PA5_Demo_Info

Smart Home Automation System - Design Patterns Overview

This document outlines the design patterns used in the Smart Home Automation System project, providing an overview of their implementation and where they can be found within the project files.

1. Command Pattern

Location:

- **Header:** `Command.h`
- **Implementation:** `Command.cpp`
- **Tested in:** `TestingMain.cpp`

Description:

The **Command Pattern** is used to encapsulate requests as objects, allowing the system to execute commands on devices in a decoupled way. This is particularly useful for actions like turning off all lights or locking all doors in the home automation system.

Implementation Details:

- **Commands:** `TurnOffAllLights`, `LockAllDoors`
- Each command operates on a list of `SmartDevice` objects. For example, `TurnOffAllLights` iterates over all devices and toggles the lights if the device type is "Light".

Usage Example in Code:

```
TurnOffAllLights turnOffAllLights(devices);
turnOffAllLights.execute(); // Executes the command to turn off all lights
```

2. Adapter Pattern

Location:

- **Header:** `SmartThermostatAdapter.h`
- **Implementation:** `SmartThermostatAdapter.cpp`
- **Tested in:** `TestingMain.cpp`

Description:

The **Adapter Pattern** is used to adapt legacy devices that don't support smart functionality, allowing them to be controlled within the smart home system. In this project, the `SmartThermostatAdapter` adapts a `LegacyThermostat` so that it behaves like a `SmartDevice`.

Implementation Details:

- **Class:** `SmartThermostatAdapter`
- The adapter wraps the legacy thermostat and provides methods like `performAction()` and `getStatus()` to make it compatible with the system.

Usage Example in Code:

```
LegacyThermostat oldThermostat;
SmartThermostatAdapter smartThermostat(&oldThermostat);
std::cout << smartThermostat.getStatus(); // Outputs the adapted
thermostat's status
```

3. Observer Pattern

Location:

- **Header:** `Sensor.h`
- **Implementation:** `Sensor.cpp`
- **Tested in:** `TestingMain.cpp`

Description:

The **Observer Pattern** is used for notifying smart devices (observers) when certain environmental conditions are met. For example, when a motion sensor detects movement, it can notify connected devices (e.g., lights) to perform actions.

Implementation Details:

- **Classes:** `Sensor` (Subject), `MotionSensor` (Observer)
- The sensor holds a list of observers (smart devices) and notifies them when a specific condition, like motion detection, occurs.

Usage Example in Code:

```
MotionSensor motionSensor(&livingRoomLight);
Sensor sensor;
sensor.addDevice(&motionSensor);
sensor.notifyDevices(); // Notifies devices when motion is detected
```