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

Лабораторна робота № 8

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Патерни проектування»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Патерни проектування

Тема проєкту: 28. JSON Tool (ENG) (strategy, command, observer, template method,flyweight) Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Autosave\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1) Ознайомитись з короткими теоретичними відомостями

Шаблон «Composite»

Призначення: Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Даний шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю .

Простим прикладом може служити складання компонентів всередині звичайної форми. Форма може містити дочірні елементи (поля для введення тексту, цифр, написи, малюнки тощо); дочірні елементи можуть в свою чергу містити інші елементи. Наприклад, при виконанні операції розтягування форми необхідно, щоб вся ієрархія розтягнулася відповідним чином. В такому випадку 90 форма розглядається як композитний об'єкт і операція розтягування застосовується до всіх дочірніх елементів рекурсивно.

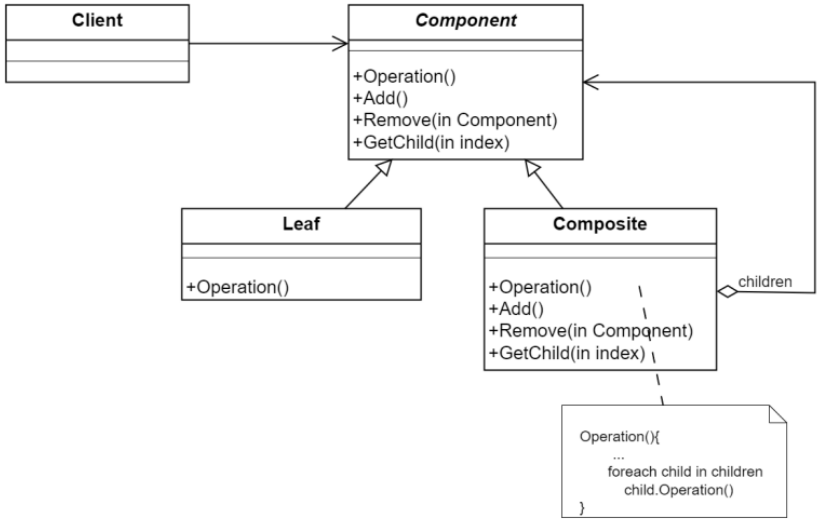


Рисунок 8.1. Структура патерна «Компонувальник»

Шаблон «Flyweight»

Призначення: Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт. Дуже важливою є концепція «внутрішнього» і «зовнішнього» станів [6]. Внутрішній стан відображає дані, характерні саме поділюваному об'єкту 92 (наприклад, код букви); зовнішній стан несе інформацію про його застосування в додатку (наприклад, рядок і стовпчик). Внутрішній стан зберігається в самому поділюваному об'єкті, зовнішній – в об'єктах додатку (контексту використання поділюваного об'єкта).

