Smart Home Controller – Project Report

Introduction

This report documents the implementation of a Java-based **Smart Home Controller** application. By simulating centralised control of smart devices across multiple rooms, the application lets users:

- Switch individual devices on or off,
- Group devices by room, and
- Activate preset routines such as **Night Mode**, **Vacation Mode, Morning Mode** and **Party Mode**.

The codebase is grounded in well-established design patterns and object-oriented principles to ensure scalability and maintainability.

Design Patterns Employed

Pattern	Purpose	Key Class(es)
Singleton	Guarantees a single instance of the controller, preventing inconsistent device states or duplicated logic.	SmartHomeController
Factory	Decouples device creation from controller logic, easing future extension to new device types.	DeviceFactory

Observer	Enables modular monitoring of device-state changes without tying observers to device internals.	DeviceObserver, ActionLogger
Strategy	Encapsulates automation routines to be added, removed, or swapped without altering controller code.	NightMode, VacationMode, MorningMode, NightMode

Core Components

- **SmartHomeController**: Central hub for device and room management, observer registration, and automation execution.
- **Device (abstract) + subclasses**: Define common properties (name, powerState) and specialised behaviour for Light, Door, Thermostat, Camera and future devices.
- **DeviceFactory**: Static factory that validates input and instantiates the appropriate device type.
- **Room**: Aggregates multiple devices and supplies batch-toggle functionality.
- **ActionLogger**: Concrete observer that records every power-state change, functioning as a basic audit log.

Challenges Encountered

1. **Selecting Appropriate Patterns**

   The Singleton and Factory choices were straightforward, but integrating logging (Observer) and automation routines (Strategy) required several iterations to avoid cluttering core classes.

2. **Console Input & Error Handling**

   Building a robust CLI proved trickier than expected. Extensive defensive programming and testing were necessary to cope with invalid or unexpected user entries.

3. **Maintaining Codebase Organisation**

   As the project grew, the number of classes grew, which threatened the clarity of the code. Splitting the code into directories (devices, observer, controller, strategy) and periodic refactoring kept the architecture coherent.

Conclusion

The project successfully demonstrates the practical application of the Singleton, Factory, Observer, and Strategy patterns to create a clean, modular Smart Home Controller. Despite challenges with input handling and architectural upkeep, the final system meets its functional and extensibility goals and provides a solid foundation for future enhancements.