块，负责测试程序运行是否正确。

agenda.bat是Agenda项目的运行脚本，双击该文件即可运行程序。

batch.txt里提前写好了若干命令，在程序内输入$batch batch.txt即可自动运行txt文件里的命令。

pom.xml规定了maven项目所需要的插件等。

二、UML类图



由UML类图可得，Agenda涉及到了六个类，分别为：

AgendaService：议程服务类，提供命令行接口供用户交互。该类负责初始化用户管理器、会议管理器和命令处理器，并启动命令行交互式服务。

CommandProcessor：命令处理器类，用于解析和执行用户输入的命令。该类支持注册、添加会议、查询会议、删除会议、清空会议等操作，并能批量处理命令文件。

MeetingManager：会议管理类，用于管理用户的会议，包括添加、查询、删除会议等功能。本类提供了对会议的添加、查询、删除等操作，并处理会议冲突的检测。

UserManager：用户管理类，用于注册、认证和管理用户信息。本类提供了注册、身份认证、查询用户等功能。

Meeting：表示一个会议，包含会议的标题、发起人、参与人、开始时间和结束时间等信息。会议通过标题和当前系统时间生成唯一的 ID 来标识。

User：表示一个用户，包含用户的用户名、密码和已参与的会议 ID 列表。用户可以添加、删除和清空其参与的会议 ID。

三、类的详解

AgendaService

```
public class AgendaService {

    /**
     * 主方法，程序入口。
     * 启动用户命令行界面，接收用户输入并通过命令处理器处理命令。
     *
     * @param args 命令行参数
     */
    public static void main(String[] args) {
        // 创建用户管理器、会议管理器和命令处理器
        UserManager userManager = new UserManager();
        MeetingManager meetingManager = new MeetingManager(userManager);
        CommandProcessor processor = new CommandProcessor(userManager,
meetingManager);
        try (Scanner scanner = new Scanner(System.in)) {
            while (true) {
                System.out.print("$ "); // 打印提示符
                String line = scanner.nextLine(); // 读取用户输入
                processor.processLine(line); // 处理输入的命令
            }
        }
    }
}
```

`AgendaService` 类是本议程管理系统的主控制模块，承担系统的启动、初始化以及命令行交互逻辑的核心职责。其设计目标是为用户提供一个简洁高效的命令行界面（CLI），以便对议程系统中的用户信息与会议数据进行交互式管理。

程序通过 `main` 方法作为执行入口，采用交互式命令行界面实现用户操作。其主要流程如下：

1. **模块初始化**：依次创建 `UserManager`、`MeetingManager` 与 `CommandProcessor` 的实例。
2. **交互式循环**：通过 Java 标准输入类 `Scanner` 实现一个循环结构，持续监听用户输入。
3. **命令执行调度**：将读取到的用户指令传递给 `CommandProcessor`，由其完成后续解析与操作执行。
4. **资源管理**：使用 `try-with-resources` 结构自动释放输入流资源，保证程序健壮性与安全性。