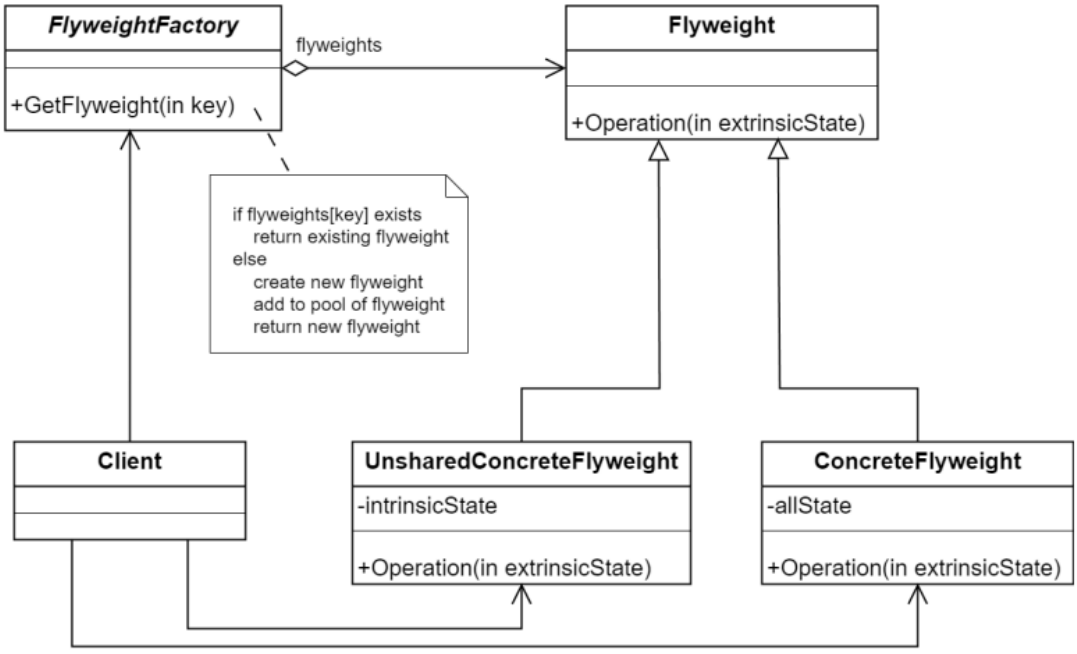


Рисунок 8.2. Структура патерну Flyweight (Легковаговик)

Шаблон «Interpreter»

Призначення: Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової) [6]. Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції.

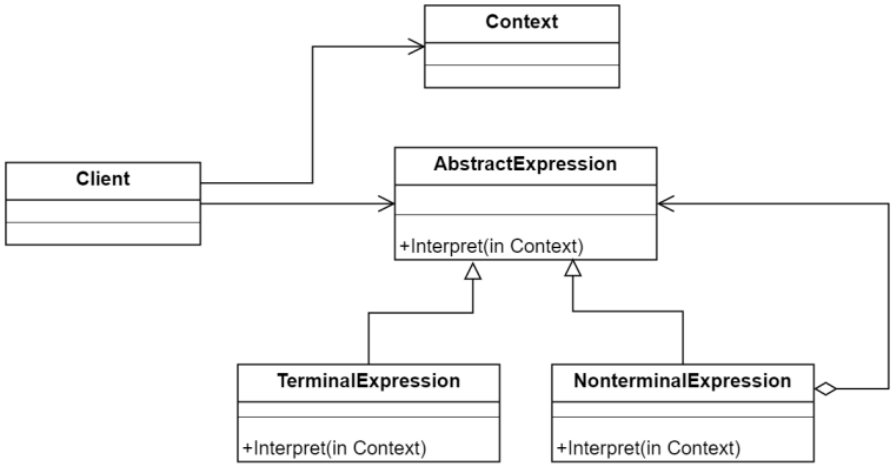


Рисунок 8.4. Структура патерна «Інтерпретатор»

Шаблон «Visitor» Призначення: Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів [6]. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача)

