|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  высшего образования |
| **«Национальный исследовательский ядерный университет «МИФИ»**  **(НИЯУ МИФИ)** |

**Отчет**

**по результатам выполнения задания**

**демонстрационного экзамена**

|  |  |
| --- | --- |
| Студент | Ревель Руслан Сергеевич |
| Организация | МИФИ |
| Группа | М24-525 |
|  |  |
| Дата | 31.07.2025 |

Москва 2025

**Оглавление**

[I Диаграмма классов 3](#_Toc14712)

[II Реализация объявления классов 5](#_Toc27292)

[1. Интерфейсы (абстрактные классы) 5](#_Toc24756)

[2. Класс Device 5](#_Toc29138)

[3. Класс Room 5](#_Toc29814)

[4. Класс SmartSpeaker 5](#_Toc23043)

[5. Класс SmartHome 5](#_Toc29345)

[6. Контроллеры 6](#_Toc22501)

[Особенности реализации 6](#_Toc14832)

[III Реализация методов классов 7](#_Toc16572)

[Класс SmartHome (Умный дом) 7](#_Toc15934)

[Класс SmartSpeaker (Колонка) 7](#_Toc1528)

[Особенности реализации: 7](#_Toc19509)

[VI Скриншоты работы программы 8](#_Toc18094)

