



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 4
із дисципліни «Технології розробки програмного забезпечення»
Тема: «Вступ до паттернів проектування.»

Виконав
Студенти групи ІА-31:
Корнійчук М.Р

Перевірив:
Мягкий М.Ю

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на гіт: https://github.com/MkEger/Lab4_trpz.git

Варіант :

3. Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати один з розглянутих шаблонів за обраною темою.
- Реалізувати не менше 3-х класів відповідно до обраної теми. •

Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Хід роботи

Тема проекту: Текстовий редактор

1. Реалізувати не менше 3-х класів відповідно до обраної теми

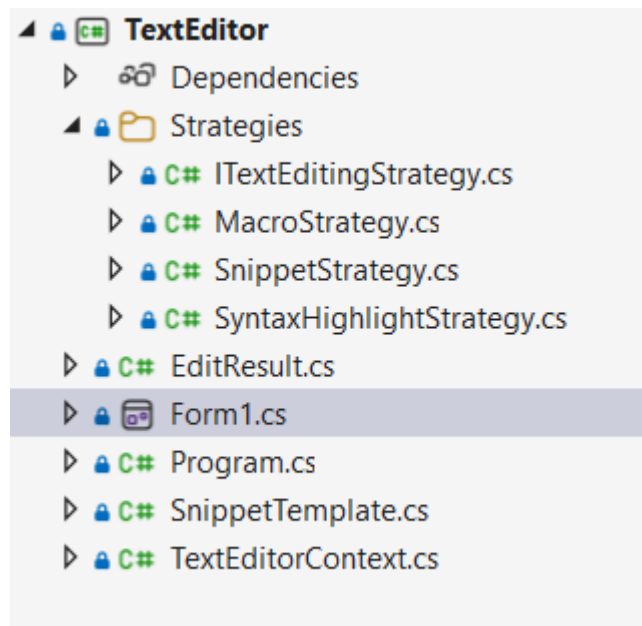


Рис. 1 - Структура проекту

У процесі виконання лабораторної роботи було створено п'ять основних класів, які забезпечують реалізацію функціоналу текстового редактора з використанням шаблону Strategy (рис. 1). Наведемо детальний опис трьох ключових класів:

1. ITextEditingStrategy (інтерфейс)

Центральний інтерфейс шаблону Strategy, який визначає контракт для всіх стратегій редагування тексту. Містить у собі такі властивості та методи:

- Name - назва стратегії для відображення в користувацькому інтерфейсі
- Description - детальний опис функціональності стратегії
- Execute() - основний метод для виконання операції редагування над текстом з параметрами вихідного тексту, позиції виділення та додатковими параметрами

Інтерфейс також включає допоміжний клас EditResult, який інкапсулює результат виконання операції, включаючи модифікований текст, нову позицію курсора, статус успішності та повідомлення про результат. Використовується як основа для створення всіх конкретних стратегій редагування.

2. MacroStrategy

Конкретна реалізація стратегії для роботи з макросами - попередньо визначеними фрагментами коду. Клас містить внутрішній словник `_macros` з готовими шаблонами для різних мов програмування (HTML, C#, JavaScript) та конструкцій (класи, методи, цикли, коментарі).

Основні можливості:

- Вставка готових фрагментів коду за назвою (наприклад, "class", "method", "for")
- Динамічне додавання нових макросів через метод AddMacro()
- Отримання списку доступних макросів через GetAvailableMacros()
- Видалення макросів та отримання їх вмісту

Стратегія забезпечує швидку вставку часто використовуваних конструкцій коду без необхідності їх ручного набору.

3. TextEditorContext

Контекстний клас, який реалізує основну логіку шаблону Strategy. Управляє колекцією доступних стратегій та забезпечує їх виконання. Містить внутрішній словник _strategies для зберігання всіх зареєстрованих стратегій та посилення на поточну активну стратегію _currentStrategy.

