**Practical Work Report №5**

[**https://github.com/KNari06/SDP\_Assignment\_5.git**](https://github.com/KNari06/SDP_Assignment_5.git)

Course: Software Design and Pattern

Topic: Assignment 5

Student: Kaidar Nariman

Group: SE-2406

Instructor: Makasheva Teili

Date: 17.10.2025

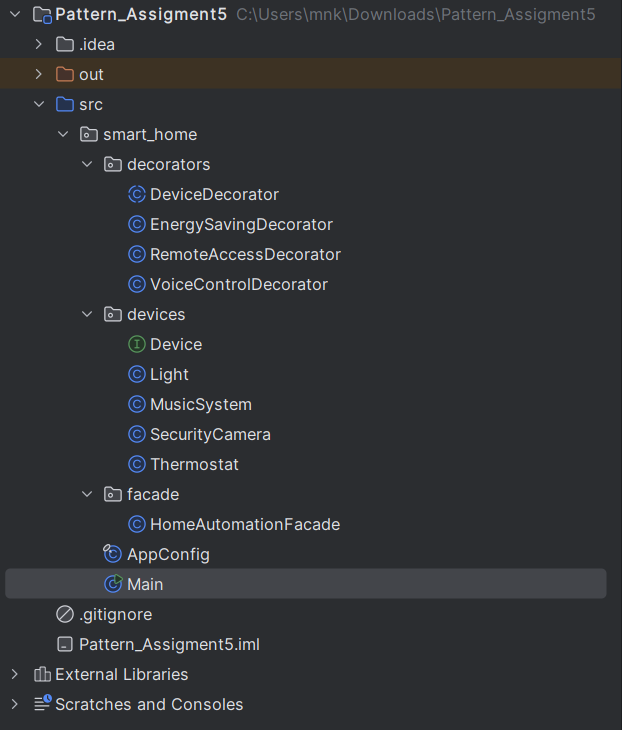
**Assignment 5: Smart Home Automation System**

**Patterns Used:** Decorator + Facade

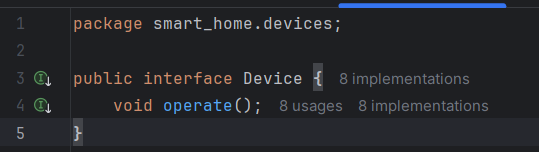
## **Introduction**

This project implements a **Smart Home Automation System** that manages multiple smart devices such as lights, thermostats, cameras, and music systems.  
The system demonstrates the use of two structural design patterns — **Decorator** (for extending device functionality dynamically) and **Facade** (for simplifying user control).