# **I Диаграмма классов**

Диаграмма для структуры моделей

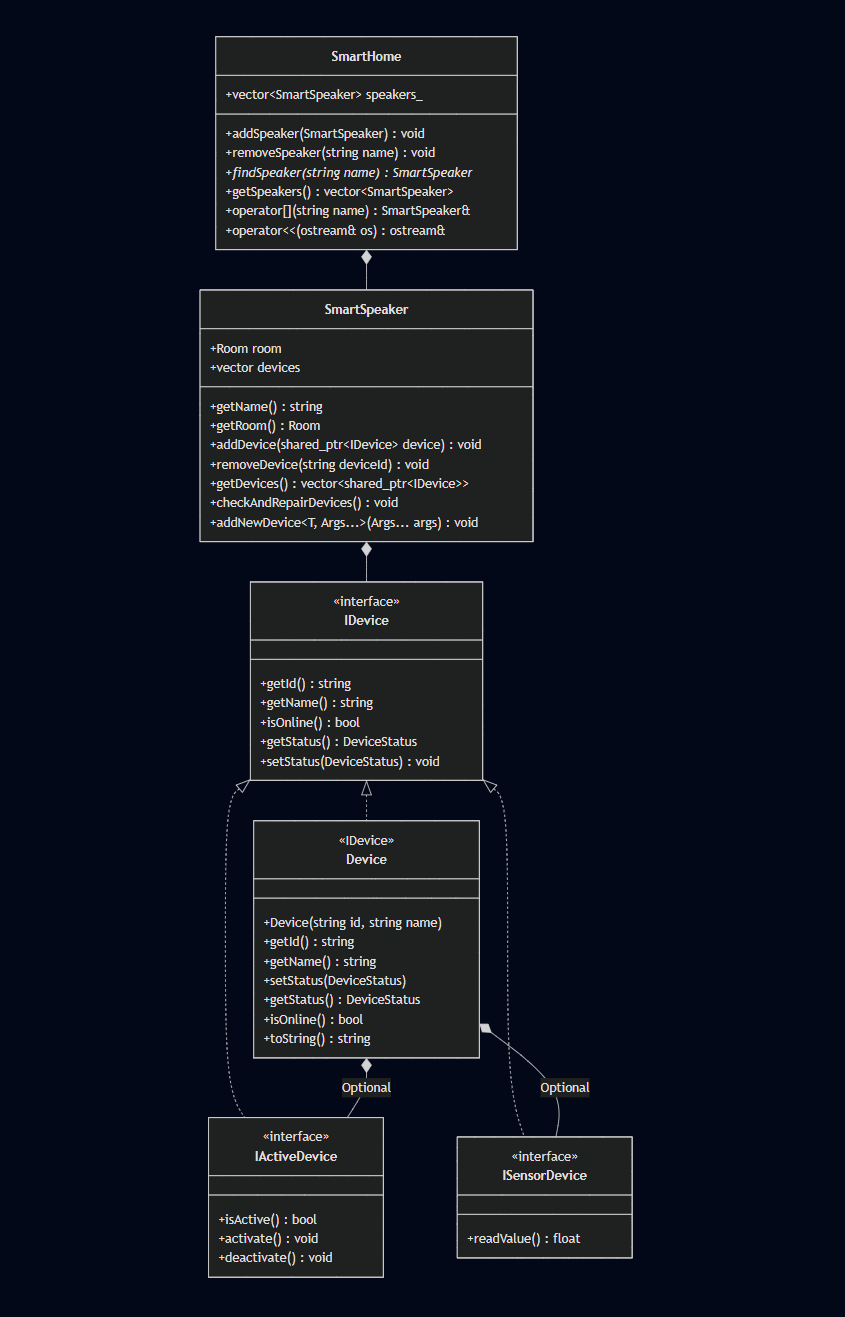
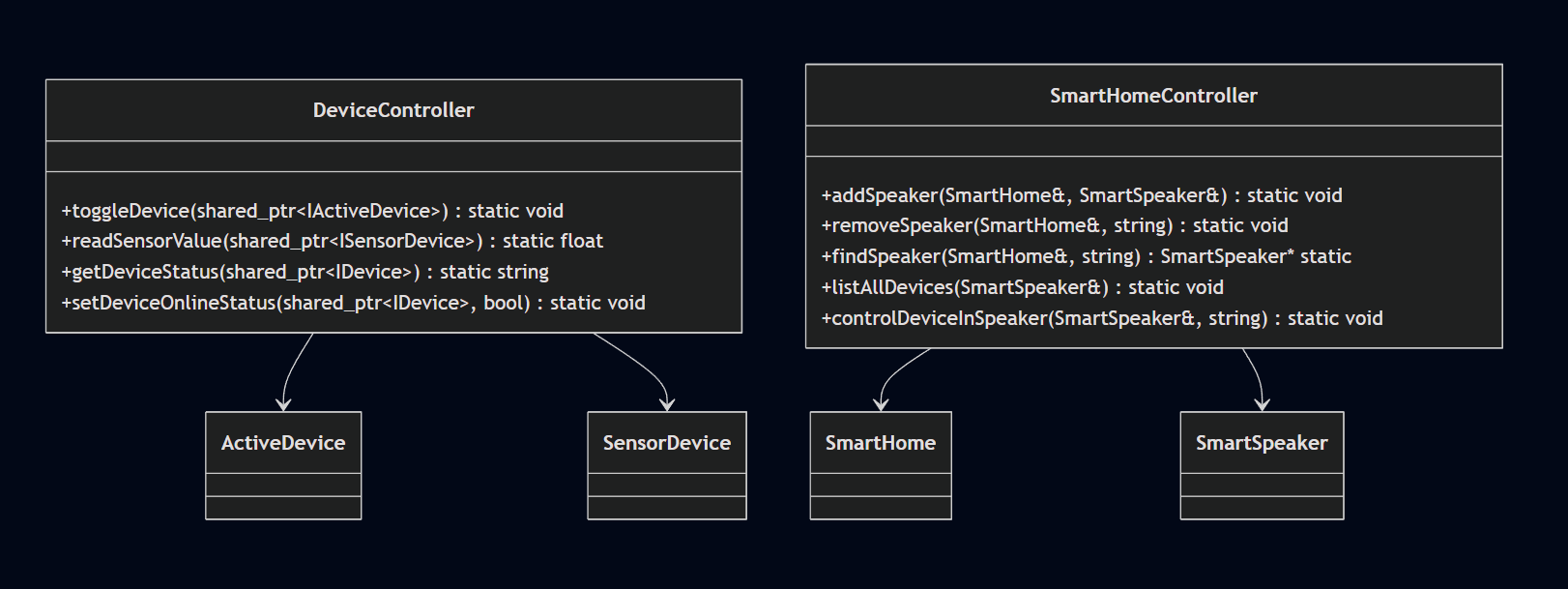


