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

Технології розроблення програмного забезпечення

Тема: «Патерни проектування»

Виконала:
Студентка групи ІА-34
Сизоненко А.О
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Мета: Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

Тема роботи: 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).

Теоретичні відомості

1. Composite (Компонувальник)

Використовується для представлення об'єктів у деревоподібній структурі, дозволяючи однаково працювати як з окремими елементами, так і з об'єктами з вкладеністю.

Наприклад, форма з дочірніми елементами (поля вводу, написи, малюнки) може застосовувати операції рекурсивно до всіх елементів, при цьому клієнтський код не потребує знання типів дочірніх об’єктів.

Патерн спрощує роботу з ієрархіями, додає гнучкості, але вимагає хорошої початкової організації інтерфейсів.

2. Flyweight (Легковаговик)

Зменшує кількість об’єктів у додатку шляхом їх спільного використання. Flyweight — це поділюваний об’єкт.

Концепція внутрішнього та зовнішнього стану:

- Внутрішній стан: дані, характерні для об’єкта (наприклад, код букви).
- Зовнішній стан: контекст використання (наприклад, позиція літери в рядку).
- Внутрішній стан зберігається всередині Flyweight, зовнішній — у контексті, де він застосовується.

Приклад: Текстовий редактор, де для кожної літери фізично існує один об’єкт, а багаторазові посилання на нього визначають розташування та форматування. Це значно економить пам’ять при великій кількості однакових елементів (літер, графічних примітивів тощо).

Переваги:

- Заощаджує оперативну пам’ять.
- Зменшує дублювання об’єктів.

Недоліки:

- Потребує додаткового часу на керування контекстом.
- Ускладнює код через введення додаткових класів.

3. Interpreter (Інтерпретатор)

Використовується для подання граматики мови та її інтерпретації через абстрактне синтаксичне дерево, де термінальні та нетермінальні вирази обчислюються рекурсивно.

Підходить для невеликих граматик і відносно простих контекстів, полегшує розширення граматики, але складний у супроводженні для великих мов.

4. Visitor (Відвідувач)

Дозволяє додавати нові операції над об’єктами без зміни їхньої структури. Наприклад, в онлайн-корзині різні відвідувачі обчислюють вартість товарів за різними правилами, при цьому самі класи товарів не змінюються.

Це спрощує додавання нових операцій, але ускладнює додавання нових типів елементів у ієрархію.

Хід роботи

1. Діаграма класів для реалізації шаблону шаблонний метод “templateMethod”.