CommandProcessor

```
//核心函数
public void processLine(String line) {
        try {
            if (line.isEmpty())
                return;
            String[] tokens = line.trim().split(" ");
            String command = tokens[0].toLowerCase();

            switch (command) {
                case "register":
                    if (tokens.length != 3) {
```

```
                        System.out.println("[Warning] Usage: register <username>
<password>");
                        break;
                    }
                    System.out.println(userManager.register(tokens[1], tokens[2])
? "Register success."
                            : "Username already exists.");
                    break;

                case "add":
                    if (tokens.length != 7) {
                        System.out.println(
                                "[Warning] Usage: add <username> <password>
<participant> <start> <end> <title>");
                        break;
                    }
                    if (!userManager.authenticate(tokens[1], tokens[2])) {
                        System.out.println("Auth failed.");
                        break;
                    }
                    LocalDateTime s = LocalDateTime.parse(tokens[4], formatter);
                    LocalDateTime e = LocalDateTime.parse(tokens[5], formatter);
                    System.out.println(meetingManager.addMeeting(tokens[1],
tokens[3], s, e, tokens[6]));
                    break;

                case "query":
                    if (tokens.length != 5) {
                        System.out.println("[Warning] Usage: query <username>
<password> <start> <end>");
                        break;
                    }
                    if (!userManager.authenticate(tokens[1], tokens[2])) {
                        System.out.println("Auth failed.");
                        break;
                    }
                    LocalDateTime qs = LocalDateTime.parse(tokens[3], formatter);
                    LocalDateTime qe = LocalDateTime.parse(tokens[4], formatter);
                    List<Meeting> meetings =
meetingManager.queryMeetings(tokens[1], qs, qe);
                    for (Meeting m : meetings) {
                        System.out.printf("%s %s %s %s\n", m.getStart(),
m.getEnd(), m.getTitle(), m.getParticipant());
                    }
                    break;

                case "delete":
                    if (tokens.length != 4) {
                        System.out.println("[Warning] Usage: delete <username>
<password> <title>");
                        break;
                    }
                    if (!userManager.authenticate(tokens[1], tokens[2])) {
                        System.out.println("Auth failed.");
                        break;
                    }
```

```
                    System.out.println(meetingManager.deleteMeeting(tokens[1],
tokens[3]) ? "Delete success."
                            : "Meeting not found or unauthorized.");
                    break;

                case "clear":
                    if (tokens.length != 3) {
                        System.out.println("[Warning] Usage: clear <username>
<password>");
                        break;
                    }
                    if (!userManager.authenticate(tokens[1], tokens[2])) {
                        System.out.println("Auth failed.");
                        break;
                    }
                    meetingManager.clearMeetings(tokens[1]);
                    System.out.println("All meetings cleared.");
                    break;

                case "batch":
                    if (tokens.length != 2) {
                        System.out.println("[Warning] Usage: batch <filename>");
                        break;
                    }
                    try {
                        List<String> lines =
Files.readAllLines(Paths.get(tokens[1]));
                        for (String l : lines) {
                            processLine(l);
                        }
                    } catch (IOException e1) {
                        System.out.println("[Warning] Failed to read batch file:
" + e1.getMessage());
                    }
                    break;

                case "quit":
                    System.out.println("Bye.");
                    System.exit(0);
                    break;

                default:
                    System.out.println("[Warning] Unknown command.");
            }
        } catch (Exception e) {
            System.out.println("[Warning] Invalid command or error: " +
e.getMessage());
        }
    }
```

`processLine` 方法是命令行议程管理系统中的指令解析与调度核心，用于接收并处理来自用户的单条指令。该方法设计体现了面向对象中的**命令分发模型（Command Dispatch Model）**，通过结构化的字符串解析与模块化调用，实现对系统功能的动态控制。

MeetingManager

```
public class MeetingManager {

    /** 存储会议ID和会议对象的映射 */
    private final Map<String, Meeting> meetings = new HashMap<>();
    /** 用户管理器，用于获取和管理用户信息 */
    private final UserManager userManager;

    public MeetingManager(UserManager userManager) {
        ......
    }

    public String addMeeting(String sponsor, String participant, LocalDateTime
start, LocalDateTime end, String title) {
    ......
    }

    public List<Meeting> queryMeetings(String username, LocalDateTime start,
LocalDateTime end) {
        ......
    }

    public boolean deleteMeeting(String username, String title) {
        ......
    }


    public void clearMeetings(String username) {
       ......
    }
}
```

MeetingManager(UserManager) 构造方法，初始化会议管理器与用户管理器
addMeeting(...) 添加会议，进行用户验证与时间冲突检查
queryMeetings(...) 查询指定用户在某一时间段内的会议
deleteMeeting(...) 删除用户发起的某个会议（按标题）
clearMeetings(...) 清空用户发起的所有会议记录

UserManager

类似于MeetingManager：

UserManager() 构造方法，初始化用户映射表
register(String, String) 注册新用户，若用户名不存在则添加用户信息
authenticate(String, String) 验证用户名和密码是否匹配，实现身份认证

```
public boolean authenticate(String username, String password) {
        return users.containsKey(username) &&
users.get(username).getPassword().equals(password);
    }
```

getUser(String) 获取指定用户名对应的用户对象
userExists(String) 判断指定用户名是否已注册

Meeting

Meeting(String, String, String, LocalDateTime, LocalDateTime) 构造方法，初始化会议对象并生成唯一ID
generateId(String) 私有方法，根据标题和系统时间生成会议ID

```
private String generateId(String base) {
        return "MID-" + Math.abs((base + System.nanoTime()).hashCode());
    }
```

getId() 获取会议的唯一ID
getTitle() 获取会议标题
getSponsor() 获取会议发起人
getParticipant() 获取会议参与人
getStart() 获取会议开始时间
getEnd() 获取会议结束时间
overlap(LocalDateTime, LocalDateTime) 判断会议时间是否与指定时间区间重叠

```
 public boolean overlap(LocalDateTime otherStart, LocalDateTime otherEnd) {
        if (!start.isAfter(otherEnd) && !end.isBefore(otherStart)) {
            return true;
        }
        return false;
    }
```