Диаграмма для структуры контроллеров



Полное подробное описания проекта есть в README

# **II Реализация объявления классов**

### **1. Интерфейсы (абстрактные классы)**

****IDevice**** - базовый интерфейс для всех устройств:

* Содержит методы для получения идентификатора, имени, проверки онлайн-статуса
* Позволяет получать и устанавливать статус устройства (Online, Offline, Error, Unknown)
* Виртуальный деструктор для корректного удаления производных классов

****IActiveDevice**** (наследует IDevice) - интерфейс для управляемых устройств:

* Добавляет методы для проверки активности (вкл/выкл)
* Методы активации и деактивации устройства

****ISensorDevice**** (наследует IDevice) - интерфейс для сенсорных устройств:

* Добавляет метод чтения значений с датчика

### **2. Класс Device**

Базовая реализация интерфейса IDevice:

* Содержит поля: id (строка), name (строка), status (перечисление DeviceStatus)
* Конструктор принимает id и name, устанавливает начальный статус Unknown
* Реализует все методы интерфейса IDevice
* Дополнительный метод toString() для строкового представления устройства

### **3. Класс Room**

Описывает комнату в умном доме:

* Поля: name (строка), number (целое число), type (перечисление RoomType)
* RoomType включает варианты: LivingRoom, Kitchen, Bedroom и др.
* Методы доступа к полям
* Метод getTypeString() возвращает строковое представление типа комнаты

### **4. Класс SmartSpeaker**

Управляет устройствами в конкретной комнате:

* Содержит объект Room и вектор умных указателей на IDevice
* Методы:
  + Доступа к имени и комнате
  + Добавления/удаления устройств
  + Получения списка устройств
  + Проверки и "ремонта" устройств
* Шаблонный метод addNewDevice для удобного добавления устройств разных типов

### **5. Класс SmartHome**

Управляет всеми колонками в доме:

* Содержит вектор объектов SmartSpeaker
* Основные методы:
  + Добавление/удаление колонок
  + Поиск колонки по имени
  + Получение списка всех колонок
* Перегруженные операторы:
  + [] для доступа к колонке по имени (с проверкой)
  + << для вывода информации о доме

### **6. Контроллеры**

****DeviceController**** (статические методы):

* Управление устройствами: переключение состояния, чтение сенсоров
* Получение и установка статусов устройств
* Обработка ошибок (например, при чтении оффлайн-сенсора)

****SmartHomeController**** (статические методы):

* Управление колонками: добавление, удаление, поиск
* Вывод списка устройств для конкретной колонки
* Управление устройствами через колонку (поиск и взаимодействие)
* Использует dynamic\_pointer\_cast для безопасного приведения к IActiveDevice

## **Особенности реализации**

****Использование умных указателей****:

* + shared\_ptr для безопасного управления устройствами
  + Гарантирует корректное освобождение ресурсов

****Шаблонные методы****:

* + Позволяют удобно добавлять устройства разных типов

****Обработка ошибок****:

* + Проверки статусов устройств
  + Исключения при неверных операциях

****Гибкая архитектура****:

* + Разделение интерфейсов и реализаций
  + Четкое разделение модели и контроллеров

****Интуитивный API****:

* + Перегруженные операторы для удобного использования
  + Статические методы контроллеров для простого доступа к функционалу

# **III Реализация методов классов**

### **Класс SmartHome (Умный дом)**

****1. Перегрузка оператора вывода в поток (**<<**):****

* Создается копия списка колонок для сортировки
* Колонки сортируются по имени с использованием лямбда-функции
* Выводится заголовок и отсортированный список колонок с их именами и названиями комнат
* Возвращается поток вывода для поддержки цепочки операций

****2. Перегрузка оператора индексации (**[]**):****

* Реализовано два варианта:
  + Неконстантный вариант возвращает ссылку на колонку для модификации
  + Константный вариант возвращает константную ссылку для чтения
* Оба варианта ищут колонку по имени с помощью find\_if
* Если колонка не найдена, генерируется исключение out\_of\_range
* При успешном поиске возвращается найденная колонка

### **Класс SmartSpeaker (Колонка)**

****1. Функция добавления устройства:****

* Проверяется наличие устройства с таким же ID через find\_if
* При обнаружении дубликата генерируется исключение invalid\_argument
* Если проверка пройдена, устройство добавляется в вектор devices\_