Основний функціонал:

- SetStrategy() - динамічна зміна поточної стратегії
- ExecuteOperation() - виконання операції з поточною стратегією
- ExecuteWithStrategy() - виконання операції з конкретною стратегією
- AddStrategy() та RemoveStrategy() - управління набором доступних стратегій
- GetStrategiesInfo() - отримання інформації про всі зареєстровані стратегії

Клас забезпечує єдиний інтерфейс для роботи з різними алгоритмами редагування та дозволяє легко розширювати функціональність системи додаванням нових стратегій.

Ця структура класів забезпечує модульність і гнучкість роботи з різними типами редагування тексту, а також спрощує розширення функціоналу редактора у майбутньому.

2. Реалізувати один з розглянутих шаблонів за обраною темою

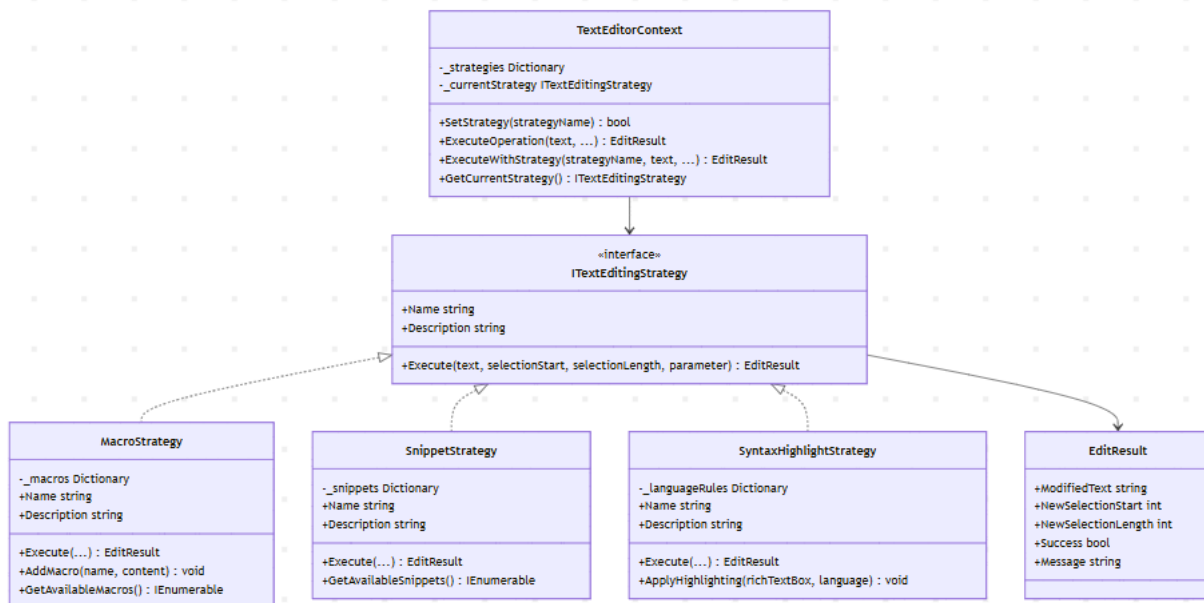


Рис. 2 - Діаграма класів

У рамках проєкту текстового редактора патерн Strategy було реалізовано для забезпечення гнучкого та розширюваного механізму редагування тексту. Його використання дозволило створити уніфікований спосіб роботи з різними алгоритмами обробки тексту, незалежно від їхньої конкретної реалізації.

Реалізація

Основна ідея полягала у створенні стратегій, які дозволяють працювати з текстом у різних режимах: вставка макросів, розгортання сніпетів та підсвічування синтаксису.

- Контекст (TextEditorContext) виступає координатором, який управляє набором стратегій та забезпечує їх виконання. Він створює відповідну стратегію залежно від обраного режиму редагування.
- Стратегії інкапсулюють логіку конкретного типу редагування, приховуючи деталі реалізації від основної програми редактора.

