# 

# **Documentation:** Smart Home Control System Project

By Tolstova Kristina and Rashidov Aziz

# **Table of contents:**

[Documentation: Smart Home Control System Project](#_t55gh0ujoi9g)

[Table of contents:](#_txz2pmqg2l93)

[Part 1 - Architecture and Basic Logic](#_ieq3rd3tyxpq)

[1. Introduction](#_o83clsgleu9i)

[2. Code structure](#_ttrf8bwt8451)

[2.1 Composite Pattern (Device structure)](#_16c96y34aiqu)

[2.1.1 Interface SmartDevice](#_xkiognhskusy)

[2.1.2 Классы устройств (Leaf)](#_kdf42jmf0ch3)

[Class Light](#_j1x3penblbfa)

[Class Camera](#_ujv1vppyxt24)

[Class RobotVacuum](#_16c5av7xhe73)

[2.1.3 Class DeviceGroup](#_1tdj0kc2nvug)

[2.2 Abstract Factory Pattern](#_a66xezjghxj6)

[Specific implementation BasicSmartHomeFactory](#_7xus78dbhyrr)

[2.3 Facade Pattern](#_rg8b3865zxuk)

[3. Conclusion](#_vkktyqzhom1u)

[Part 2 - Extended Functionality](#_b7a007glxk6l)

[1. Introduction](#_ywd0mjsqgleh)

[2. Code Structure](#_7zgwkl96jk9l)

[2.1 Advanced Factory for Smart Devices](#_y9rbcrva7hoh)

[2.2 AI\_Thermostat Class](#_ezbah6kp5yj0)

[2.3 Decorator Pattern (Enhancing Devices)](#_vgs5uz3wn9fd)

[EnergySavingDecorator](#_6gn1bbuf63uk)

[LoggingDecorator](#_ele4vfgdwmom)

[ScheduledOperationDecorator](#_nudoub9n68ay)

[2.4 Adapter Pattern (Integrating Legacy Systems)](#_5lqlptusp50i)

[LegacyDoorLock](#_iwfmiy4v9bvr)

[LegacyDoorLockAdapter](#_l80hhg4ox763)

[3. Conclusion](#_bsfskaldyj28)

[Conclusion: Smart Home Control System](#_geofl6bx8rj2)

[Overview](#_65gcl4dxzdvk)

[Summary of Part 1: Core Architecture](#_2euvhqnxomiw)

[Summary of Part 2: Advanced Functionality](#_3owt2mx5u1sm)

[Overall Conclusion](#_9ix8calfsiew)

# **Part 1 - Architecture and Basic Logic**

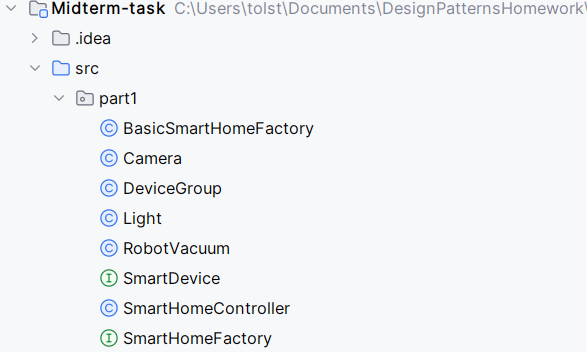
By Tolstova Kristina

## **1. Introduction**

**Smart Home Control System** - is a smart home control system that demonstrates the use of several design patterns. The first part implements:

* **Composite** (device hierarchy);
* **Abstract Factory** (device creation);
* **Facade** (single point of device control).

## **2. Code structure**

The code is organised in the part1 package, which contains the following key components:

### **2.1 Composite Pattern (Device structure)**

**Description:** Allows you to organise a hierarchy of devices where individual devices (Leaf) can be combined into groups (Composite) such as rooms or floors.

