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 №8

із дисципліни «Технології розроблення програмного забезпечення»
Тема: «Шаблони «Composite», «Flyweight», «Interpreter», «Visitor»»

Виконав:
Студент групи ІА-24
Боднар А. Д.

Перевірив:
Мягкий М.Ю.

Київ-2024

Зміст

Завдання:
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Теоретичні відомості

Шаблон Composite

Суть шаблону: Composite дозволяє створювати дерево об'єктів, де окремі об'єкти та їхні групи обробляються однаково. Наприклад, текстові документи можуть мати абзаци, абзаци — рядки, рядки — слова.

- **Переваги**:
    1. Спрощує роботу з ієрархіями.
    2. Полегшує додавання нових компонентів.
    3. Дозволяє працювати з окремими об'єктами та їх групами однаково.
- **Недоліки**:
    1. Складно контролювати доступ до компонентів.
    2. Може бути перевантажений, якщо компоненти мають надмірну логіку.

Шаблон Flyweight

Суть шаблону: Flyweight оптимізує використання пам'яті шляхом зберігання загальних об'єктів у вигляді одного екземпляра (наприклад, стилі тексту: жирний, курсив).

- **Переваги**:
    1. Економія пам’яті за рахунок повторного використання об’єктів.
    2. Централізоване управління спільними даними.
- **Недоліки**:
    1. Ускладнює код, особливо при управлінні станом.
    2. Підходить не для всіх типів додатків.

- **Приклад використання**: У текстових редакторах для зберігання інформації про шрифт, колір або стиль окремих символів.

Шаблон Interpreter

Суть шаблону: Interpreter дозволяє створити мову для виконання певних команд і обробляти ці команди у вигляді об'єктів. Наприклад, вирази типу 2 + 3 * 4 можуть бути представлені деревом інтерпретаторів.

- **Переваги**:
    1. Легко розширювати новими виразами.
    2. Можливість реалізації доменних мов.
- **Недоліки**:
    1. Непідходящий для складних мов.
    2. Важкий для розуміння в масштабних системах.
- **Приклад використання**: Сценарії, де треба парсити текст і виконувати команди (наприклад, команди у текстовому редакторі).

Шаблон Visitor

Суть шаблону: Visitor дозволяє додавати нові операції до об'єктів, не змінюючи самі класи об'єктів. Це корисно для обробки структур, наприклад, елементів документа.

- **Переваги**:
    1. Полегшує додавання операцій.
    2. Сприяє розділенню логіки між класами.
- **Недоліки**:
    1. Ускладнює додавання нових класів об'єктів.
    2. Утруднює розуміння при великій кількості відвідувачів.
- **Приклад використання**: У текстових редакторах для додавання аналізаторів синтаксису.

Хід роботи

Варіант - 3

Текстовий редактор

Використання шаблону “Flyweight”

```
public class StyleFactory {  2 usages
    private Map<String, Style> styleCache = new HashMap<>();  2 usages

    public Style getStyle(String colorCode) {  1 usage
        System.out.println("[DEBUG] Необхідно стиль для кольору: " + colorCode);
        Style style = styleCache.get(colorCode);
        if (style == null) {
            System.out.println("[DEBUG] Виклик створення нового стилю для кольору: " + colorCode);
            style = createNewStyle(colorCode);
            styleCache.put(colorCode, style);
            System.out.println("[LOG] Створено новий стиль для кольору: " + colorCode);
        }
        return style;
    }

    public Style createNewStyle(String colorCode) {  1 usage
        Style style = new StyleContext().addStyle(colorCode,  parent: null);
        Color highlightColor = parseColor(colorCode);  // Перетворення кольору
        StyleConstants.setForeground(style, highlightColor);
        return style;
    }
```

Рис. 1 - Перша частина класу StyleFactory

```
    public static Color parseColor(String color) {  1 usage
        try {
            if (color.startsWith("#")) {
                return Color.decode(color);
            } else {
                return (Color) Color.class.getField(color.toUpperCase()).get(null); // Наприклад, red, blue
            }
        } catch (Exception e) {
            System.err.println("Invalid color format: " + color);
            return Color.BLACK; // За замовчуванням чорний
        }
    }
}
```

Рис. 2 - Друга частина класу StyleFactory

```java
for (SyntaxHighlighterWord word : words) {
    String searchWord = word.getWord();
    int idColor = word.getIdColor();
    if (searchWord != null && idColor != 0) {
        String colorCode = colorsRepository.getColorCodeById(idColor);
        if (colorCode != null) {
            Pattern pattern = Pattern.compile( regex: "\\b" + Pattern.quote(searchWord) + "\\b");
            Matcher matcher = pattern.matcher(text);

            while (matcher.find()) {
                int start = matcher.start();
                int length = searchWord.length();

                for (int i = start; i < start + length; i++) {
                    AttributeSet attributes = doc.getCharacterElement(i).getAttributes();
                    MutableAttributeSet newAttributes = new SimpleAttributeSet(attributes);

                    // Отримуємо стиль через фабрику
                    Style style = styleFactory.getStyle(colorCode);
                    Color highlightColor = StyleConstants.getForeground(style);
                    StyleConstants.setForeground(newAttributes, highlightColor);

                    doc.setCharacterAttributes(i,  length: 1, newAttributes,  replace: true);
                    System.out.println("[LOG] Застосовано створений стиль для тексту: " + searchWord + " на позиції " + i);
                }
            }
        }
    }
```