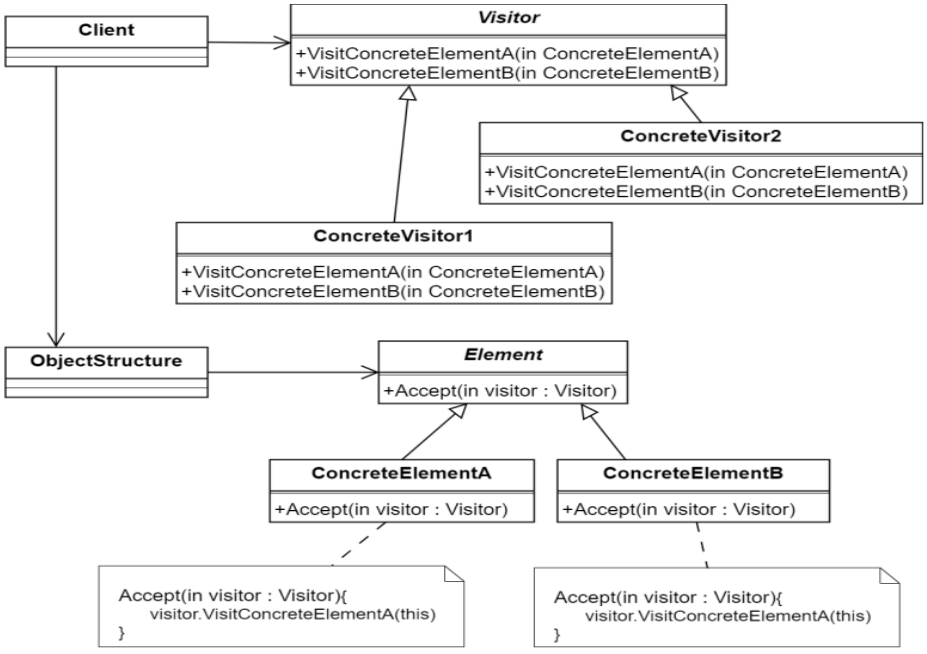


Рисунок 8.5. Структура патерна «Відвідувач»

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

```java
package com.example.jsontool.flyweight;


public class JsonStyle {  9 usages  new *
    private final String colorHex;  3 usages
    private final String fontStyle;  2 usages
    private final String description;  2 usages

    public JsonStyle(String colorHex, String fontStyle, String description) {  5 usages  new *
        this.colorHex = colorHex;
        this.fontStyle = fontStyle;
        this.description = description;
    }

    public String applyStyle(String text) {  1 usage  new *
        return String.format("<span style='color:%s; font-weight:%s' title='%s'>%s</span>",
                colorHex, fontStyle, description, text);
    }

    public String getColorHex() {  no usages  new *
        return colorHex;
    }
}
```

JsonStyle.java

```java
package com.example.jsontool.flyweight;

import java.util.HashMap;
import java.util.Map;


public class JsonStyleFactory {  1 usage  new *

    private static final Map<String, JsonStyle> styleCache = new HashMap<>();  2 usages


    public static JsonStyle getStyle(String type) {  1 usage  new *
        JsonStyle style = styleCache.get(type);

        if (style == null) {
            switch (type) {
                case "KEY":
                    style = new JsonStyle( colorHex: "#A52A2A",  fontStyle: "bold",  description: "JSON Key");
                    break;
                case "STRING":
                    style = new JsonStyle( colorHex: "#228B22",  fontStyle: "normal",  description: "String Value");
                    break;
                case "NUMBER":
                    style = new JsonStyle( colorHex: "#0000FF",  fontStyle: "normal",  description: "Numeric Value");
                    break;
                case "BOOLEAN":
                    style = new JsonStyle( colorHex: "#800080",  fontStyle: "bold",  description: "Boolean Value");
                    break;
                default:
                    style = new JsonStyle( colorHex: "#000000",  fontStyle: "normal",  description: "Default Text");
            }
            styleCache.put(type, style);
            System.out.println("[Flyweight Factory]: Створено новий стиль для типу: " + type);
        } else {
            // System.out.println("[Flyweight Factory]: Використано існуючий стиль для: " + type);
        }
        return style;
    }
}
```

JsonStyleFactory.java

```java
package com.example.jsontool.flyweight;

public class JsonToken {  6 usages  new *
    private String content;  2 usages
    private JsonStyle style;  2 usages

    public JsonToken(String content, String tokenType) {  4 usages  new *
        this.content = content;
        this.style = JsonStyleFactory.getStyle(tokenType);
    }

    public String render() {  4 usages  new *
        return style.applyStyle(content);
    }
}
```

JsonToken.java

3)Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.