#### **2.1.1 Interface SmartDevice**

package part1;

*// Интерфейс умного устройства*

public interface SmartDevice {

void turnOn();

void turnOff();

String getStatus();

}

#### **2.1.2 Классы устройств (Leaf)**

##### **Class Light**

package part1;

*// Конкретные устройства (Листовые элементы для Composite Pattern)*

class Light implements SmartDevice {

private String name;

public Light(String name) {

this.name = name;

}

@Override

public void turnOn() {

System.*out*.println(name + " включен.");

}

@Override

public void turnOff() {

System.*out*.println(name + " выключен.");

}

@Override

public String getStatus() {

return name + ": Включен";

}

}

##### **Class Camera**

package part1;

class Camera implements SmartDevice {

private String name;

public Camera(String name) {

this.name = name;

}

@Override

public void turnOn() {

System.*out*.println(name + " активирована.");

}

@Override

public void turnOff() {

System.*out*.println(name + " деактивирована.");

}

@Override

public String getStatus() {

return name + ": Активна";

}

}

##### **Class RobotVacuum**

package part1;

class RobotVacuum implements SmartDevice {

private String name;

public RobotVacuum(String name) {

this.name = name;

}

@Override

public void turnOn() {

System.*out*.println(name + " начал уборку.");

}

@Override

public void turnOff() {

System.*out*.println(name + " остановил уборку.");

}

@Override

public String getStatus() {

return name + ": Работает";

}

}

#### **2.1.3 Class DeviceGroup**

package part1;

import java.util.ArrayList;

import java.util.List;

class DeviceGroup implements SmartDevice {

private String name;

private List<SmartDevice> devices = new ArrayList<>();

public DeviceGroup(String name) {

this.name = name;

}

public void addDevice(SmartDevice device) {

devices.add(device);

}

@Override

public void turnOn() {

System.*out*.println("Включение всех устройств в " + name);

for (SmartDevice device : devices) {

device.turnOn();

}

}

@Override

public void turnOff() {

System.*out*.println("Выключение всех устройств в " + name);

for (SmartDevice device : devices) {

device.turnOff();

}

}

@Override

public String getStatus() {

StringBuilder status = new StringBuilder(name + " содержит:\n");

for (SmartDevice device : devices) {

status.append("- ").append(device.getStatus()).append("\n");

}

return status.toString();

}

}

### **2.2 Abstract Factory Pattern**

**Description:** Provides for the creation of families of related objects, ensuring a unified interface.

package part1;

*// Abstract Factory: Фабрика для создания устройств*

public interface SmartHomeFactory {

Light createLight(String name);

Camera createCamera(String name);

RobotVacuum createRobotVacuum(String name);

}

#### **Specific implementation BasicSmartHomeFactory**

package part1;

public class BasicSmartHomeFactory implements SmartHomeFactory {

@Override

public Light createLight(String name) {

return new Light(name);

}

@Override

public Camera createCamera(String name) {

return new Camera(name);

}

@Override

public RobotVacuum createRobotVacuum(String name) {

return new RobotVacuum(name);

}

}

### **2.3 Facade Pattern**

**Description:** Provides a single interface for controlling all devices.

package part1;

import java.util.ArrayList;

import java.util.List;

*// Facade: Контроллер умного дома*

public class SmartHomeController {

public List<SmartDevice> devices = new ArrayList<>();

public void addDevice(SmartDevice device) {

devices.add(device);

}

public void turnAllLightsOn() {

System.*out*.println("Включение всех светильников...");

for (SmartDevice device : devices) {

if (device instanceof Light) {

device.turnOn();

}

}

}

public void setGlobalTemperature(int temperature) {

System.*out*.println("Установка глобальной температуры: " + temperature + "°C (пример)");

}

public void getSystemStatusReport() {

System.*out*.println("Статус системы умного дома:");

for (SmartDevice device : devices) {

System.*out*.println(device.getStatus());

}

}

}

## **3. Conclusion**

The first part of the project implements the basic architecture and logic of the devices using **Composite, Abstract Factory** and **Facade** templates. The next stage includes **Decorator** and **Adapter** for advanced functionality.

