# Технічна документація: Smart Home Automation System

**Студент**: Луханін Богдан Сергійович , 244(Б)
**Дисципліна**: Об'єктно-орієнтоване програмування (C#)
**Технології / стек**: .NET 9.0, C# 13, Console Application

## 1. Анотація проєкту

**Smart Home Automation System** - це програмна платформа для централізованого управління розумними пристроями в будинку. Система моделює взаємодію між контролером та різнорідними приладами (освітлення, клімат-контроль, побутова техніка), забезпечуючи моніторинг енергоспоживання та виконання специфічних сценаріїв.

**Ключова мета:** Реалізація розширюваної архітектури з використанням просунутих можливостей C# (.NET), таких як **Узагальнення (Generics)**, **Переліки (Enums)** та **LINQ - що знадобляться мені особисто, як виконавцю, для другого завдання - з Entity Framework**.

*P.S. Саме тому основні моменти я захотів зробити саме в окремому завданні, де можу придумати власні приклади і вже до них підбирати функціонал, який буде реалізовано надалі(щоб точно не наробити помилок🙃 мені вистачило хаха).*

*Дякую за увагу, пане Юрію!*

## 2. Архітектурне Рішення

Система спроєктована за модульним принципом, де кожен клас відповідає за свою зону відповідальності (Single Responsibility Principle).

### 2.1. Ієрархія Класів (Class Hierarchy)

В основі лежить поліморфна модель даних:

1. `SmartDevice` **(Abstract Base Class):**
   - Виступає "скелетом" для всіх приладів.
   - Містить фундаментальні властивості: `Name` (Ім'я), `PowerUsage` (Споживання), `IsOn` (Стан).
   - Реалізує базову логіку перемикання живлення ( `TogglePower` ).
   - Визначає абстрактний контракт `PerformTask()`, який зобов'язані реалізувати всі нащадки.
2. `SmartLight` **(Concrete Class):**
   - Реалізує логіку освітлення. Додає параметр `Brightness` (Яскравість).
   - *Поведінка:* При виконанні задачі змінює яскравість залежно від стану живлення.
3. `AirConditioner` **(Concrete Class):**
   - Складний пристрій зі змінним станом.
   - *Унікальна логіка:* Впроваджено режим `EcoMode`. При активації споживання енергії знижується вдвічі, що демонструє динамічну зміну стану об'єкта.
4. `SmartCoffeeMachine` **(Concrete Class):**
   - Пристрій з управлінням ресурсами ( `WaterLevel` , `BeanLevel` , `MilkLevel` ).
   - Використовує **Enum** для вибору типу напою, запобігаючи помилкам при введенні текстових команд.

## 2.2. Центр Управління (Controller Layer)

- `HomeController` **:**
  - Виконує роль "хаба". Зберігає всі пристрої в єдиній поліморфній колекції `List<SmartDevice>` .
  - Забезпечує агрегацію даних (підрахунок загальної енергії через LINQ).

- Надає методи для безпечного доступу до конкретних функцій пристроїв через Generics.

---

# 3. Технічна Реалізація та ООП

У проєкті застосовано повний спектр парадигм об'єктно-орієнтованого програмування:

## 3.1. Узагальнення - дженерикі

Для уникнення небезпечного приведення типів (Unsafe Casting) та дублювання коду, в контролері реалізував узагальнений метод:

```
public T GetDevice<T>(string name) where T : SmartDevice
{
   return devices.FirstOrDefault(d ⇒ d.Name == name) as T;
}
```

**Перевага:** Це дозволяє клієнтському коду (Program.cs) запитувати конкретний тип пристрою (наприклад, `AirConditioner`), отримуючи доступ до його унікальних методів (`ToggleEcoMode`), зберігаючи при цьому строгу типізацію.

## 3.2. Переліки

Для стандартизації вхідних даних використано тип `Enum`:

```
public enum CoffeeType { Espresso, Americano, Latte }
```

**Перевага:** Усуває проблему "магічних рядків" (Magic Strings). Користувач не може помилитися в назві напою, оскільки вибирає зі строго визначеного списку.

## 3.3. Поліморфізм

Метод `PerformTask()` є віртуальним (абстрактним). Контролер викликає цей метод для списку пристроїв, не знаючи їх конкретного типу.

- Лампа — світить.

- Кавомашина — варить каву (за замовчуванням еспресо).
- Кондиціонер — охолоджує.
- *Результат:* Система легко розширюється. Додавання нового пристрою (наприклад, `SmartTV`) не вимагає зміни коду контролера.

### 3.4. Інкапсуляція та Валідація

- Властивість `Name` захищена від запису пустих значень (викидається `ArgumentException`).
- Рівень води в кавомашині (`WaterLevel`) можна змінити тільки через методи `MakeCoffee()` або `Refill()`, прямий доступ на запис закритий (`private set`).

---

## 4. Алгоритмічна логіка

### 4.1. Логіка приготування кави

Алгоритм `MakeCoffee` перевіряє наявність ресурсів перед виконанням дії:

1. Перевірка живлення (`IsOn`).
2. Визначення рецепта через `switch` по `CoffeeType`.
3. Перевірка наявності інгредієнтів (Вода, Зерна, Молоко).
4. Якщо ресурсів достатньо — списання та приготування.
5. Якщо ні — вивід попередження без виконання дії.

### 4.2. Підрахунок енергоспоживання

Замість громіздких циклів використано декларативний підхід LINQ:

```
return devices.Where(d ⇒ d.IsOn).Sum(d ⇒ d.PowerUsage);
```

Це забезпечує миттєвий розрахунок навантаження на мережу, враховуючи лише активні прилади.

---

## 5. UML Діаграма Класів

*Візуалізація структури:*

- `SmartDevice` (Abstract)
  - Inherits: `SmartLight`
  - Inherits: `AirConditioner`
  - Inherits: `SmartCoffeeMachine` ⇒Uses `CoffeeType` (Enum)
- `HomeController` ⇒ Aggregates `List<SmartDevice>`

---

# 6. Інструкція з використання

## Ініціалізація та додавання

```
HomeController home = new HomeController();
home.AddDevice(new SmartLight("Kitchen", 15, 100));
home.AddDevice(new AirConditioner("Main AC", 2000, 24));
```

## Робота з унікальним функціоналом (через Generics)

```
// Отримуємо доступ до специфічних функцій кондиціонера
var ac = home.GetDevice<AirConditioner>("Main AC");
if (ac != null)
{
   ac.ToggleEcoMode(); // Зменшуємо споживання енергії
}
```

## Сценарне управління

```
// Одночасний запуск задач для всіх пристроїв
home.ActivateTask("Kitchen");  // Лампа увімкнеться
home.ActivateTask("Main AC");  // Кондиціонер почне охолоджувати
```

---

# 7. Висновки

Розроблена система демонструє високий рівень абстракції та гнучкості. Використання **Generics** дозволило створити типобезпечний контролер, а **Enums** підвищили надійність коду. Проєкт повністю відповідає принципам SOLID, зокрема принципу відкритості/закритості (Open/Closed Principle), оскільки додавання нових типів пристроїв не ламає існуючу логіку.