****2. Функция обхода и проверки устройств:****

* Для каждого устройства выполняется попытка преобразования в IActiveDevice
* Для активных устройств проверяется их состояние:
  + Если устройство выключено, оно включается (activate)
  + Увеличивается счетчик активированных устройств
  + Проверяется и корректируется онлайн-статус устройства
* Возвращается количество активированных устройств

### **Особенности реализации:**

В проекте реализована четкая архитектура с разделением ответственностей между компонентами. Класс SmartHomeCLI выступает как точка входа для пользовательского интерфейса, делегируя основную логику другим компонентам. Метод listAllSpeakers() Использует show\_speakers\_list, который получает объект SmartHome и выводит информацию через перегруженный оператор <<, обеспечивая стандартизированный вывод.

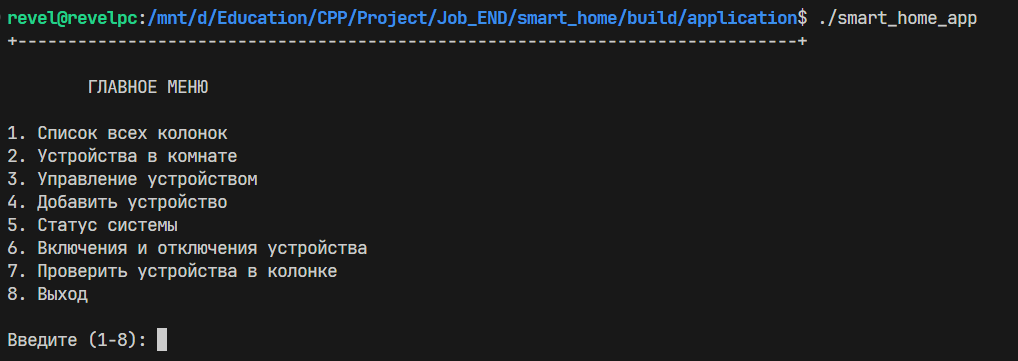
Для работы с устройствами реализована гибкая система: оператор [] в SmartHome позволяет напрямую обращаться к колонкам по имени, что используется в методах добавления устройств и их проверки. Однако основная работа с ведется через SmartHomeController, который предоставляет более высокоуровневые методы, такие как findSpeaker, абстрагируя детали реализации.

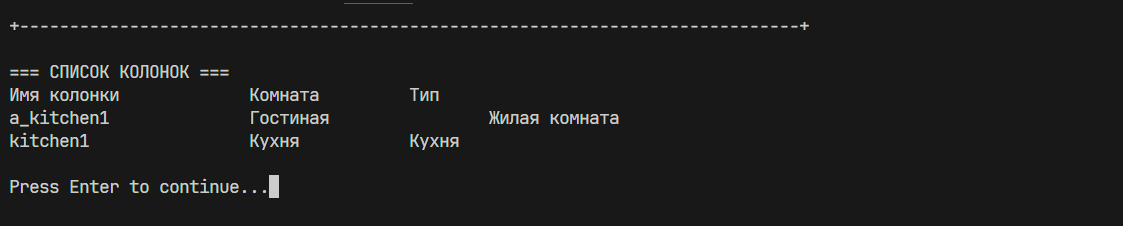
Особое внимание уделено расширяемости системы - шаблонный метод addNewDevice в SmartSpeaker позволяет добавлять любые устройства, наследующие от IDevice, без модификации основного кода. В проекте реализовано 5 типов устройств (SmartBulb, Thermometer, SmartKettle, LeakSensor, AirConditioner) с различными комбинациями функций, демонстрируя гибкость архитектуры.

Функция checkAndRepairDevices инкапсулирует логику проверки состояния устройств, вызываясь из пункта меню после нахождения соответствующей колонки. Хотя в идеале вывод информации стоило бы вынести в отдельные CLI-компоненты (что упростило бы поддержку разных интерфейсов), в рамках ТЗ реализован прямой вывод через стандартные потоки, как того требовало задание.