# **Part 2 - Extended Functionality**

By Rashidov Aziz

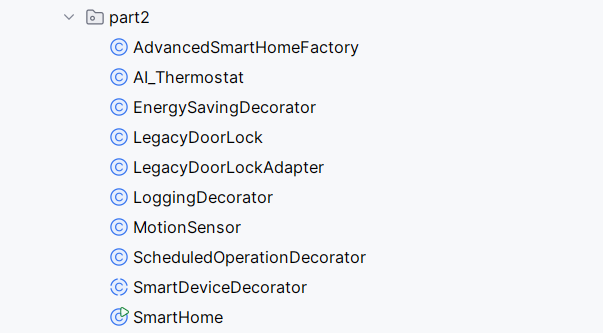
## **1. Introduction**

The second part of the **Smart Home Control System** extends the core architecture by adding advanced features using additional design patterns:

* **Decorator Pattern** (enhancing device functionality);
* **Adapter Pattern** (integrating legacy or third-party systems);
* **Advanced Factory** (creating high-end smart devices).

## **2. Code Structure**

The code is organized in the part2 package, containing the following key components:



### **2.1 Advanced Factory for Smart Devices**

**Description:** Extends the BasicSmartHomeFactory from Part 1, allowing the creation of AI-powered thermostats and motion sensors.

package part2;

import part1.BasicSmartHomeFactory;

public class AdvancedSmartHomeFactory extends BasicSmartHomeFactory {

public AdvancedSmartHomeFactory() {

super();

}

public AI\_Thermostat createAIThermostat(String name) {

return new AI\_Thermostat(name);

}

*// Новый метод для создания MotionSensor*

public MotionSensor createMotionSensor(String name) {

return new MotionSensor(name);

}

}

### **2.2 AI\_Thermostat Class**

**Description:** A smart thermostat that allows temperature control with AI-powered features.

package part2;

import part1.SmartDevice;

public class AI\_Thermostat implements SmartDevice {

private String name;

private int temperature;

public AI\_Thermostat(String name) {

this.name = name;

this.temperature = 25;

}

@Override

public void turnOn() {

System.*out*.println("Вы включили " + name + ".");

}

@Override

public void turnOff() {

System.*out*.println("Вы выключили " + name + ".");

}

@Override

public String getStatus() {

return "Температура на " + name +" установлена " + temperature + "C";

}

public void setTemperature(int temperature) {

this.temperature = temperature;

System.*out*.println("Вы установили температуру "+ temperature + "C");

}

}

### **2.3 Decorator Pattern (Enhancing Devices)**

**Description:** Decorators extend the behavior of smart devices without modifying their core implementation.

#### **EnergySavingDecorator**

package part2;

import part1.\*;

import java.time.LocalTime;

public class EnergySavingDecorator extends SmartDeviceDecorator {

private LocalTime startTime = LocalTime.*of*(00,00);

private LocalTime endTime = LocalTime.*of*(07,00);

private LocalTime currentTime = LocalTime.*now*();

public EnergySavingDecorator(SmartDevice device) {

super(device);

}

@Override

public void turnOn() {

if(currentTime.isAfter(startTime) && currentTime.isBefore(endTime)) {

device.turnOn();

System.*out*.println("Устройство включено в режиме энергосбережения.");

}

}

@Override

public void turnOff() {

if(!(currentTime.isAfter(startTime) && currentTime.isBefore(endTime))) {

device.turnOff();

System.*out*.println("Устройство выключено, активирован режим энергосбережения.");

}

}

}

#### **LoggingDecorator**

package part2;

import part1.\*;

public class LoggingDecorator extends SmartDeviceDecorator {

public LoggingDecorator(SmartDevice device) {

super(device);

}

@Override

public void turnOn() {

System.*out*.println("Logging: Включение устройства " + device.getStatus());

device.turnOn();

}

@Override

public void turnOff() {

System.*out*.println("Logging: Выключение устройства " + device.getStatus());

device.turnOff();

}

}

#### **ScheduledOperationDecorator**

package part2;

import part1.\*;

import java.time.LocalTime;

*// Decorator Pattern: Декоратор для расписания*

public class ScheduledOperationDecorator extends SmartDeviceDecorator {

private LocalTime startTime;

private LocalTime endTime;

public ScheduledOperationDecorator(SmartDevice device, LocalTime startTime, LocalTime endTime) {

super(device);

this.startTime = startTime; *// Время начала работы устройства*

this.endTime = endTime; *// Время окончания работы устройства*

}

@Override

public void turnOn() {

LocalTime currentTime = LocalTime.*now*();

if (currentTime.isAfter(startTime) && currentTime.isBefore(endTime)) {*//Если нынешнее время больше стартового и меньше конечного*