Рис. 3 - Частина класу applyHighlighting

Опис реалізації шаблону Flyweight

1. **Ціль шаблону Flyweight**: Flyweight дозволяє оптимізувати використання пам'яті, уникаючи створення багатьох однакових об'єктів. У цьому випадку замість створення стилю для кожного слова/кольору щоразу, стилі кешуються і повторно використовуються.
2. **Елементи Flyweight у коді**:
    - **styleCache**: Мапа для зберігання створених стилів (Style) за їх кольором (colorCode).
    - **getStyle**: Перевіряє, чи стиль із потрібним кольором уже існує. Якщо так, повертає його; якщо ні, створює новий і зберігає в кеші.
    - **createNewStyle**: Створює новий стиль із заданим кольором.
3. **Використання Flyweight**:
    - У **SyntaxHighlighterProcessor** метод styleFactory.getStyle(colorCode) відповідає за отримання або створення стилю. Це запобігає повторному створенню стилів для тих самих кольорів.
    - Завдяки кешуванню стиль кожного кольору створюється лише один раз, незалежно від кількості слів або їх позицій.

4. **Переваги**:
    - **Ефективність**: Зменшує витрати на створення однакових стилів.
    - **Скорочення використання пам'яті**: Один стиль використовується для багатьох слів із однаковим кольором.

Ця реалізація відповідає класичному шаблону Flyweight, де:

- **Контекст** — це слова, позиції в тексті та кольори.
- **Спільний стан** — це стилі (Style), які зберігаються в styleCache.
- **Неспільний стан** — це специфічні атрибути кожного символу в тексті.

```
[DEBUG] Необхідно стиль для кольору: #a720f0
[DEBUG] Виклик створення нового стилю для кольору: #a720f0
[LOG] Створено новий стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 0
[DEBUG] Необхідно стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 1
[DEBUG] Необхідно стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 2
[DEBUG] Необхідно стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 0
[DEBUG] Необхідно стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 1
[DEBUG] Необхідно стиль для кольору: #a720f0
[LOG] Застосовано створений стиль для тексту: out на позиції 2
```

Рис. 4 - Приклад використання шаблону

(1) Після повного введення слова для підсвідки:
До першої червної пунктирної лінії, отримується інформація про стиль, застосунок бачить що стиль для такого кольору нема і створнює, після чого застосовує до першої букви
Від першої червоної пунктирної лінії до першої синьої стиль зразу знаходиться і застосовується до другої букви

Так само можна сказати і про 3 букву за кодру відповідає частина від першої синьої пунктирної лінії до першої червоної лінія

(2) Після зміни в текстовому редакторі, котра не порушила слово, виконуються 3 дії для застосування вже створеного стил

Висновки

У даній лабораторній роботі було реалізовано застосування шаблону **Flyweight** для оптимізації підсвітки синтаксису у текстовому редакторі. Шаблон **Flyweight** дозволяє знизити витрати пам'яті, забезпечуючи спільне використання однакових об'єктів для елементів, що мають однакові властивості. Це дуже корисно у випадках, коли є велика кількість схожих об'єктів, але кожен з них має лише незначні відмінності.

У нашому випадку, для підсвітки тексту з різними кольорами, було створено механізм кешування стилів, де стилі, що відповідають однаковим кольорам, використовуються повторно замість створення нового стилю для кожного слова в тексті. Ось кілька основних моментів, які були досягнуті завдяки використанню шаблону **Flyweight**:

1. **Зниження витрат пам'яті**:
    - Завдяки кешуванню стилів, кожен колір, що використовується для підсвітки, має лише один стиль, який застосовується до всіх слів того ж кольору. Це дозволяє уникнути дублювання стилів у пам'яті та суттєво знижує її використання.
2. **Оптимізація роботи програми**:
    - Коли користувач виділяє текст, система перевіряє, чи існує стиль для конкретного кольору в кеші. Якщо стиль вже є, він використовується повторно, що дозволяє значно прискорити підсвітку тексту, зменшуючи кількість операцій із створенням нових об'єктів.
3. **Реалізація Flyweight через кешування**:
    - Важливою частиною реалізації є **StyleFactory**, який відповідає за створення нових стилів для кольорів та їх кешування. Якщо стиль для певного кольору вже існує, він використовується повторно, що забезпечує ефективне використання пам'яті.
    - **Flyweight** був застосований для стилів, що мають однакові параметри (колір), і зберігання таких стилів у кеші дозволяє

зекономити ресурси при великій кількості текстових елементів, що використовують однакові кольори.

4. **Застосування Flyweight у текстових редакторах**:
    - Цей шаблон є особливо корисним у текстових редакторах, де однакові стилі можуть бути застосовані до різних частин тексту (наприклад, для ключових слів у коді). Використання Flyweight дозволяє значно зменшити кількість необхідних об'єктів стилю, що підвищує ефективність редактора.
5. **Інші можливості для застосування Flyweight**:
    - Окрім стилів, шаблон **Flyweight** може бути застосований і до інших частин текстового редактора, таких як шрифти або навіть інші властивості тексту, що часто повторюються. Цей підхід дозволяє уникнути дублювання об'єктів у пам'яті та підвищити ефективність програми.

Загальний підсумок: Використання шаблону **Flyweight** для підсвітки синтаксису в текстовому редакторі значно підвищує ефективність програми за рахунок оптимізації пам'яті та зменшення кількості створюваних об'єктів для однакових стилів. Такий підхід є важливим для розробки програм, де необхідно обробляти великі об'єми схожих даних (наприклад, для підсвітки багатьох слів з однаковими властивостями).