華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY

软件构造实验报告

第二次实验

姓名 单禹嘉

学号 2023215177

课程 软件构造

学院 计算机学院

2025 年 10 月 30 日

摘　要

本实验致力于构建一个智能家居设备管理系统，能够统一管理和控制多种智能设备，包括灯、空调、窗帘及电视。通过面向对象的设计与实现，本系统满足设备的通用与个性化控制，并通过 HomeManager 类实现统一的调度与管理。实验通过实际编码和测试，验证了面向对象封装、继承、多态等核心思想在工程开发中的作用。

目录

1 实验目的和意义

本实验旨在:

- 理解面向对象程序设计思想在实际工程中的应用;
- 熟练掌握类的封装、继承、多态及其带来的系统扩展性和维护性优势;
- 学会如何统一管理多种智能家居设备，并支持设备个性特性的扩展;
- 学会细致分析类的职责与对象关系，为高内聚低耦合的软件设计打下基础。

2 问题描述

在实际的智能家居系统中，需要统一管理各类智能设备，对设备开关状态进行控制，并对不同类型设备实现特有控制功能，如调节亮度、温度或打开百分比等，同时需要能够展示所有设备的当前状态。本实验的目标是用面向对象程序设计方式，实现这样一个能够灵活扩展、方便维护和统一控制的智能家居设备管理系统。

3 需求与实验要求

- 管理多种智能设备，包括智能灯、空调、窗帘等，并可灵活扩展新的设备类型;
- 实现设备的统一开关控制接口，并支持各自的特有功能（如亮度调节、温度调节、模式切换等);
- 可视化展示所有设备的当前状态（开/关、亮度、温度等);
- 支持异常处理，如设备未找到、参数设置非法等情况;
- 支持面向对象的场景模式（如“夜间模式”同时控制多设备);
- 源代码实现需符合封装、继承、多态、单一职责、开闭原则等面向对象设计原则。

4 实验环境

- 开发工具：VS Code
- 编程语言：JavaScript (Node.js)

5 设计思想与实验步骤

5.1 总体设计

设计上，系统采用典型的面向对象分层结构：

- 抽象父类 `SmartDevice` 封装设备共性；
- 各设备子类（如 `SmartLight`、`SmartAC`、`SmartCurtain`、`SmartTV` ）分别实现个性化行为；
- `HomeManager` 类负责统一设备管理和调度，体现多态扩展能力；
- `SceneMode` 类体现组合设计思想，实现一组设备的批量自动化控制。

5.2 类结构与继承关系

见 **??**。

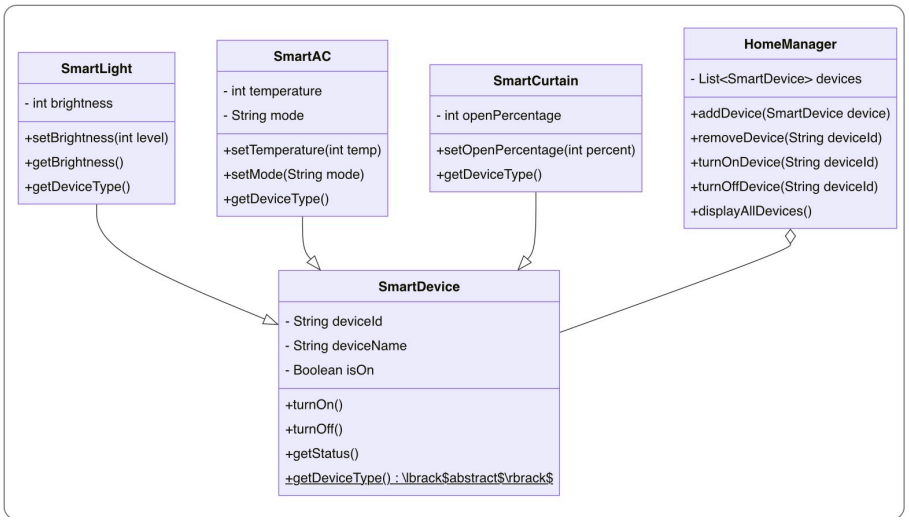

图 1: 智能家居系统类图

5.3 主要类及方法说明

- **Controllable（接口）**：定义统一的 `turnOn/turnOff` 方法（接口/ABC）。
- **SmartDevice（抽象父类）**：所有设备的统一属性和通用方法；ID、名称、开关状态私有化存储，暴露公共只读接口。
- **SmartLight / SmartAC / SmartCurtain / SmartTV（具体设备子类）**：分别实现特有的控制能力，如设置亮度、温度、频道、音量等。