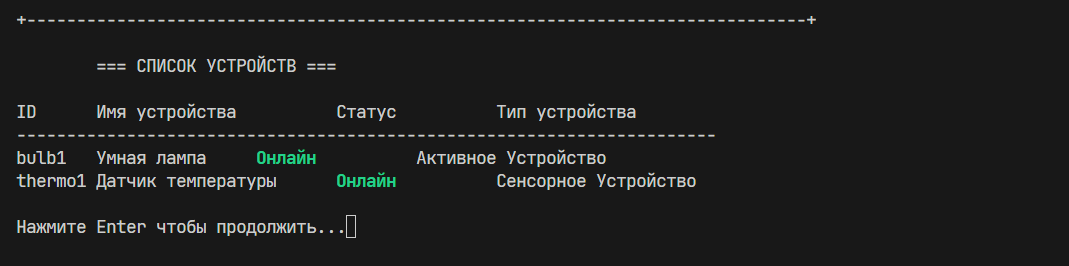
# **VI Скриншоты работы программы**

**Главное меню**

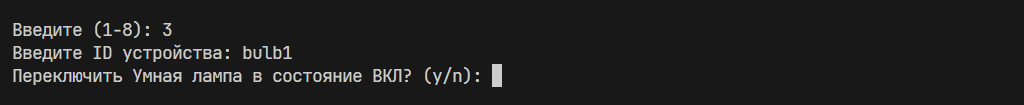


**Список умных колонок** 

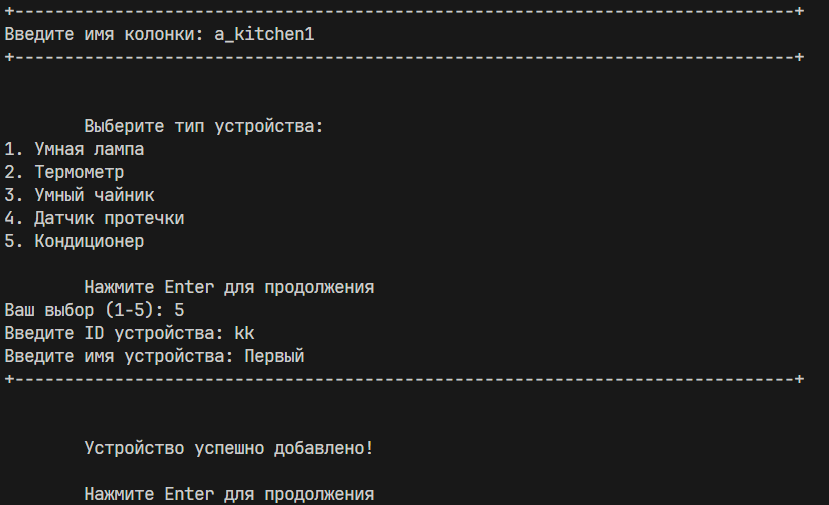
**Список устройств**



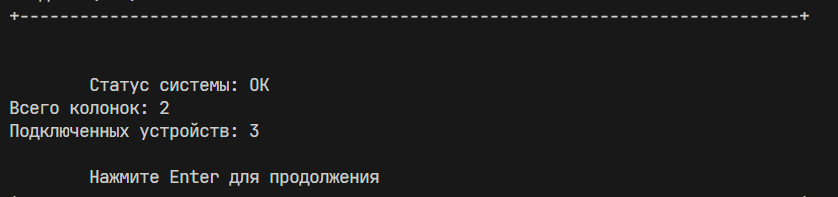
**Управления устройствами**



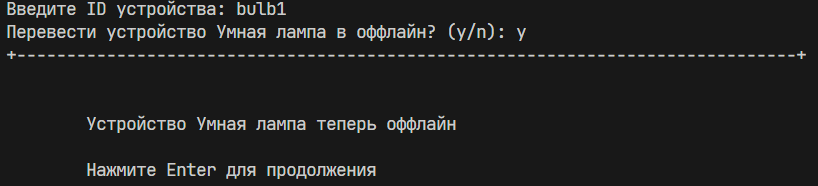
**Добавления устройства**



**Статус системы**



**Включения и отключения устройства**



**Проверить устройства в колонке**

