МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Казанский национальный исследовательский технический

университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт компьютерных технологий и защиты информации

*(наименование института (факультета)*

Кафедра динамики процессов и управления

*(наименование кафедры)*

Лабораторная работа № 3

По дисциплине «Программирование»

|  |  |
| --- | --- |
| Выполнил: | обучающийся группы\_4165\_ |
|  | Шестаков М.Д. |

Казань 2025 г.

**Задание 1**

**Цель:** реализовать минимальную архитектуру устройств и комнаты, продемонстрировать всё необходимое ООП-поведение.

**Требуется реализовать (минимум):**

1. abstract class Device
   * приватные поля: \_id (string), \_name (string), \_isOn (bool);
   * свойства:
     + public string Id { get; } — устанавливается в конструкторе, например Guid.NewGuid().ToString();
     + public string Name { get; init; } — нельзя null/пустую строку;
     + public bool IsOn { get; protected set; };
   * абстрактное свойство public abstract string Type { get; };
   * виртуальные методы public virtual void TurnOn() и public virtual void TurnOff() — по умолчанию переключают IsOn и пишут короткий лог; можно переопределять;
   * виртуальный метод public virtual string Status() — возвращает строку состояния.
2. interface IControllable
   * void TurnOn(); void TurnOff();.
3. Два подкласса: Light и Thermostat (наследуют Device и реализуют IControllable)
   * Light:
     + приватное поле \_brightness и свойство Brightness с валидацией;
     + переопределяет Type => "Light" и Status();
   * Thermostat:
     + приватное поле \_temperature и свойство Temperature с валидацией разумного диапазона;
     + реализует Type => "Thermostat" и Status().
4. class Room
   * приватный List<Device> \_devices;
   * публичное string Name { get; };
   * методы: void AddDevice(Device d), bool RemoveDevice(string id), IReadOnlyList<Device> GetDevices();
   * метод void TurnAllOn() — проходит по устройствам и вызывает TurnOn() через интерфейс IControllable (если устройство его реализует).
5. **Demo (Program.Main)**
   * создать комнату, добавить Light и Thermostat, вывести Status() для всех устройств, вызвать TurnAllOn(), снова вывести Status(). Сделать добавление устройств и комнат пользователем. Добавить команду, которая показывает состояния всех устройств во всех комнатах с группировкой по комнатам.

**Объяснение работы программы**

Целью задания было создать минимальную, но полноценную архитектуру системы «умного дома», демонстрирующую ключевые принципы объектно-ориентированного программирования: абстракцию, наследование, инкапсуляцию, полиморфизм и использование интерфейсов. В основе архитектуры лежит абстрактный базовый класс Device, который определяет общее поведение для всех устройств. У него есть приватные поля \_id, \_name и \_isOn, а также соответствующие свойства: Id устанавливается один раз в конструкторе (например, через Guid.NewGuid().ToString()), Name доступен только для инициализации и защищён от null или пустых значений, а IsOn может изменяться только внутри класса или его наследников через protected set. Также Device объявляет абстрактное свойство Type, которое обязаны реализовать все дочерние классы, и предоставляет виртуальные методы TurnOn() и TurnOff(), по умолчанию просто переключающие состояние IsOn и выводящие краткое сообщение в лог; эти методы можно переопределять для специфичного поведения. Дополнительно реализован виртуальный метод Status(), возвращающий строковое представление текущего состояния устройства. Для поддержки полиморфного управления устройствами введён интерфейс IControllable с методами TurnOn() и TurnOff(), который реализуют все управляемые устройства. На основе Device созданы два конкретных подкласса: Light и Thermostat. Light содержит приватное поле \_brightness и публичное свойство Brightness с валидацией (например, значение от 0 до 100), переопределяет Type как "Light" и предоставляет расширенную реализацию Status(), включающую яркость. Thermostat хранит текущую температуру в поле \_temperature, предоставляет свойство Temperature с проверкой на разумный диапазон (например, от -20 до 50 градусов), задаёт Type как "Thermostat" и также переопределяет Status(), отображая текущую температуру. Все устройства объединяются в комнаты через класс Room, который содержит приватный список \_devices, публичное имя Name и методы для управления устройствами: AddDevice() добавляет устройство, RemoveDevice() удаляет по идентификатору и возвращает true/false в зависимости от успеха, GetDevices() предоставляет только для чтения доступ ко всем устройствам. Особое внимание уделено методу TurnAllOn(), который проходит по всем устройствам в комнате и, используя проверку через is IControllable, вызывает TurnOn() только у тех, кто поддерживает управление — это демонстрирует безопасное применение полиморфизма и интерфейсов. В демонстрационной программе (Main) создаётся интерактивный сценарий: пользователь может добавлять комнаты и устройства (Light и Thermostat), система выводит текущие состояния всех устройств, затем вызывается TurnAllOn() для всех комнат, после чего состояния выводятся повторно. Также реализована команда, которая отображает все устройства во всех комнатах с группировкой по комнатам, наглядно показывая структуру системы. Весь код строится на принципах инкапсуляции (поля скрыты, доступ через свойства), наследования (Light и Thermostat наследуют общую логику от Device), полиморфизма (вызов TurnOn() через IControllable работает для любого совместимого устройства) и абстракции (Device и IControllable скрывают детали реализации). Такой подход обеспечивает читаемость, расширяемость (легко добавить новые типы устройств) и безопасность (валидация входных данных, защита внутреннего состояния), что делает архитектуру пригодной для дальнейшего развития.