Проблеми, які вирішує патерн

- Інкапсуляція алгоритмів редагування. Завдяки стратегіям розробники можуть працювати з різними типами редагування через єдиний інтерфейс без необхідності знати їхню внутрішню реалізацію.
- Уніфікованість виконання операцій. Незалежно від типу стратегії (макроси, сніпети, підсвічування), методи доступу (Execute()) залишаються однаковими, що спрощує взаємодію з системою.
- Легкість додавання нових типів редагування. Завдяки базовому інтерфейсу

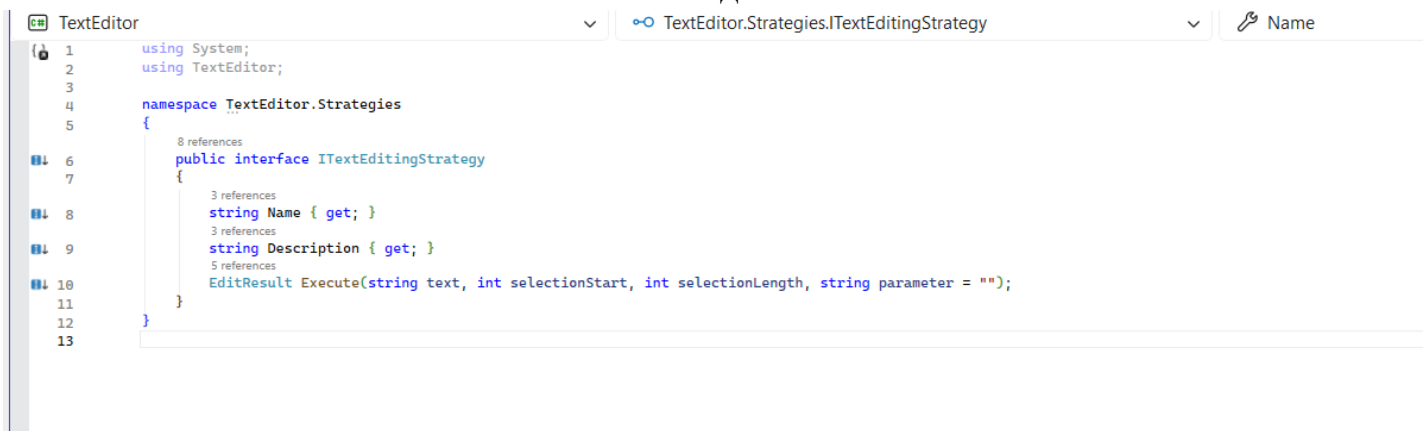
ITextEditingStrategy і можливості створення нових реалізацій, легко додавати нові функції (наприклад, автозаповнення або форматування коду), не змінюючи існуючий код.

- Покращення читабельності та підтримки коду. Відокремлення логіки кожного типу редагування від основного функціоналу редактора дозволяє створювати код, який легше підтримувати, тестувати та модифікувати.

Переваги використання

- Модульність: Патерн забезпечує чітке розмежування відповідальності між управлінням стратегіями (контекст) і їхнім виконанням (конкретні стратегії).
- Гнучкість: Додаючи новий тип стратегії, можна розширити функціонал редактора без змін в існуючій логіці управління.

Код:



```
1 using System;
2 using TextEditor;
3
4 namespace TextEditor.Strategies
5 {
6     public interface ITextEditingStrategy
7     {
8         string Name { get; }
9         string Description { get; }
10        EditResult Execute(string text, int selectionStart, int selectionLength, string parameter = "");
11    }
12 }
13
```