## Project Structure

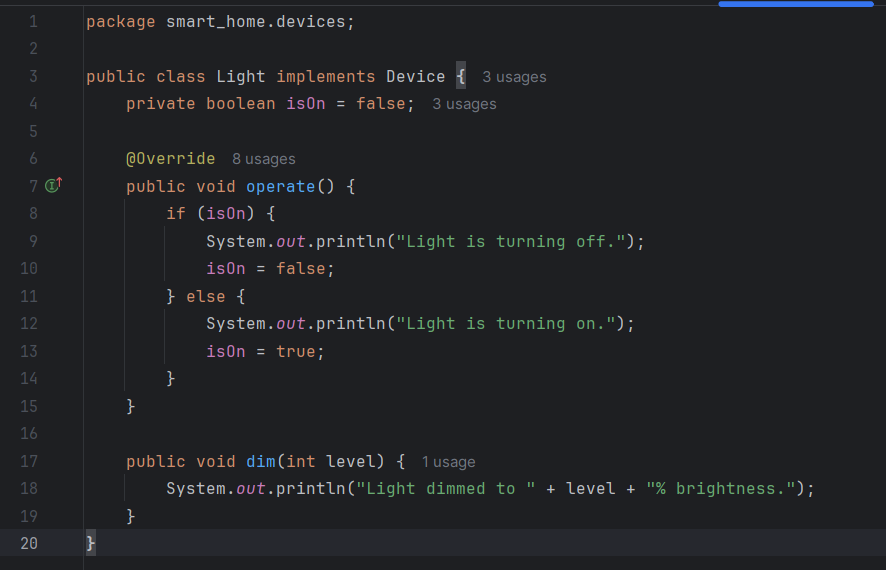


## **🔹 Device Abstraction**



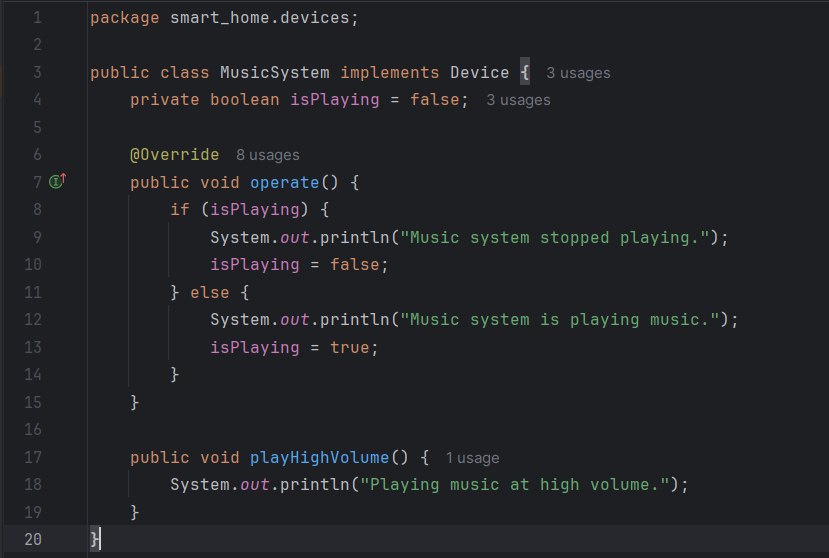
**Explanation:**  
Defines a common interface for all smart devices. Each device must implement the operate() method.

### **Light.java**



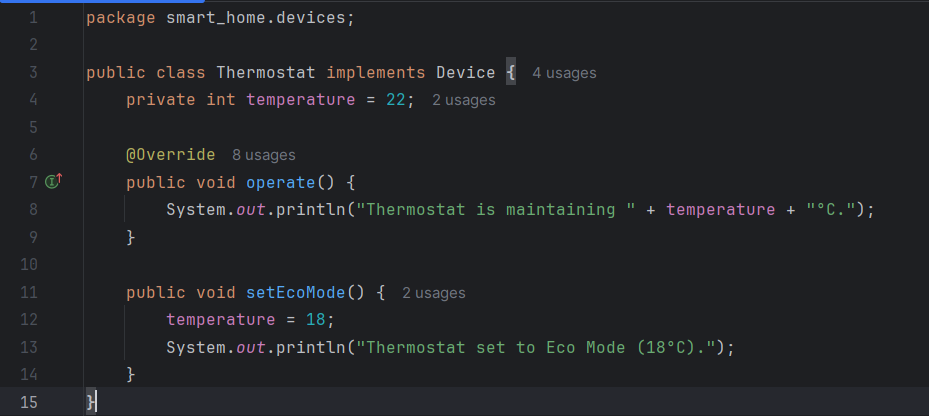
**Explanation:**  
Represents a light device that can turn on/off and adjust brightness using dim().