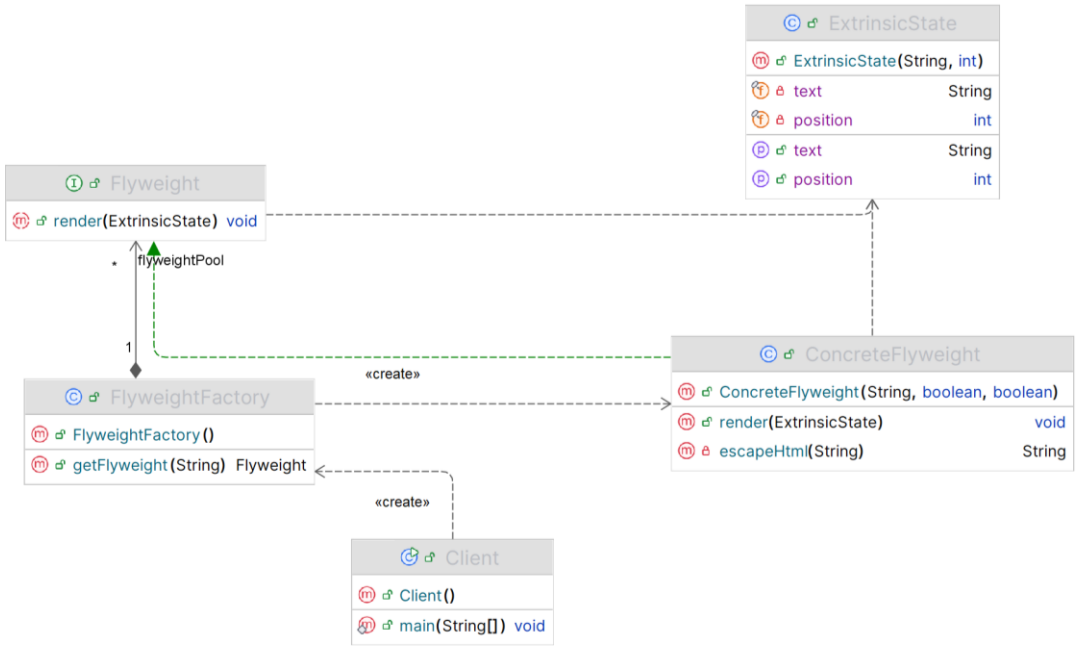


Рисунок 1 – Діаграма класів

Опис діаграми класів

1. Concrete Flyweight — це об’єкт, який зберігає спільні характеристики токена, такі як колір, жирність і курсив. Він використовується повторно для багатьох токенів одного типу, що дозволяє економити пам’ять.
2. Extrinsic State — це дані, що змінюються для кожного конкретного використання Flyweight, наприклад текст токена та його позиція у документі.
3. Flyweight Factory відповідає за створення або повернення вже існуючих Concrete Flyweight, щоб для одного типу токена не створювалися зайві об’єкти.
4. Клас Flyweight — це інтерфейс, який визначає загальні операції для об’єктів Flyweight.

Приклад реалізації шаблону:

Для текстового редактора кожен тип токена (ключове слово, число, рядок, коментар) має спільні властивості стилю (колір, жирність, курсив). Ці стилі повторно використовуються через Flyweight, а текст і позиція передаються як зовнішній стан.

1. Клієнт бере рядок коду, який потрібно підсвітити, і записує результат у HTML-файл. Він створює FlyweightFactory, який забезпечує повторне використання стилів для різних типів токенів (ключові слова, числа, рядки, коментарі).

2. Далі клієнт створює об’єкт JavaSyntaxHighlighter, який аналізує код, визначає типи токенів і для кожного токена запитує у фабрики відповідний Flyweight. Flyweight отримує ExtrinsicState із текстом токена і його позицією та відображає його з правильним стилем.
3. HTML-код формується із застосованими стилями для кожного токена, і весь результат записується у файл. Завдяки Flyweight, для всіх токенів одного типу не створюються нові об’єкти стилю, а повторно використовуються існуючі, що економить пам’ять.

HTML використано для наочного відображення тексту (кольорів, позицій, стилів)

Демонстрація роботи шаблону Flyweight

```
public class Client {
    public static void main(String[] args) {
        String code = "//привіт коментар\n if (x == 10) {\n    System.out.println(\"Hello\");\n}\nfor (int i = 0; i < 5; i++)
        String filename = "highlighted_output_client_only.html";

        PrintStream originalOut = System.out;

        try {
            PrintStream fileOut = new PrintStream(new FileOutputStream(filename));

            System.setOut(fileOut);

            System.out.println("<!DOCTYPE html>");
            System.out.println("<html lang=\"uk\">");
            System.out.println("<head><meta charset=\"UTF-8\"><title>Syntax Highlighted Code</title></head><body>");
            System.out.println("<pre style=\"font-family: monospace; background-color: white; padding: 10px;\">");

            FlyweightFactory factory = new FlyweightFactory();
            JavaSyntaxHighlighter highlighter = new JavaSyntaxHighlighter(factory);

            highlighter.highlight(code);

            System.out.println("</pre>");
            System.out.println("</body></html>");

            fileOut.close();

            System.setOut(originalOut);
            System.out.println("Код успішно записано у файл: " + filename);
```