Рис. 3.1 – ItextEditingStrategy



```
4
5 namespace TextEditor.Strategies
6 {
7     public class MacroStrategy : ITextEditingStrategy
8     {
9         public string Name => "Макроси";
10        public string Description => "Вставка попередньо записаних фрагментів коду";
11
12        private readonly Dictionary<string, string> _macros;
13
14        public MacroStrategy()
15        {
16            _macros = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)
17            {
18                { "class", "public class ClassName{\n    public ClassName()\n    {\n        \n    }\n}" },
19                { "method", "public void MethodName()\n{\n    \n}" },
20                { "property", "public string PropertyName { get; set; }" },
21                { "for", "for (int i = 0; i < length; i++)\n{\n    \n}" },
22                { "if", "if (condition)\n{\n    \n}" },
23                { "try", "try{\n    \n}\ncatch (Exception ex)\n{\n    \n}" }
24            };
25        }
26
27        public EditResult Execute(string text, int selectionStart, int selectionLength, string parameter = "")
28        {
29            if (string.IsNullOrEmpty(parameter))
30            {
31                var availableMacros = string.Join(" ", _macros.Keys);
32                return new EditResult(text, selectionStart, selectionLength, false,
33                    $"Вкажіть ім'я макроса. Доступні: {availableMacros}");
34            }
35        }
36    }
37 }
```

```

35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
    if (!_macros.ContainsKey(parameter))
    {
        var availableMacros = string.Join(" ", _macros.Keys);
        return new EditResult(text, selectionStart, selectionLength, false,
            $"Макрос '{parameter}' не знайдено. Доступні: {availableMacros}");
    }

    string macroContent = _macros[parameter];
    string beforeSelection = text.Substring(0, selectionStart);
    string afterSelection = text.Substring(selectionStart + selectionLength);
    string newText = beforeSelection + macroContent + afterSelection;

    int newSelectionStart = selectionStart + macroContent.Length;

    return new EditResult(newText, newSelectionStart, 0, true,
        $"Макрос '{parameter}' вставлено успішно");
}

0 references
public void AddMacro(string name, string content)
{
    if (!string.IsNullOrEmpty(name))
    {
        _macros[name] = content ?? string.Empty;
    }
}

0 references
public IEnumerable<string> GetAvailableMacros()
{
    return _macros.Keys;
}
}

```

Рис. 3.2 – MacroStrategy

```

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
using TextEditor.Strategies;

namespace TextEditor
{
    2 references
    public class TextEditorContext
    {
        private readonly Dictionary<string, ITextEditingStrategy> _strategies;
        private ITextEditingStrategy _currentStrategy;

        1 reference
        public TextEditorContext()
        {
            _strategies = new Dictionary<string, ITextEditingStrategy>(StringComparer.OrdinalIgnoreCase)
            {
                { "macro", new MacroStrategy() },
                { "snippet", new SnippetStrategy() }
            };
            _currentStrategy = _strategies["macro"];
        }

        0 references
        public bool SetStrategy(string strategyName)
        {
            if (_strategies.ContainsKey(strategyName))
            {
                _currentStrategy = _strategies[strategyName];
                return true;
            }
            return false;
        }

        0 references
        public EditResult ExecuteOperation(string text, int selectionStart, int selectionLength, string parameter = "")
        {
            if (_currentStrategy == null)
            {
                return new EditResult(text, selectionStart, selectionLength, false, "Стратегія не вибрана");
            }

            try
            {
                return _currentStrategy.Execute(text, selectionStart, selectionLength, parameter);
            }
            catch (Exception ex)
            {
                return new EditResult(text, selectionStart, selectionLength, false,
                    $"Помилка виконання операції: {ex.Message}");
            }
        }

        public EditResult ExecuteWithStrategy(string strategyName, string text, int selectionStart, int selectionLength, string parameter = "")
        {
            if (!_strategies.ContainsKey(strategyName))
            {
                var availableStrategies = string.Join(" ", _strategies.Keys);
                return new EditResult(text, selectionStart, selectionLength, false,
                    $"Стратегія '{strategyName}' не знайдено. Доступні: {availableStrategies}");
            }

            try
            {
                return _strategies[strategyName].Execute(text, selectionStart, selectionLength, parameter);
            }
        }
    }
}

```

```

66         catch (Exception ex)
67         {
68             return new EditResult(text, selectionStart, selectionLength, false,
69                 $"Помилка виконання операції: {ex.Message}");
70         }
71     }
72
73     0 references
74     public ITextEditingStrategy GetCurrentStrategy()
75     {
76         return _currentStrategy;
77     }
78
79     0 references
80     public IEnumerable<string> GetStrategyNames()
81     {
82         return _strategies.Keys;
83     }
84
85     0 references
86     public void AddStrategy(string name, ITextEditingStrategy strategy)
87     {
88         if (!string.IsNullOrEmpty(name) && strategy != null)
89         {
90             _strategies[name] = strategy;
91         }
92     }