System.*out*.println("Включение устройства в разрешенное время.");

device.turnOn();

} else {

System.*out*.println("Устройство можно включить только в пределах расписания. это с " + startTime + "до " + endTime);

}

}

@Override

public void turnOff() {

device.turnOff();

}

}

### **2.4 Adapter Pattern (Integrating Legacy Systems)**

**Description:** The adapter allows integrating an old door lock system into the smart home ecosystem.

#### **LegacyDoorLock**

package part2;

public class LegacyDoorLock {

private boolean locked;

public LegacyDoorLock() {

this.locked = true;

}

public void lockDoor() {

this.locked = true;

System.*out*.println("Дверь заблокирована.");

}

public void unlockDoor() {

this.locked = false;

System.*out*.println("Дверь разблокирована.");

}

public boolean isLocked() {

return locked;

}

}

#### **LegacyDoorLockAdapter**

package part2;

import part1.SmartDevice;

*// Адаптер для старой системы замка двери*

public class LegacyDoorLockAdapter implements SmartDevice {

private LegacyDoorLock legacyDoorLock;

public LegacyDoorLockAdapter(LegacyDoorLock legacyDoorLock) {

this.legacyDoorLock = legacyDoorLock;

}

@Override

public void turnOn() {

legacyDoorLock.unlockDoor();

System.*out*.println("Замок двери разблокирован.");

}

@Override

public void turnOff() {

legacyDoorLock.lockDoor();

System.*out*.println("Замок двери заблокирован.");

}

@Override

public String getStatus() {

return "Замок двери: " + (legacyDoorLock.isLocked() ? "Заблокирован" : "Разблокирован");

}

}

## **3. Conclusion**

The second part of the **Smart Home Control System** introduces advanced device features through **Decorator** and **Adapter** patterns. It enhances energy efficiency, scheduling, logging, and integrates a legacy door lock system into the smart home ecosystem.

# **Conclusion: Smart Home Control System**

## **Overview**

The **Smart Home Control System** project demonstrates the application of multiple design patterns to create a scalable and flexible home automation architecture. The system was implemented in two parts, each focusing on different structural and functional aspects of the project.

## **Summary of Part 1: Core Architecture**

The first part established the foundational structure of the smart home system by implementing:

* **Composite Pattern**: Enabled hierarchical organization of smart devices into groups such as rooms and floors.
* **Abstract Factory Pattern**: Provided a structured way to create different types of smart devices with a consistent interface.
* **Facade Pattern**: Introduced a centralized controller (SmartHomeController) to simplify device management and interaction.

This part ensured that devices were modular, extendable, and manageable under a unified control system.

## **Summary of Part 2: Advanced Functionality**

The second part extended the system with additional capabilities through:

* **Decorator Pattern**: Allowed dynamic enhancement of devices with functionalities such as energy-saving modes, scheduling, and logging.
* **Adapter Pattern**: Integrated external and legacy components, such as a traditional door lock system, into the smart home ecosystem.
* **Advanced Smart Device Factory**: Introduced high-end devices like AI-powered thermostats and motion sensors, extending the capabilities of the original factory model.

This part ensured flexibility, integration with older systems, and enhanced automation features for smart home devices.

## **Overall Conclusion**

The **Smart Home Control System** successfully integrates multiple design patterns to create a modular, extensible, and maintainable system. By combining structural (Composite, Factory, Facade) and behavioral (Decorator, Adapter) design patterns, the system provides:

* **Scalability**: Easily adds new device types and functionalities.
* **Ease of Use**: The SmartHomeController simplifies interaction with devices.
* **Flexibility**: Devices can be enhanced dynamically without modifying core implementations.
* **Integration**: Legacy systems can be incorporated seamlessly.

This project serves as a robust example of how design patterns contribute to developing scalable and maintainable software architectures, particularly in IoT and smart home automation systems.