Рисунок 2 – код класу Client

```
public class JavaSyntaxHighlighter {  3 usages
    private final FlyweightFactory factory;  2 usages

    public JavaSyntaxHighlighter(FlyweightFactory factory) {  1 usage
        this.factory = factory;
    }

    public void highlight(String code) {  1 usage
        String regex = "\"([^\"]*)\"|//[^\n]*|/\\*.*?\\*/|\\w+|\\d+|==|<=|>=|!=|[{}();=<>+-/*]|\\s+";
        Pattern pattern = Pattern.compile(regex, Pattern.DOTALL);
        Matcher matcher = pattern.matcher(code);

        int position = 0;
        while (matcher.find()) {
            String token = matcher.group();
            String type = "DEFAULT";

            if (token.matches( regex: "if|for|while|switch|public|class|void|static")) type = "KEYWORD";
            else if (token.matches( regex: "\\d+")) type = "NUMBER";
            else if (token.startsWith("\"") && token.endsWith("\"")) type = "STRING";
            else if (token.startsWith("//")|| (token.startsWith("/*") && token.endsWith("*/"))) type = "COMMENT";


            Flyweight flyweight = factory.getFlyweight(type);
            ExtrinsicState state = new ExtrinsicState(token, position);
            flyweight.render(state);

            position += token.length();
        }
    }
```

Рисунок 3 – код класу JavaSyntaxHighlighter (аналіз тексту)

```
public class FlyweightFactory {  5 usages
    private final Map<String, Flyweight> flyweightPool = new HashMap<>();  7 usages

    public Flyweight getFlyweight(String key) {  1 usage
        if (!flyweightPool.containsKey(key)) {
            switch (key) {
                case "KEYWORD":
                    flyweightPool.put(key, new ConcreteFlyweight( color: "blue",  bold: true,  italic: false));
                    break;
                case "NUMBER":
                    flyweightPool.put(key, new ConcreteFlyweight( color: "orange",  bold: false,  italic: false));
                    break;
                case "STRING":
                    flyweightPool.put(key, new ConcreteFlyweight( color: "green",  bold: false,  italic: true));
                    break;
                case "COMMENT":
                    flyweightPool.put(key, new ConcreteFlyweight( color: "gray",  bold: false,  italic: true));
                    break;
                default:
                    flyweightPool.put(key, new ConcreteFlyweight( color: "black",  bold: false,  italic: false));
                    break;
            }
        }
        return flyweightPool.get(key);
    }
}
```

Рисунок 4 – код класу FlyweightFactory

Результат виконання програми:

```
//привіт коментар
 if (x == 10) {
    System.out.println("Hello");
}
for (int i = 0; i < 5; i++) {
}
```

Рисунок 5 – результат обробки тексту

Тут важливий шаблон Flyweight, тому що він дозволяє не створювати окремий об’єкт стилю для кожного токена. У коді можуть бути сотні або тисячі токенів — ключові слова, числа, рядки, коментарі.

Якби для кожного токена створювався свій об’єкт стилю, це швидко забрало б багато пам’яті. Flyweight зберігає спільні властивості стилю (колір, жирність, курсив) один раз і повторно використовує їх для всіх токенів одного типу.

Зовнішній стан (текст токена і його позиція) передається окремо через ExtrinsicState, тому кожен токен виглядає правильно, але кількість об’єктів у пам’яті мінімальна, що робить підсвітку синтаксису ефективною навіть для великих файлів.

Переваги використання шаблону Flyweight для моєї теми проекту:

Використання Flyweight у текстовому редакторі дозволяє економити пам’ять, бо один об’єкт стилю (колір, жирність, курсив) використовується для багатьох токенів одного типу, замість створення окремого об’єкта для кожного слова або символу. Це особливо корисно для великих файлів із тисячами токенів.

Flyweight також забезпечує єдине джерело стилів, що полегшує їх зміну та підтримку. Зовнішній стан передається окремо, що дозволяє повторно використовувати один і той же об’єкт для різних позицій та текстів.

Недоліки використання шаблону Flyweight у текстовому редакторі:

Реалізація Flyweight ускладнює код, оскільки потрібно чітко розділяти внутрішній і зовнішній стан, а також використовувати фабрику для створення і повторного використання об’єктів.

Якщо зовнішній стан складний або часто змінюється, це може призвести до додаткових витрат на створення і передачу ExtrinsicState, що іноді зменшує ефект від економії пам’яті.