### **MusicSystem.java**



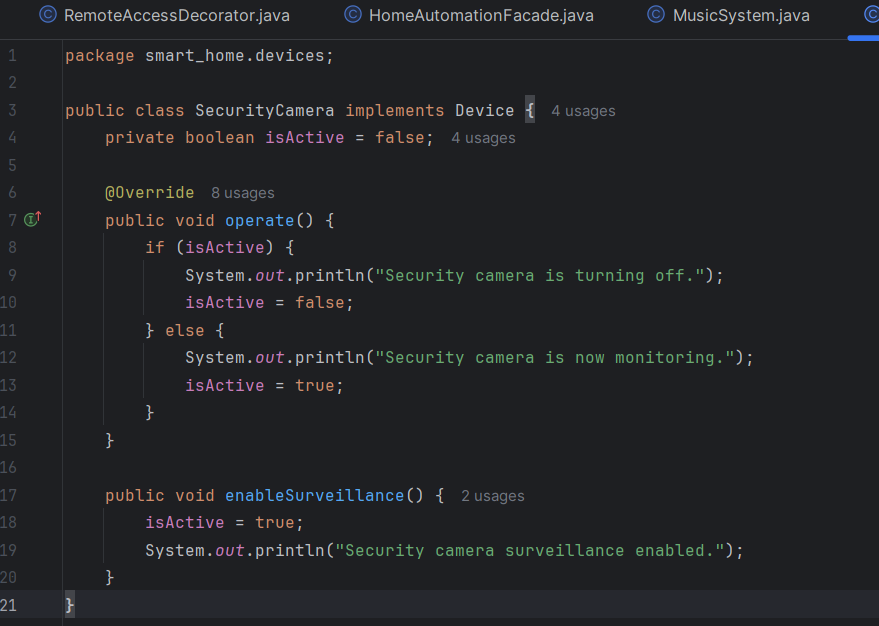
**Explanation:**  
Represents a speaker system that can play or stop music, and supports a high-volume mode.

### **Thermostat.java**



**Explanation:**  
Controls room temperature and supports Eco Mode to save energy.

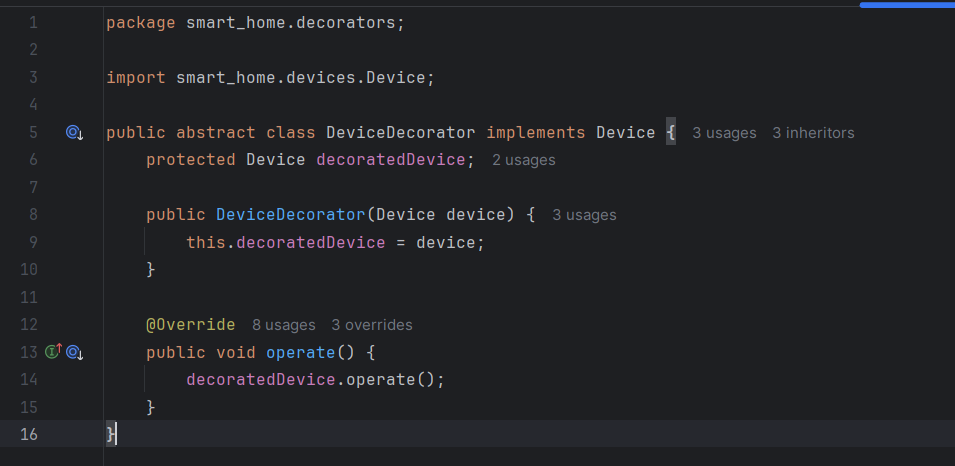
### **SecurityCamera.java**



**Explanation:**  
Represents a home security camera with activation and surveillance features.

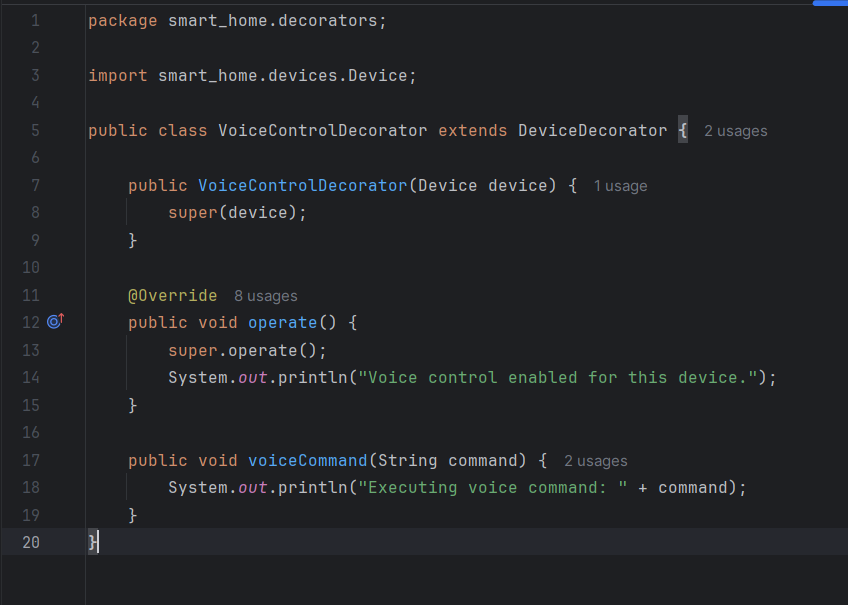
## **🔹 Decorator Pattern Implementation**