**Задание 2**

**Цель:** добавить контроллер для автоматической логики (температурное правило), показать абстрактный базовый контроллер и интерфейс для сенсоров.

**Требуется реализовать:**

1. abstract class ControllerBase
   * приватное поле \_id; public string Id { get; };
   * защищённое поле или свойство protected IReadOnlyList<Room> Rooms { get; } или protected List<Room> (если нужно модифицировать в контроллере);
   * абстрактный метод public abstract void Execute();.
2. interface ISensor
   * метод bool TryRead(out decimal value); — безопасный способ получить числовое значение (для Thermostat реализовать этот интерфейс).
3. class HomeController : ControllerBase
   * конструктор принимает IEnumerable<Room>;
   * реализация Execute():
     + обходит Rooms → во всех устройствах ищет реализацию ISensor (через is ISensor s) и IControllable (через is IControllable c);
     + если Thermostat (или другой сенсор) сообщает value < threshold, то найти устройство типа Heater в той же комнате и вызвать TurnOn();
   * метод void Run() вызывает Execute() и логирует результат.
4. Heater : Device, IControllable
   * простая модель: TargetTemperature и логика включения; Type => "Heater".
5. **Demo:** создать две комнаты; в одной — Thermostat с Temperature = 18.0, Heater выключен; вызвать HomeController.Run() → Heater включается.

**Объяснение работы программы**

Целью задания было создать архитектуру автоматизированного управления «умным домом» с использованием принципов объектно-ориентированного программирования. Для этого была реализована иерархия классов и интерфейсов, обеспечивающая гибкость, расширяемость и безопасность. В основе лежит абстрактный базовый класс ControllerBase, который определяет общую структуру любого контроллера: он содержит уникальный идентификатор Id, задаваемый один раз при создании, и защищённое свойство Rooms — список комнат, с которыми работает контроллер. Также в ControllerBase объявлен абстрактный метод Execute(), который должен быть реализован в дочерних классах, обеспечивая единый интерфейс запуска логики. Для взаимодействия с датчиками введён интерфейс ISensor с методом bool TryRead(out decimal value), позволяющим безопасно получать числовые данные без генерации исключений — такой подход аналогичен стандартным методам вроде int.TryParse и особенно важен в системах, где датчики могут временно выходить из строя. Класс Thermostat реализует этот интерфейс, предоставляя текущую температуру. Конкретная логика автоматического управления реализована в классе HomeController, унаследованном от ControllerBase. Его конструктор принимает коллекцию комнат, а метод Execute() проходит по всем комнатам и всем устройствам в них, проверяя, реализуют ли устройства интерфейсы ISensor (для чтения данных) и IControllable (для управления). Если обнаружен датчик (например, Thermostat), и его значение ниже заданного порога (например, 20°C), контроллер ищет в той же комнате устройство типа Heater и вызывает у него метод TurnOn(). Вспомогательный метод Run() просто вызывает Execute() и логирует факт выполнения, что упрощает отладку и демонстрацию. Устройство Heater наследуется от базового класса Device и реализует интерфейс IControllable: оно имеет свойство TargetTemperature и метод TurnOn(), изменяющий его состояние, а также возвращает строку "Heater" через свойство Type для идентификации. В демонстрационном сценарии создаются две комнаты, в одной из которых размещены Thermostat с температурой 18.0°C и выключенный Heater. После создания экземпляра HomeController и вызова его метода Run() контроллер автоматически обнаруживает низкую температуру, находит обогреватель в той же комнате и включает его, что подтверждает корректную работу всей системы. Такая архитектура обладает рядом преимуществ: она гибкая (легко добавлять новые типы устройств), тестируемая (благодаря интерфейсам), расширяемая (можно создавать другие контроллеры, например, для освещения или безопасности) и устойчивая к ошибкам (благодаря безопасному чтению с датчиков). В целом, решение демонстрирует применение ключевых принципов проектирования ПО — абстракции, инкапсуляции, полиморфизма и разделения ответственности — в контексте реальной задачи автоматизации.