User

User(String, String) 构造方法，创建用户并初始化用户名、密码和会议ID列表
getUsername() 获取用户的用户名
getPassword() 获取用户的密码
getMeetingIds() 获取用户参与的所有会议ID列表
addMeetingId(String) 向用户的会议ID列表中添加一个ID
removeMeetingId(String) 从用户的会议ID列表中移除指定ID
clearMeetings() 清空用户的会议ID列表

四、实现效果

双击agenda.bat启动程序：

```
[INFO]
[INFO] -------------------------< com.Agenda:Agenda >-------------------------
[INFO] Building Agenda 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ Agenda ---
[INFO] Deleting D:\25spring\Compiler-Lab\lab2\target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ Agenda ---
[INFO] skip non existing resourceDirectory D:\25spring\Compiler-Lab\lab2\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ Agenda ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 6 source files with javac [debug release 23] to target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.668 s
[INFO] Finished at: 2025-04-09T20:40:08+08:00
[INFO] ------------------------------------------------------------------------
==============================
[SUCCESS] Compilation and tests passed!
==============================
==============================
[INFO] Starting Agenda System...
==============================
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.Agenda:Agenda >-------------------------
[INFO] Building Agenda 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ Agenda ---
$
```

输入相应命令：

```
[INFO] --- exec:3.1.0:java (default-cli) @ Agenda ---
$ register fxt 1234567
Register success.
$ register fff 2345678
Register success.
$ add fxt 1234567 fff 2024-04-08_10:00 2024-04-08_11:00 meeting114514
Meeting added with ID: MID-1286514925
$ query fxt 1234567 2024-04-08_10:00 2024-04-08_11:00
2024-04-08T10:00 2024-04-08T11:00 meeting114514 fff
$ query fxt 1234567 2024-04-08_08:00 2024-04-08_09:00
$ query fxt 1234567 2024-04-08_08:00 2024-04-08_12:00
2024-04-08T10:00 2024-04-08T11:00 meeting114514 fff
$ clear fxt 1234567
All meetings cleared.
```

我们创建了两个用户，分别为fxt和fff，并成功创建了一个会议。

在非会议时间查询该程序，是没有输出的。

最后我们清除了fxt的所有会议。

异常处理

不能重复创建名字相同的用户：

```
$ register fxt 222222
Username already exists.
```

命令行要输入合规的指令：

```
$ add fxt fff xxx
[Warning] Usage: add <username> <password> <participant> <start> <end> <title>
```

会议时间不能冲突：

```
$ add fxt 1234567 fff 2024-04-08_10:00 2024-04-08_11:00 meeting114514
Meeting added with ID: MID-1001096755
$ add fxt 1234567 fff 2024-04-08_10:00 2024-04-08_11:00 meeting88888
Error: Sponsor has time conflict.
```

批处理命令

batch.txt：

```
register Anna 123
register Brian 233
register Anna 1234
register Cindy 456

add Anna 123 Brian 2025-04-08 10:00 2025-04-08 11:00 meeting1
add Anna 123 Brian 2025-04-08 10:00 2025-04-08 11:00 meeting2
add Anna 123 Cindy 2025-04-08 10:30 2025-04-08 11:30 meeting3
add Brian 233 Anna 2025-04-08 09:00 2025-04-08 10:00 meeting4
add Anna 123 Cindy 2025-04-08 11:00 2025-04-08 12:00 meeting5

query Anna 123 2025-04-08 09:00 2025-04-08 12:00
query Brian 233 2025-04-08 09:00 2025-04-08 12:00
query Cindy 456 2025-04-08 09:00 2025-04-08 12:00
query Anna 123 2025-04-08 11:00 2025-04-08 12:00

delete Anna 123 meeting1
delete Anna 123 meeting6
delete Brian 233 meeting4
delete Cindy 456 meeting5

clear Anna 123
clear Brian 233
clear Cindy 456
clear Anna 123
```

```
[INFO] --- exec:3.1.0:java (default-cli) @ Agenda ---
$ batch batch.txt
Register success.
Register success.
Username already exists.
Register success.
Meeting added with ID: MID-1315765117
Error: Sponsor has time conflict.
Error: Sponsor has time conflict.
Error: Sponsor has time conflict.
Error: Sponsor has time conflict.
2025-04-08T10:00 2025-04-08T11:00 meeting1 Brian
2025-04-08T10:00 2025-04-08T11:00 meeting1 Brian
2025-04-08T10:00 2025-04-08T11:00 meeting1 Brian
Delete success.
Meeting not found or unauthorized.
Meeting not found or unauthorized.
Meeting not found or unauthorized.
All meetings cleared.
All meetings cleared.
All meetings cleared.
All meetings cleared.
```