- **HomeManager**：存储和管理全部设备实例，提供加入/移除/统一控制及展示能力。
- **SceneMode**：聚合多个“动作”，实现场景控制模式。

5.4 计算步骤及核心流程

1. 定义统一抽象父类及接口 → 实现各个具体智能设备 → 统一加入 HomeManager 管理。
2. 通过 HomeManager 批量控制、展示、异常处理。
3. 通过 SceneMode 实现批量场景模式并测试正确性。

6 主要实现与测试

6.1 部分主要源代码

（完整代码见附录/lab2 目录，下方为部分核心片段摘录）

```
// SmartDevice.js
class SmartDevice {
    constructor(deviceId, deviceName) {
        if (new.target === SmartDevice) {
            throw new Error("不能直接实例化SmartDevice抽象类");
        }
        this._deviceId = deviceId;
        this._deviceName = deviceName;
        this._isOn = false;
    }
    turnOn() { this._isOn = true; }
    turnOff() { this._isOn = false; }
    getStatus() { return this._isOn ? "开" : "关"; }
    get deviceId() { return this._deviceId; }
    get deviceName() { return this._deviceName; }
    get isOn() { return this._isOn; }
    getDeviceType() { throw new Error("请在子类中实现getDeviceType方法"); }
}

// SmartLight.js
```

```
class SmartLight extends SmartDevice {
    constructor(deviceId, deviceName, brightness = 100) {
        super(deviceId, deviceName);
        this._brightness = brightness;
    }
    setBrightness(level) { /* ... */ }
    getDeviceType() { return "智能灯"; }
    getStatus() { return `${super.getStatus()}, 亮度: ${this._brightness}`; }
}
// ... 其余代码见附件 ...
```

6.2 功能与效果展示

- 支持多种类型设备，灵活增减，且能统一开关与特有功能控制；
- 支持一次控制多设备场景，如一键夜间模式：SceneMode 批量设置行为；
- 完善的异常处理与边界条件检验，如 ID 重复，参数非法等都能给出明确提示。

7 关键问题分析与回答

7.1 封装分析

被封装的属性：如 _deviceId、_deviceName、_isOn 等全部设为私有（类内访问），通过 getter 暴露只读访问接口。封装的目的是防止外部直接篡改，增强类的健壮性与安全性，保护对象一致性。

7.2 继承层次与说明

见 **??**。各设备子类（灯、空调、窗帘、电视）均继承 SmartDevice 的共性（ID、名称、开关、状态展示等），并根据不同子类实现额外专有操作（如亮度、温度、模式等）。

7.3 多态体现

HomeManager 中的设备操作方法（如 turnOnDevice、turnOffDevice 和 displayAllDevices），是通过调用基类接口来操作不同子类对象，具体运行时会自动分发到正确的子类实现（如 getStatus、turnOn），这是多态机制的直接体现。

7.4 类的扩展性与修改性

如果添加新设备类型，只需新增对应子类，无需更改 HomeManager 的核心逻辑。因为其对所有设备采用了接口/基类依赖，不需区分具体类型。这符合开闭原则（对扩展开放，对修改封闭）。

7.5 设计原则分析

- **开闭原则**：只需扩展即可接入新设备类型，无需大规模更改已有代码。
- **单一职责**：每个类职责单一，SmartXxx 仅描述单一设备，HomeManager 负责管理，SceneMode 负责场景。

7.6 改进建议

- 可增加设备与网络/用户的交互、事件推送、远程控制等功能。
- 设备属性和方法可进一步抽象为接口，提升灵活性。
- 可集成数据库和前端页面，提升可视化及数据持久化能力。