```

Рис. 3.3 – TextEditorContext

Висновок: У ході роботи було створено проєкт текстового редактора з використанням шаблону Strategy. Реалізовано основні класи: ITextEditingStrategy - інтерфейс для визначення спільної поведінки стратегій редагування, MacroStrategy - стратегія для вставки готових макросів, та TextEditorContext - клас, що керує вибором і виконанням стратегій. Застосування патерну дозволило зробити систему гнучкою, модульною та зручною для розширення. Також розроблено діаграму класів, яка відобразила структуру та взаємозв'язки між елементами системи.

Відповіді на контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування - це типове, перевірене рішення певної задачі проєктування програмного забезпечення, яке можна повторно використовувати у різних проєктах.

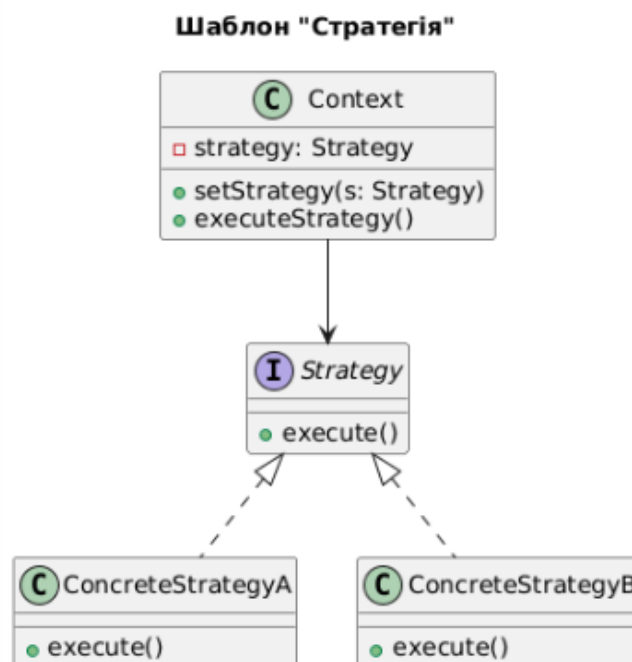
2. Навіщо використовувати шаблони проєктування?

Вони допомагають уникати типових помилок, спрощують розробку, покращують читабельність і підтримку коду, а також забезпечують гнучкість і повторне використання рішень.

3. Яке призначення шаблону «Стратегія»?

Шаблон «Стратегія» дозволяє визначити сімейство алгоритмів, інкапсулювати кожен з них і робити їх взаємозамінними без зміни клієнтського коду.

4. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

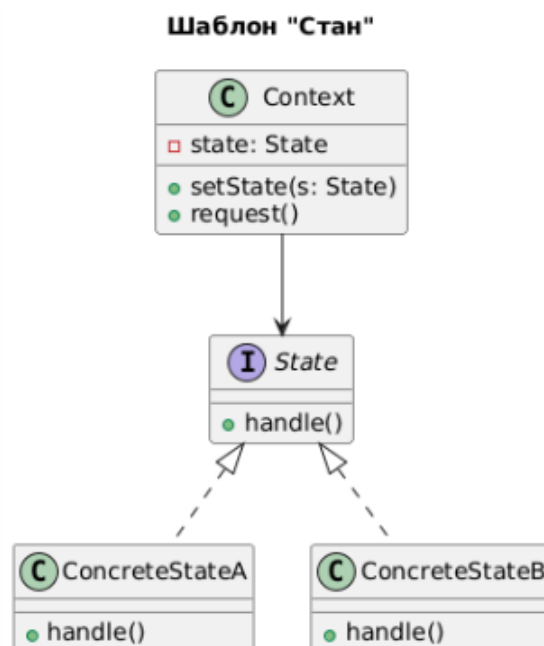
- Context - зберігає посилання на об'єкт стратегії.
- Strategy (інтерфейс) - оголошує метод, який реалізують стратегії.
- ConcreteStrategy - конкретні реалізації алгоритму.