Також добавимо класи, які допомжуть в реалізації та демонстрації

```java
package com.example.jsontool.service;

import com.example.jsontool.flyweight.JsonToken;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@Service  1 usage  new *
public class SyntaxHighlighterService {

    private final ObjectMapper objectMapper = new ObjectMapper();  2 usages

    public String highlightJson(String rawJson) {  2 usages  new *
        try {
            Object jsonObject = objectMapper.readValue(rawJson, Object.class);
            String prettyJson = objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(jsonObject);

            return tokenizeAndColor(prettyJson);

        } catch (Exception e) {

            return "<span style='color:red'>Invalid JSON: " + e.getMessage() + "</span>";
        }
    }

    private String tokenizeAndColor(String json) {  1 usage  new *
        StringBuilder htmlBuilder = new StringBuilder();

        String regex = "(\"[^\"]*\"\\s*:)|(\"[^\"]*\")|(\\b-?(?:0|[1-9]\\d*)(?:\\.\\d+)?\\b)|(\\btrue\\b|\\bfalse\\b|\

        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(json);

        int lastIndex = 0;

        while (matcher.find()) {

            String whitespace = json.substring(lastIndex, matcher.start());
            htmlBuilder.append(whitespace);

            String match = matcher.group();
            JsonToken token;
```

:e > SyntaxHighlighterService

SyntaxHighlighterService.java

SyntaxHighlighterService допоможе нам гарно виводити нам наші дані json

```java
    @GetMapping("/highlight-demo") new *
    @ResponseBody
    public String demoFlyweight() {
        String json = "{\"id\": 1, \"name\": \"Test\", \"isActive\": true, \"count\": 100, \"role\": \"ADMIN\"}";

        return "<html><body><h1>Flyweight Demo</h1>" +
                "<p>Оригінал: " + json + "</p>" +
                "<h3>Підсвічений JSON (HTML generated by Flyweights):</h3>" +
                "<div style='font-family: monospace; background: #f0f0f0; padding: 10px;'>" +
                highlighterService.highlightJson(json) +
                "</div></body></html>";
    }

    @GetMapping("/{id}") new *
    public String viewDocument(@PathVariable Long id, Model model) {

        JSONDocument document = documentService.findAll().stream() Stream<JSONDocument>
                .filter( JSONDocument d -> d.getId().equals(id))
                .findFirst() Optional<JSONDocument>
                .orElseThrow(() -> new IllegalArgumentException("Invalid document Id:" + id));

        String highlightedContent = highlighterService.highlightJson(document.getContent());

        model.addAttribute( attributeName: "document", document);
        model.addAttribute( attributeName: "highlightedContent", highlightedContent);

        return "view-document";
    }
}
```

DocumentController.java

```html
            <td>
                <a th:href="@{/documents/{id}(id=${doc.id})}" th:text="${doc.name}"
                   style="font-weight: 500; color: #007bff; text-decoration: none;">
                    Test.json
                </a>
            </td>
```

Document.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Перегляд документа</title>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;700&display=swap" rel="stylesheet">
    <style>
        body { font-family: 'Inter', sans-serif; margin: 0; background-color: #f8f9fa; padding: 2rem; }
        .container { max-width: 800px; margin: 0 auto; background: white; padding: 2rem; border-radius: 8px; box-shadow: 0 4px 12px rgba(0
        h1 { border-bottom: 1px solid #dee2e6; padding-bottom: 1rem; }

        /* Стиль для блоку з кодом */
        .code-viewer {
            background-color: #f0f0f0;
            padding: 15px;
            border-radius: 5px;
            font-family: 'Courier New', monospace;
            white-space: pre-wrap; /* Зберігає відступи та переноси */
            overflow-x: auto;
        }

        a.button {
            display: inline-block; background-color: #6c757d; color: white;
            padding: 10px 20px; text-decoration: none; border-radius: 5px; margin-top: 20px;
        }
        a.button:hover { background-color: #5a6268; }
    </style>
</head>
<body>
<div class="container">
    <h1 th:text="${document.name}">Назва документа</h1>

    <p><strong>ID:</strong> <span th:text="${document.id}"></span></p>
    <p><strong>Створено:</strong> <span th:text="${#temporals.format(document.createdAt, 'dd.MM.yyyy HH:mm')}"></span></p>

    <h3>Вміст (з підсвіткою Flyweight):</h3>
    <div class="code-viewer" th:utext="${highlightedContent}">
        Тут буде кольоровий JSON...
    </div>