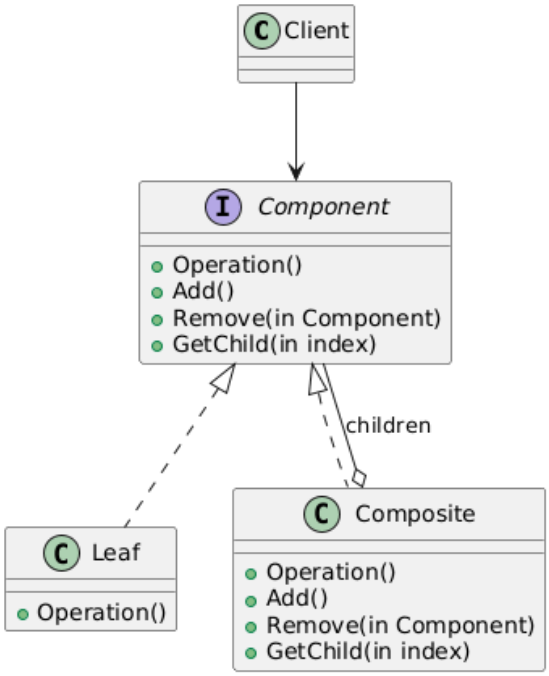
Висновок: У роботі реалізовано патерн Flyweight для ефективної підсвітки синтаксису у текстовому редакторі. Клас Flyweight визначає інтерфейс для відображення токенів із зовнішнім станом, ConcreteFlyweight зберігає спільні властивості стилю, такі як колір, жирність і курсив, а ExtrinsicState містить змінні дані токена, зокрема текст і позицію. FlyweightFactory забезпечує повторне використання об’єктів ConcreteFlyweight, завдяки чому один і той же стиль застосовується до всіх токенів одного типу без створення окремого об’єкта для кожного. Це дозволяє значно економити пам’ять при обробці великих файлів, спрощує підтримку стилів і забезпечує гнучкість у додаванні нових типів токенів. Було створено діаграму класів, яка наочно демонструє взаємодію компонентів системи: фабрика керує об’єктами Flyweight, ConcreteFlyweight зберігає внутрішній стан, ExtrinsicState передає конкретні дані для відображення, а клієнт комбінує ці елементи для підсвітки коду.

Відповіді на контрольні питання

1. Яке призначення шаблону «Композит»?

Шаблон дозволяє будувати деревоподібні структури для подання ієрархій «частина–ціле» та забезпечує однакову роботу з окремими елементами й складними об’єктами, що містять вкладені елементи. Дає змогу рекурсивно застосовувати операції до всіх елементів дерева.

2. Нарисуйте структуру шаблону «Композит».



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

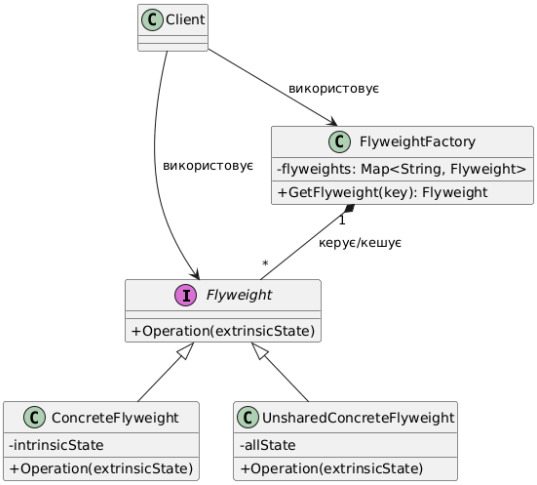
Component — базовий інтерфейс із методами Operation(), Add(), Remove(), GetChild().
Leaf — кінцевий елемент, реалізує лише Operation().
Composite — вузол, містить список children і рекурсивно виконує Operation() для всіх дітей; реалізує Add(), Remove(), GetChild().

Взаємодія: клієнт працює через Component; Composite містить інші Component (Leaf або Composite); виклики Operation() йдуть рекурсивно по дереву.

4. Яке призначення шаблону «Легковаговик»?

Шаблон призначений для зменшення кількості об’єктів шляхом спільного використання внутрішнього стану об’єкта. Внутрішній стан зберігається у Flyweight-об’єкті, а зовнішній передається клієнтом при використанні.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Flyweight (Інтерфейс Пристосуванця): Визначає спільний інтерфейс для всіх об'єктів.