### **DeviceDecorator.java**

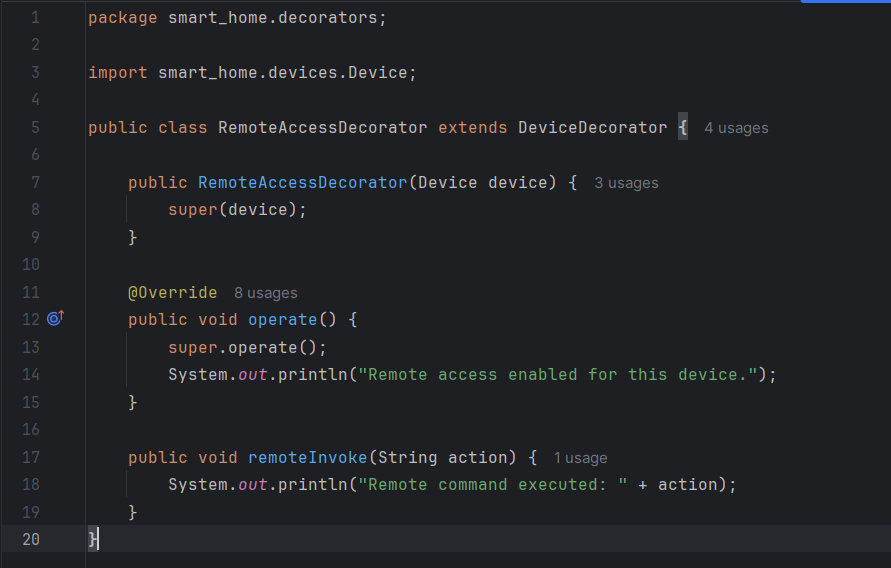


**Explanation:**  
This abstract class allows wrapping devices dynamically.  
It follows the **Open/Closed Principle** — new behaviors can be added without changing existing code.