可见里面成功创建了用户，同时也对错误操作进行了正确的响应。

退出程序

输入quit会退出程序

```
$ quit
Bye.
================================
[INFO] Execution finished. Press any key to exit.
================================
Press any key to continue . . .
```

五、Test板块

```java
public class AgendaTest {
    @Test
    public void testUserRegistrationAndAuthentication() {
        UserManager um = new UserManager();
        assertTrue(um.register(username:"alice", password:"123"));
        assertFalse(um.register(username:"alice", password:"456"));
        assertTrue(um.authenticate(username:"alice", password:"123"));
        assertFalse(um.authenticate(username:"alice", password:"wrong"));
    }

    @Test
    public void testMeetingAddAndConflict() {
        UserManager um = new UserManager();
        um.register(username:"a", password:"p1");
        um.register(username:"b", password:"p2");
        MeetingManager mm = new MeetingManager(um);
        LocalDateTime s1 = LocalDateTime.parse(text:"2025-04-08T14:00");
        LocalDateTime e1 = LocalDateTime.parse(text:"2025-04-08T15:00");
        assertTrue(mm.addMeeting(sponsor:"a", participant:"b", s1, e1, title:"meet1").contains(s:"Meeting added"));
        assertTrue(mm.queryMeetings(username:"a", s1, e1).size() == 1);
        assertFalse(mm.addMeeting(sponsor:"a", participant:"b", s1, e1, title:"meet2").contains(s:"Meeting added"));
        assertTrue(mm.deleteMeeting(username:"a", title:"meet1"));
    }
```

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.Agenda.AgendaTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.059 s -- in com.Agenda.AgendaTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.700 s
[INFO] Finished at: 2025-04-09T20:55:21+08:00
[INFO] ------------------------------------------------------------------------
```

六、Javadoc

在doc文件夹中

所有类和接口

类

类	说明
AgendaService	议程服务类，提供命令行接口供用户交互。
CommandProcessor	命令处理器类，用于解析和执行用户输入的命令。
Meeting	表示一个会议，包含会议的标题、发起人、参与人、开始时间和结束时间等信息。
MeetingManager	会议管理类，用于管理用户的会议，包括添加、查询、删除会议等功能。
User	表示一个用户，包含用户的用户名、密码和已参与的会议 ID 列表。
UserManager	用户管理类，用于注册、认证和管理用户信息。

所有程序包

程序包概要

程序包
com.Agenda
com.Agenda.Manager
com.Agenda.Model

索引

A C D G M O P Q R U
所有程序包 | 所有类和接口

A

addMeeting(String, String, LocalDateTime, LocalDateTime, String) - 类中的方法 com.Agenda.Manager.MeetingManager
添加一场会议，检查会议时间冲突和参与者是否注册。

addMeetingId(String) - 类中的方法 com.Agenda.Model.User
向用户的会议 ID 列表中添加一个新的会议 ID。

AgendaService - com.Agenda中的类
议程服务类，提供命令行接口供用户交互。

AgendaService() - 类的构造器 com.Agenda.AgendaService

authenticate(String, String) - 类中的方法 com.Agenda.Manager.UserManager
验证用户的身份。

C

clearMeetings() - 类中的方法 com.Agenda.Model.User
清空用户的会议 ID 列表。

clearMeetings(String) - 类中的方法 com.Agenda.Manager.MeetingManager
清空指定用户的所有会议。

com.Agenda - 程序包 com.Agenda

com.Agenda.Manager - 程序包 com.Agenda.Manager

com.Agenda.Model - 程序包 com.Agenda.Model

CommandProcessor - com.Agenda中的类
命令处理器类，用于解析和执行用户输入的命令。

CommandProcessor(UserManager, MeetingManager) - 类的构造器 com.Agenda.CommandProcessor
构造函数，初始化命令处理器。

D

deleteMeeting(String, String) - 类中的方法 com.Agenda.Manager.MeetingManager
删除指定用户的指定会议。

G

getEnd() - 类中的方法 com.Agenda.Model.Meeting
获取会议的结束时间。

getId() - 类中的方法 com.Agenda.Model.Meeting
获取会议的唯一 ID。

所有程序包的分层结构

程序包分层结构:
com.Agenda, com.Agenda.Manager, com.Agenda.Model

类分层结构

- java.lang.**Object**
  - com.Agenda.**AgendaService**
  - com.Agenda.**CommandProcessor**
  - com.Agenda.Model.**Meeting**
  - com.Agenda.Manager.**MeetingManager**
  - com.Agenda.Model.**User**
  - com.Agenda.Manager.**UserManager**