ConcreteFlyweight (Конкретний Пристосуванець): Зберігає внутрішній (спільний, незмінний) стан. Ці об'єкти кешуються і використовуються повторно.

FlyweightFactory (Фабрика): Керує створенням і кешуванням об'єктів ConcreteFlyweight, повертає існуючі екземпляри за запитом.

Client (Клієнт): Зберігає зовнішній (унікальний, змінний) стан і передає його методам Flyweight як параметр.

Взаємодія: Клієнт просить фабрику надати об'єкт (який береться з кешу або створюється), а потім викликає його метод, передаючи унікальні дані як аргумент.

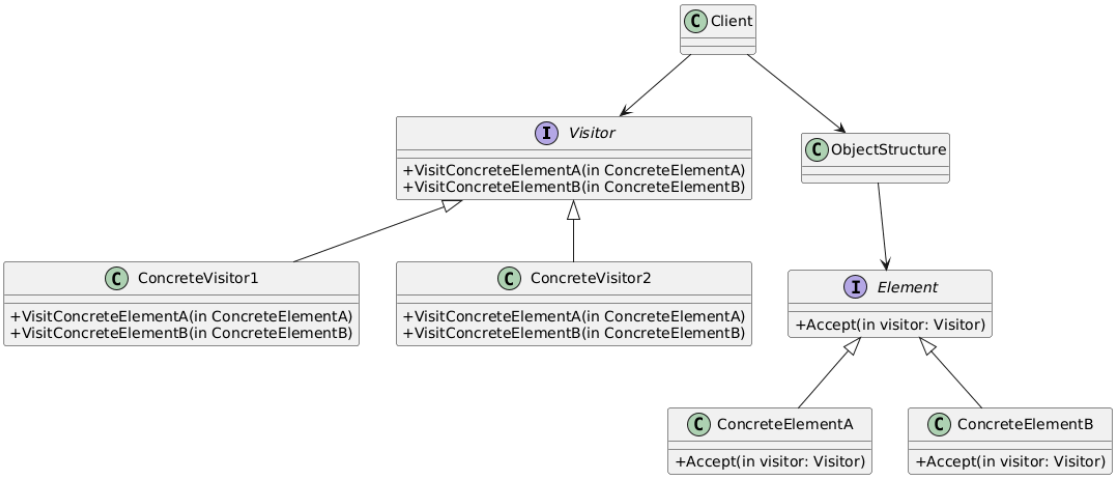
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон описує граматику мови та спосіб її інтерпретації через абстрактне синтаксичне дерево. Кожен вузол (вираз) інтерпретується рекурсивно, обчислюючи результат на основі контексту.

8. Яке призначення шаблону «Відвідувач»?

Шаблон дозволяє додавати нові операції над елементами складних структур, не змінюючи класи елементів. Дані й логіка розносяться в різні ієрархії: елементи містять дані, Visitor — логіку.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Visitor (Відвідувач): Інтерфейс, який оголошує методи Visit() для кожного конкретного типу елемента. Визначає операції, які можна виконувати.

ConcreteVisitor (Конкретний Відвідувач): Реалізує інтерфейс Visitor і містить конкретну логіку операції для кожного типу елемента.

Element (Елемент): Інтерфейс, який оголошує єдиний метод Accept(visitor: Visitor).

ConcreteElement (Конкретний Елемент): Реалізує Element, викликаючи відповідний метод Visit() на відвідувачі (visitor.VisitConcreteElementA(this)).

ObjectStructure (Структура Об'єктів): Контейнер, який зберігає колекцію елементів і може автоматично застосовувати відвідувача до всіх елементів одразу.

Взаємодія відбувається так:

Клієнт ініціює процес, передаючи конкретного Visitor елементу через метод Accept().

ConcreteElement у своєму методі Accept() викликає назад відповідний метод Visit() на об'єкті відвідувача, передаючи себе як посилання.

ConcreteVisitor отримує посилання на конкретний елемент і виконує над ним свою специфічну операцію.