### **VoiceControlDecorator.java**

**Explanation:**  
Adds voice control functionality to any device (e.g., "Turn on the lights").

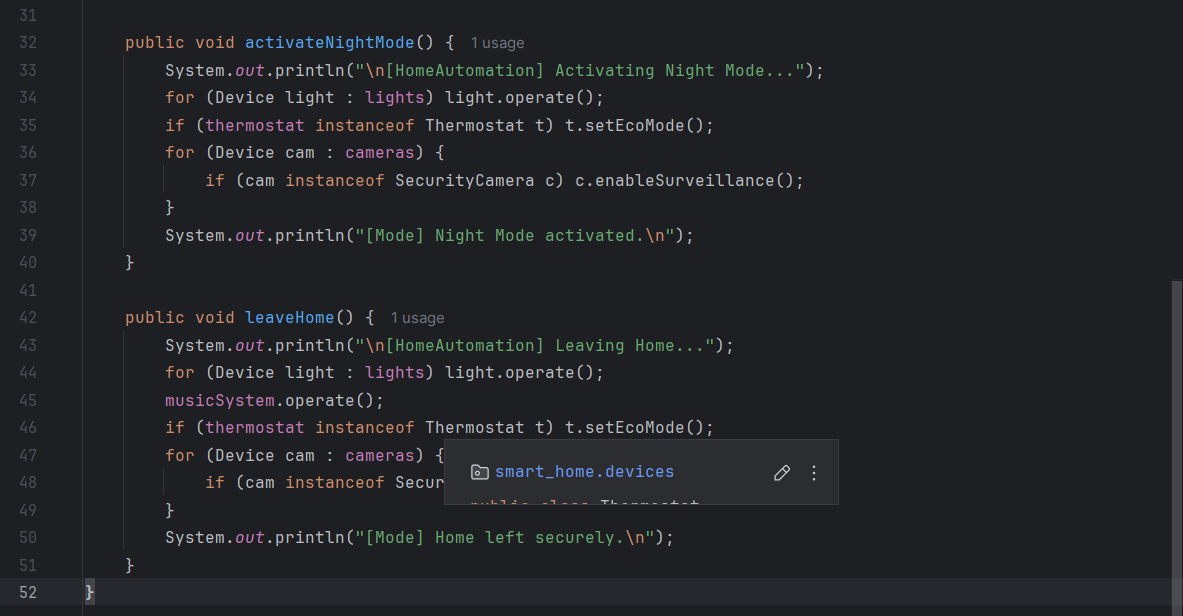
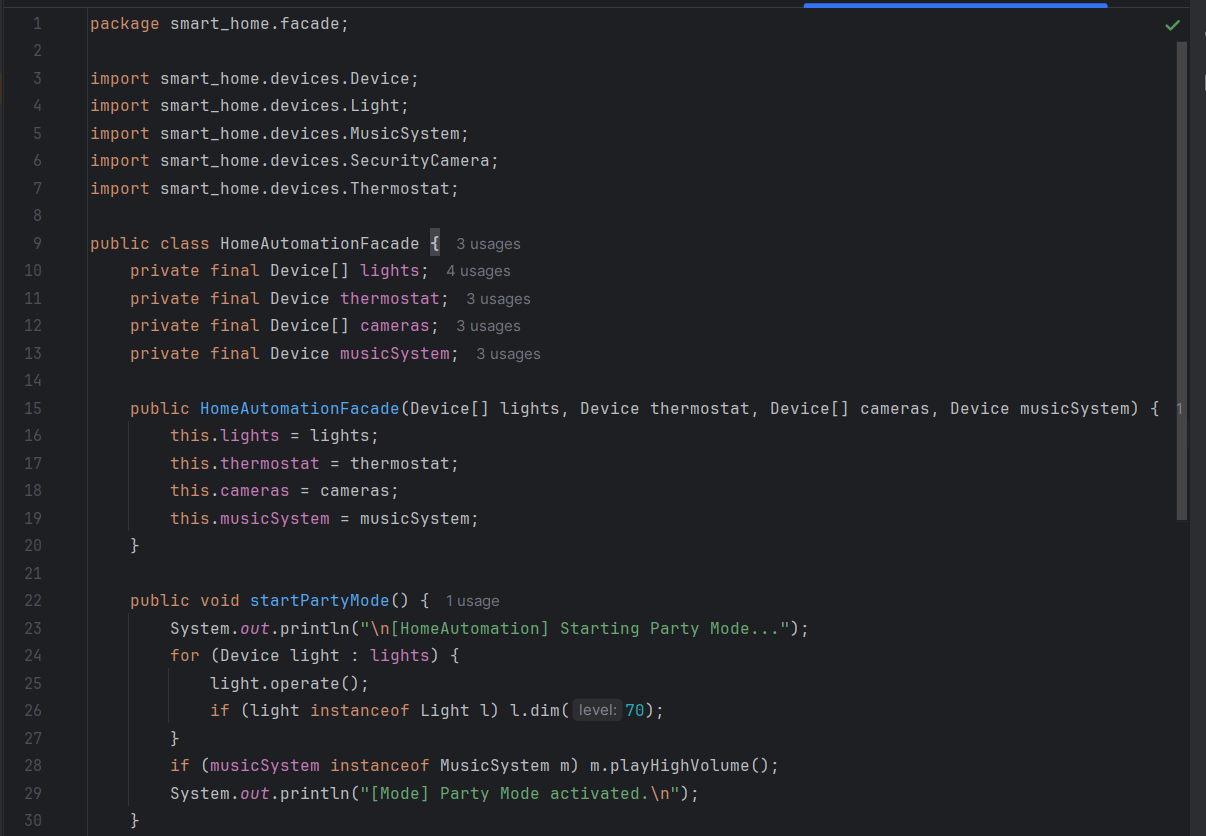
### **RemoteAccessDecorator.java**



