软件构造实验2

1. 类设计与规则

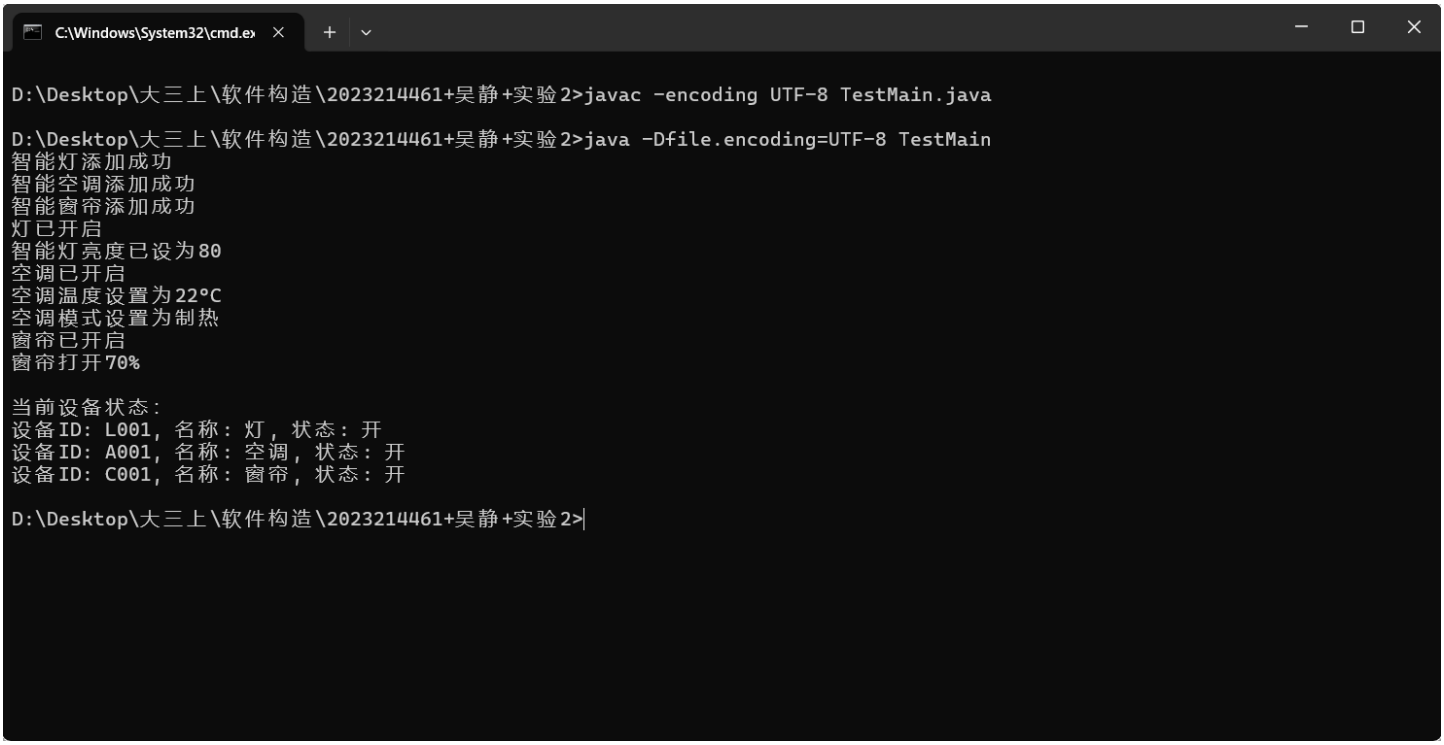
- 类图

2. 编码实现

代码见文件夹

3. 测试与验证

结果如图



4. 问题回答

- 封装分析：哪些属性被封装了？为什么要将这些属性设为private?
  - 被封装的属性：`deviceId`，`deviceName`，`isOn`，以及子类特有属性
  - 原因：防止外部直接修改设备状态，保证数据安全和一致性，通过公共方法控制访问。
- 继承层次：画出完整的类继承图，说明每个子类继承了哪些特性?
  - 见步骤1的类图
  - `SmartDevice`是基类，`SmartLight、SmartAC、SmartCurtain`都继承自它。每个子类继承了公共方法（`turnOn、turnOff、getStatus`）和公共属性（`deviceId、deviceName、isOn`）
- 多态体现：在`HomeManager`的哪个方法中体现了多态？如果添加新的设备类型，需要修改`HomeManager`类吗？为什么？
  - `HomeManager`中的`turnOnDevice()`与`turnOffDevice()`体现了多态。调用`SmartDevice`的`turnOn()`方法时，实际执行的是子类重写或继承的实现。
  - 添加新设备时，不需要修改`HomeManager`，只需要继承`smartDevice`，体现了开闭原则
- 设计原则：这个设计是否符合开闭原则？请举例说明。每个类的职责是否单一？分析各个类的职责。
  - 符合开闭原则（添加新设备时，不需要修改`HomeManager`，只需要继承`smartDevice`）
  - 每个类的职责都单一。分别负责定义通用特性、控制灯、控制空调、控制窗帘和负责设备管理

- 改进建议：你认为这个系统在面向对象设计方面还有哪些可以改进的地方？
  - 添加新设备类型、场景模式、异常处理、接口设计

5. **扩展**

- 接口

代码块

```
public interface Controllable {
    void turnOn();
    void turnOff();
    String getStatus();
}

在SmartDevice.java中
public abstract class SmartDevice implements Controllable{
...
}
```

- smartTV

代码块

```
public class SmartTV extends SmartDevice {
    private int channel;
    private int volume;

    public SmartTV(String id, String name) {
        super(id, name);
        this.channel = 1;
        this.volume = 20;
    }

    public void setChannel(int channel) {
        if (channel <= 0) {
            System.out.println("频道号必须大于0");
            return;
        }
        this.channel = channel;
        System.out.println(getDeviceName() + "切换到频道" + channel);
    }

    public void setVolume(int volume) {
        if (volume < 0 || volume > 100) {
            System.out.println("音量范围为0-100");
            return;
```

```
        }
        this.volume = volume;
        System.out.println(getDeviceName() + "音量设置为" + volume);
    }

    public int getChannel() {
        return channel;
    }

    public int getVolume() {
        return volume;
    }

    @Override
    public String getDeviceType() {
        return "智能电视";
    }
}

```

- 设备不存在异常

代码块

```
public class DeviceNotFoundException extends Exception {
    public DeviceNotFoundException(String message) {
        super(message);
    }
}

```

- 场景模式

代码块

```
import java.util.*;

public class SceneMode {
    private String sceneName;
    private List<SmartDevice> deviceList;

    public SceneMode(String sceneName) {
        this.sceneName = sceneName;
        this.deviceList = new ArrayList<>();
    }

```

```
    public void addDevice(SmartDevice device) {
        deviceList.add(device);
    }

    public void activate() {
        System.out.println("激活场景: " + sceneName);
        for (SmartDevice device : deviceList) {
            device.turnOn();
        }
    }

    public void deactivate() {
        System.out.println("关闭场景: " + sceneName);
        for (SmartDevice device : deviceList) {
            device.turnOff();
        }
    }
}

```