    <a th:href="@{/documents}" class="button">Назад до списку</a>
</div>
</body>
</html>
```

View-document.html

Демонастрація роботи патерну

```
2025-11-22T01:29:08.294+02:00  INFO 8880 --- [json-tool] [nio-8080-exe
2025-11-22T01:29:08.294+02:00  INFO 8880 --- [json-tool] [nio-8080-exe
2025-11-22T01:29:08.295+02:00  INFO 8880 --- [json-tool] [nio-8080-exe
[Flyweight Factory]: Створено новий стиль для типу: KEY
[Flyweight Factory]: Створено новий стиль для типу: STRING
[Flyweight Factory]: Створено новий стиль для типу: BOOLEAN
```

У консолі ми бачимо повідомлення від JsonStyleFactory. Повідомлення "Створено новий стиль..." з'явиться лише один раз для кожного типу ("KEY", "NUMBER"), навіть якщо у нас є 100 ключів і 100 чисел. Це і є економія пам'яті завдяки реалізованому патерну Flyweight

Мої JSON документи

ID	НАЗВА ДОКУМЕНТА	ДАТА СТВОРЕННЯ	ДІЇ	
1	Burger	22.10.2025 13:08	[MD] [TXT]	Burger
2	test	22.10.2025 13:14	[MD] [TXT]	test
3	Добрий день	25.10.2025 01:48	[MD] [TXT]	Добрий день
4	Андрій 1	25.10.2025 02:09	[MD] [TXT]	Андрій 1
5	Тест	25.10.2025 02:22	[MD] [TXT]	Тест
8	Burger	08.11.2025 00:20	[MD] [TXT]	Burger

Справа можна бачити кнопку з назвою документу, при натисканні на неї можемо переглянути наш результат:


```
{
  "employees": [ {
    "firstName": "John",
    "lastName": "Doe"
  }, {
    "firstName": "Anna",
    "lastName": "Smith"
  }, {
    "firstName": "Peter",
    "lastName": "Jones"
  } ]
}
```


Назад до списку

Бачимо сторінку, де наш JSON розфарбований різними кольорами.

5)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 шаблон

Висновки

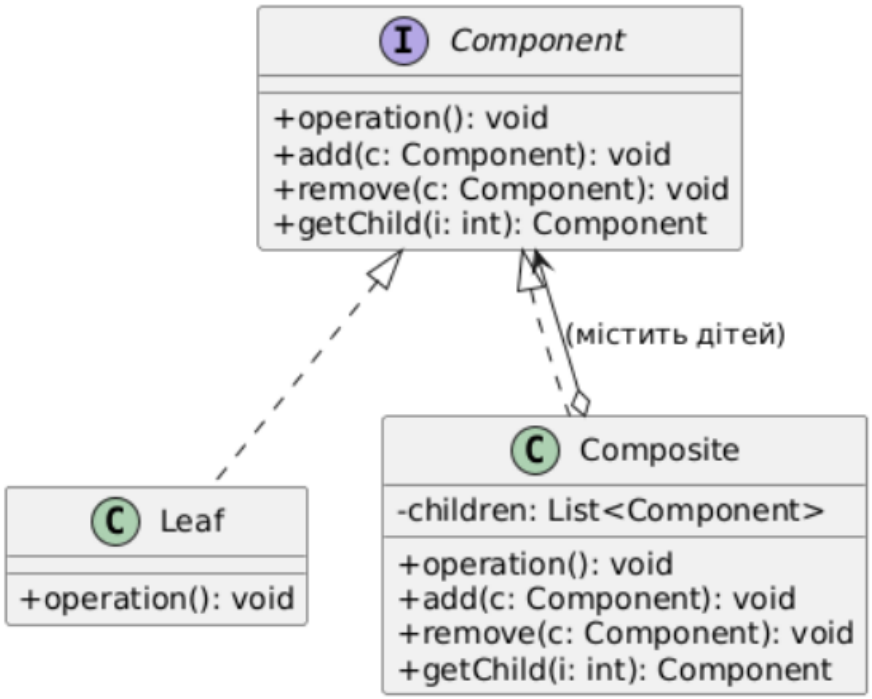
Під час виконання лабораторної роботи я вивчив структуру та призначення патерну проєктування **"Легковаговик" (Flyweight)**. Метою роботи було реалізувати цей патерн для оптимізації використання пам'яті при відображенні JSON-документів у моєму проєкті "JSON Tool". Для реалізації було створено три ключові класи, які допомогли в реалізації патерну. Було створено сервіс, який використовує ці класи для розбору JSON-тексту та генерації HTML-коду з підсвіткою синтаксису. Цей функціонал було інтегровано в веб-додаток: додано сторінку перегляду документа, де JSON відображається у зручному, кольоровому форматі.

Відповіді на контрольні питання

1. Яке призначення шаблону «Композит»?

Шаблон «Композит» (Composite) призначений для групування об'єктів у деревоподібні структури для представлення ієрархій "частина-ціле". Він дозволяє клієнтам працювати з окремими об'єктами та групами об'єктів (композиціями) однаково.

2. Нарисуйте структуру шаблону «Композит».



3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

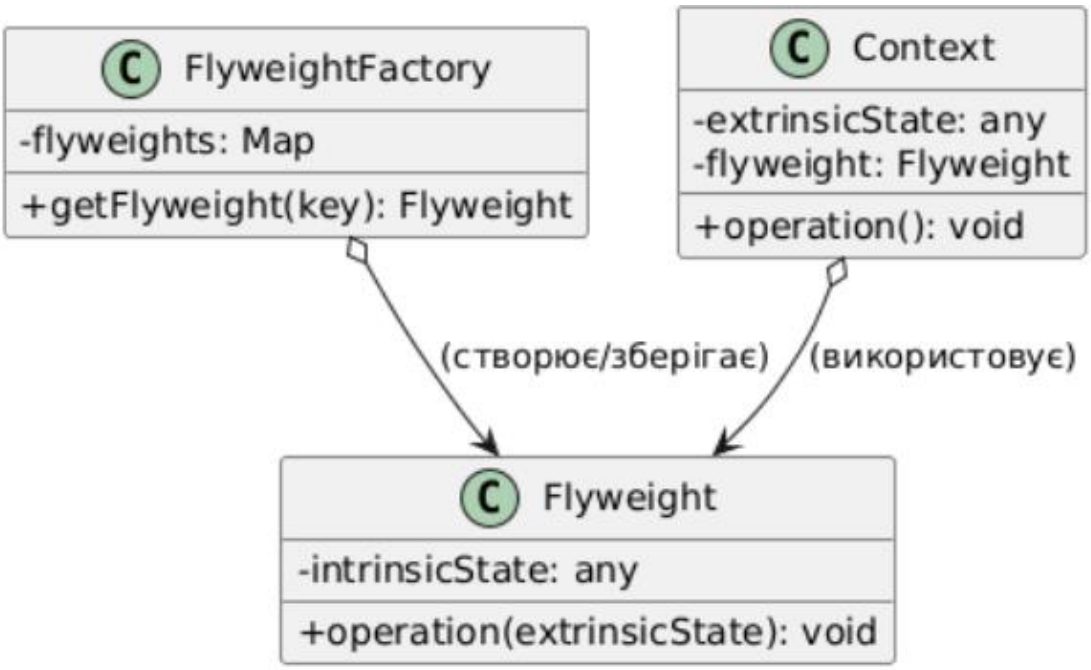
1. Component (Компонент): Абстракція для всіх елементів у структурі (і листків, і контейнерів). Оголошує інтерфейс для доступу до "дітей".
2. Leaf (Листок): Представляє кінцеві об'єкти, які не мають підлеглих. Реалізує методи Component.
3. Composite (Композит): Контейнер, який може містити інші компоненти (Leaf або Composite). Реалізує методи для додавання/видалення дітей.

Взаємодія: Клієнт працює через інтерфейс Component. Якщо він викликає метод у Leaf, виконується дія. Якщо у Composite — він передає виклик усім своїм дочірнім компонентам.

4. Яке призначення шаблону «Легковаговик»?

