Smart Home Controller Application: Design Pattern Implementation Report

1. Introduction

This project implements a Java-based Smart Home Controller application that simulates controlling various smart home devices like lights, doors, and air conditioners. The system used Singleton, Factory, Decorator, Observer, and Strategy design patterns to create a flexible, maintainable architecture that allows for basic device control, room management, action logging, and automation modes.

2. Design Patterns Implemented

2.1 Singleton Pattern

I used the Singleton pattern in the SmartHomeController and ActionLogger classes to make sure only one instance of the controller and logger exists throughout the system. This was important to prevent conflicting device operations from multiple controller instances and to keep a consistent, centralized log of all actions across the system. It also provides global access to these components which simplifies the Smart Home Controller system management.

2.2 Factory Pattern

The Factory Pattern was implemented in DeviceFactory class to create different device types based on input parameters. This centralizes device creation, and encapsulates complex instantiation logic, and makes the system easier to maintain and extend with new device types.

2.3 Decorator Pattern

To add functionality to devices dynamically, I applied the Decorator pattern through the DeviceDecorator and MotionSensorDecorator classes. This allowed flexible feature extensions like motion detection capabilities to a basic device without modifying existing classes.

2.4 Observer Pattern

The Observer pattern was implemented through a Subject interface in the AbstractDevice class and an Observer interface in the ActionLogger class. The Observer pattern was used to create a

loosely coupled interaction between devices and the logging mechanism. This setup allowed devices to automatically notify the logger when their state changed to ensure that actions were consistently recorded while allowing new observers to be integrated without modifying the device logic.

2.5 Strategy Pattern

Finally, I implemented the Strategy design pattern using the AutomationMode interface, with NightMode class and VacationMode class as concrete strategies. Using this approach allowed for interchangeable automation behaviors that could be selected at runtime, which made the system more adaptable. It also supported adding new automation modes without modifying the rest of the system, by encapsulating each strategy in a separate class for clarity and modularity.

3. Key Classes and Interfaces

Device Interface: Defines the basic operations for all smart devices

AbstractDevice Class: Provides common device functionality and implements the Subject interface.

Room Class: Groups devices by location and provides room-level operations

SmartHomeController Class: Central controller implementing Singleton pattern, manages rooms and devices.

DeviceFactory Class: Implements Factory pattern to create device instances.

DeviceDecorator and MotionSensorDecorator Classes: Implement Decorator pattern for runtime feature extension.

Subject and Observer Interfaces: Define Observer pattern contracts.

ActionLogger Class: Implements Observer pattern to log device state changes.

AutomationMode Interface: Defines the Strategy pattern contract for automation modes.

NightMode and VacationMode Classes: Implement concrete strategies for different automation scenarios.

4. Challenges Faced

One major challenge was the initial tight coupling between the controller and logger due to an incomplete Observer pattern. Instead of devices notifying observers, the controller directly called the logger. I fixed this by fully implementing the Subject-Observer relationship, where devices notify registered observers automatically when their state changes.

Another challenge was making sure decorators added new behavior while still forwarding method calls to the original device and supporting observation. I addressed this by designing the decorator methods to both extend functionality and preserve the original behavior of the device.

Lastly, organizing the project structure that reflected the relationships between components took me some effort. I handled this by organizing the code into clear packages such as controllers, devices, decorators, observers, rooms, and strategy that clearly define component relationships and responsibilities.

In conclusion, the Smart Home Controller application demonstrates how multiple design patterns can work together to create a flexible, maintainable system. By applying the Singleton pattern, I ensured centralized control and consistent logging. The Factory pattern simplified device creation and made it easy to scale with new device types. Through the Decorator pattern, I was able to add features like motion detection without altering existing code, while the Observer pattern enabled automatic logging of device activity. Finally, the Strategy pattern allowed for flexible automation modes that can adapt to user preferences. The combined use of Singleton, Factory, Decorator, Observer, and Strategy patterns creates a Smart Home Controller application that is easy to modify, maintain, while remaining extensible for future enhancements.