**Explanation:**  
Adds internet-based remote control capability.

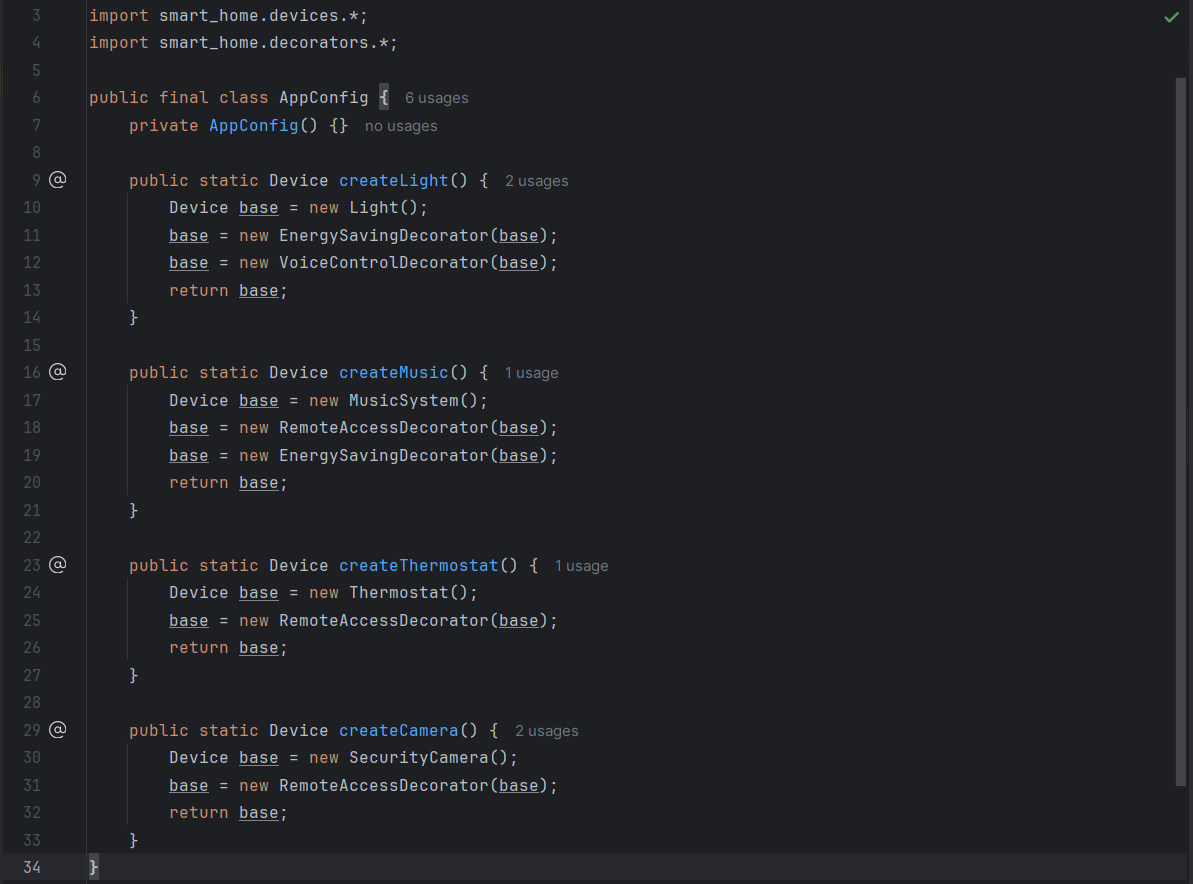
**🔹 Facade Pattern Implementation**

### **HomeAutomationFacade.java**

**Explanation:**  
The **Facade** hides complexity by offering simple high-level commands like:

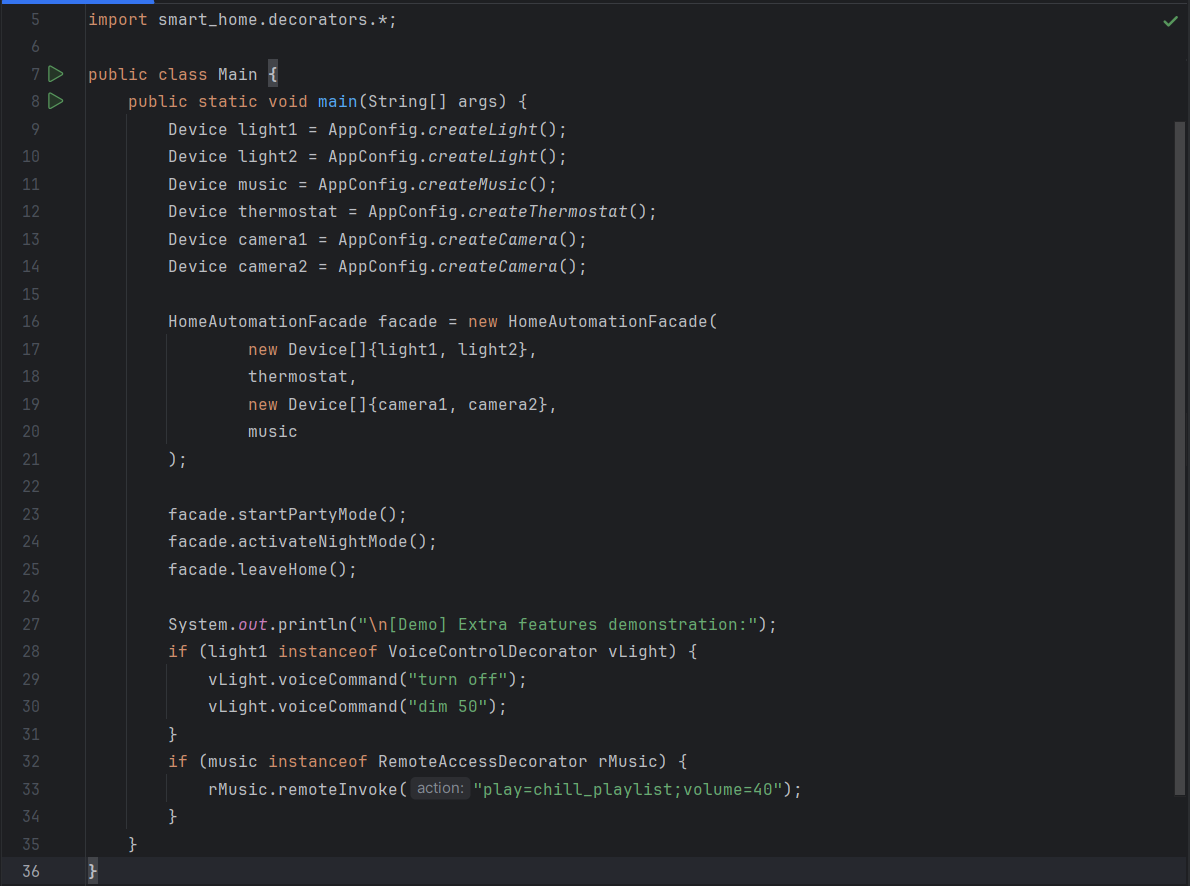
* startPartyMode()
* activateNightMode()
* leaveHome()

## **🔹 AppConfig.java**



**Explanation:**  
Central configuration class — creates devices and applies decorators cleanly.  
Follows **Single Responsibility** and **Open/Closed** principles.

## **🔹 Main.java**

**Explanation:**  
The main method only runs high-level actions, keeping the code clean and simple.  
It demonstrates **Facade simplification** and **Decorator flexibility**.

**.🔹 Conclusion**

This project demonstrates a **clean, extensible, and maintainable** smart home system that uses two powerful design patterns:

* **Decorator** → for dynamic feature extension
* **Facade** → for simplified control

The system follows **Clean Code** principles:

* Small, single-responsibility classes
* Clear separation of concerns
* Easy to add new devices or decorators