Шаблон «Легковаговик» (Flyweight) призначений для ефективної підтримки великої кількості дрібних об'єктів. Він дозволяє економити оперативну пам'ять шляхом спільного використання (sharing) спільного стану між багатьма об'єктами замість зберігання однакових даних у кожному екземплярі.

5. Нарисуйте структуру шаблону «Легковаговик».



6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

1. Flyweight (Легковаговик): Клас, що містить внутрішній стан (intrinsic state), який є спільним і незмінним.
2. FlyweightFactory (Фабрика): Створює та керує пулом легковаговиків. Забезпечує їх повторне використання.
3. Context (Контекст): Клас, що містить зовнішній стан (extrinsic state), унікальний для кожного об'єкта. Посилається на Flyweight.

4. Client (Клієнт): Обчислює або зберігає зовнішній стан і передає його методам легковаговика.

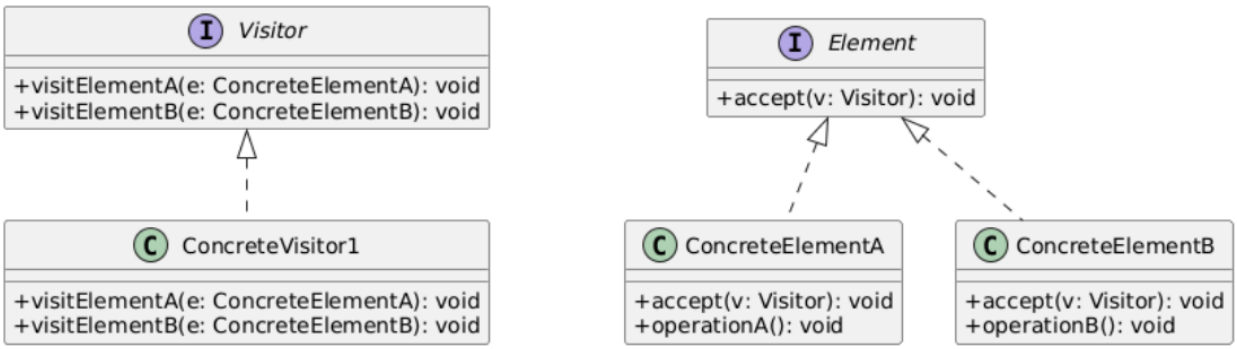
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Інтерпретатор» (Interpreter) використовується для визначення граматики простої мови та інтерпретації речень цією мовою. Він будує дерево об'єктів (синтаксичне дерево), де кожен вузол представляє правило граматики, і використовує це дерево для обчислення або виконання виразів. Часто використовується для SQL-подібних мов, математичних виразів тощо.

8. Яке призначення шаблону «Відвідувач»?

Шаблон «Відвідувач» (Visitor) дозволяє додавати нові операції до ієрархії класів без зміни самих класів цих об'єктів. Він відокремлює алгоритм від структури об'єктів, над якими цей алгоритм оперує.

9. Нарисуйте структуру шаблону «Відвідувач».



10. Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

1. Visitor (Відвідувач): Інтерфейс, що оголошує методи visit для кожного типу конкретного елемента (наприклад, visitElementA, visitElementB).
2. ConcreteVisitor: Реалізує конкретну операцію (алгоритм) для кожного класу елементів.
3. Element (Елемент): Інтерфейс, що оголошує метод accept(Visitor).
4. ConcreteElement: Класи елементів структури. Реалізують метод accept, який зазвичай просто викликає visitor.visit(this).

Взаємодія (Double Dispatch): Клієнт викликає element.accept(visitor). Елемент "знає" свій тип і викликає відповідний метод у відвідувача: visitor.visitElementA(this). Тепер Відвідувач знає і тип елемента, і яку операцію виконати.