Взаємодія: Context викликає метод Strategy, не знаючи, який саме алгоритм використовується.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об'єкту змінювати свою поведінку при зміні внутрішнього стану, створюючи ілюзію зміни його класу.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

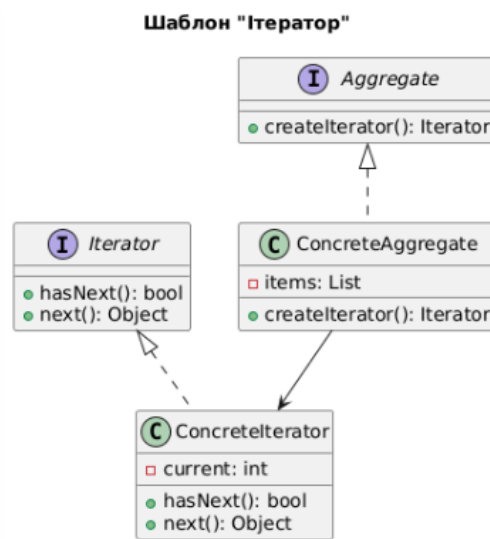
- Context - зберігає поточний стан і делегує йому поведінку.
- State (інтерфейс) - визначає інтерфейс для станів.
- ConcreteState - реалізує конкретні варіанти поведінки.

Взаємодія: Context викликає метод поточного стану, а стан може змінювати Context на інший стан.

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» дає змогу послідовно отримувати доступ до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

- **Iterator** (інтерфейс) - визначає інтерфейс для обходу елементів.
- **ConcreteIterator** - реалізує цей інтерфейс для конкретної колекції.
- **Aggregate** (інтерфейс) - створює об'єкт ітератора.
- **ConcreteAggregate** - реалізує **Aggregate** і зберігає колекцію.

Взаємодія: Клієнт отримує ітератор від колекції та використовує його для послідовного доступу до елементів.

12. В чому полягає ідея шаблону «Одинак»?

Ідея полягає в тому, щоб клас мав лише один екземпляр (instance) у програмі та забезпечував глобальну точку доступу до нього.

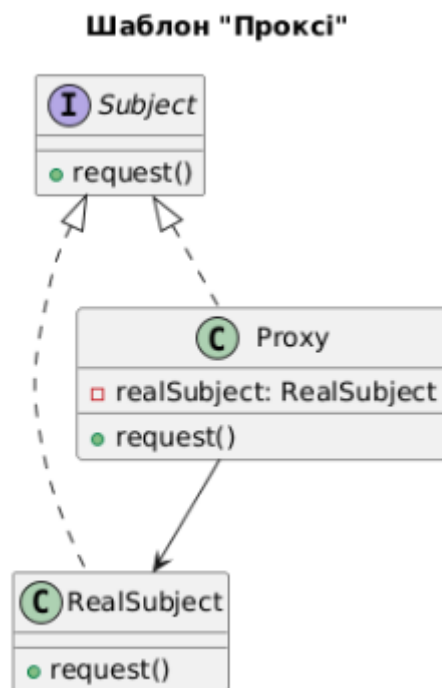
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Тому що він порушує принципи чистої архітектури — створює глобальний стан, ускладнює тестування, приховано зв'язує частини програми та знижує гнучкість.

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» використовується для контролю доступу до об'єкта, наприклад, для відкладеного створення, кешування або безпеки.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

- **Subject** (інтерфейс) - оголошує спільний інтерфейс для **Proxy** і **RealSubject**.
- **RealSubject** - реальний об'єкт, до якого здійснюється доступ.
- **Proxy** - зберігає посилання на **RealSubject** і контролює звернення до нього.

Взаємодія: Client викликає метод Proxy, Proxy вирішує — передати запит RealSubject чи виконати додаткову логіку.