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”

```
Color highlightColor = StyleFactory.parseColor(colorCode);
Style style = doc.getStyle(searchWord);
if (style == null) {
    style = doc.addStyle(searchWord,  parent: null);
    StyleConstants.setForeground(style, highlightColor);
}
```

Рис. 1 - Частина методу applyHighlighting класу SyntaxHighlighterProcessorTemplate

Color highlightColor = StyleFactory.parseColor(colorCode);

- Метод **parseColor** в **StyleFactory** перетворює рядкове значення кольору, отримане з бази даних (у вигляді коду кольору), в об'єкт **Color**.
- Це частина логіки Flyweight, оскільки ми обробляємо колір у єдиному місці та повторно використовуємо його для всіх слів, які потребують того ж кольору. Це дозволяє уникнути дублювання об'єктів **Color**.

Style style = doc.getStyle(searchWord);

- Використовується для пошуку вже існуючого стилю, асоційованого з конкретним словом **searchWord**.
- Якщо стиль існує, то він повторно використовується. Це є ще одним проявом шаблону **Flyweight**, оскільки ми не створюємо новий стиль для кожного випадку, якщо стиль для цього слова вже існує.

if (style == null) { style = doc.addStyle(searchWord, null); }

- Якщо стиль ще не існує, то створюється новий стиль за допомогою **addStyle** і присвоюється йому ім'я, яке співпадає з **searchWord**.

- Це створення нового стилю також враховує принцип **Flyweight**, оскільки ми додаємо стиль в **StyledDocument** лише тоді, коли він відсутній, тим самим економлячи ресурси.

StyleConstants.setForeground(style, highlightColor);

- Тут встановлюється колір для стилю, використовуючи колір, отриманий на першому кроці (збережений у **highlightColor**).
- Це реалізація принципу **Flyweight**, оскільки колір є загальним ресурсом, який можна використовувати багато разів, замість того, щоб створювати окремі об'єкти стилю для кожного слова з власним кольором.

```
Color highlightColor = StyleFactory.parseColor(colorCode);
if (highlightColor != null) {
    Style style = styleFactory.getStyle(colorCode);
    if (style == null) {
        style = styleFactory.createNewStyle(colorCode);
    }
```

Рис. 2 - Частина методу applyStyleToWord класу SyntaxHighlighterProcessorTemplate

Ця частина методу applyStyleToWord відповідає принципам Flyweight так, як вона використовує централізовану фабрику стилів StyleFactory для забезпечення спільного використання об'єктів стилів, замість того, щоб кожного разу створювати новий стиль для кожного окремого слова. Це дозволяє зменшити витрати пам'яті, коли однакові стилі (у цьому випадку, кольори) повторно використовуються для різних елементів.

```java
public class StyleFactory {  8 usages
    private Map<String, Style> styleCache = new HashMap<>();  2 usages
    public Style getStyle(String colorCode) {  2 usages
        Style style = styleCache.get(colorCode);
        if (style == null) {
            // Створюємо новий стиль
            style = createNewStyle(colorCode);
            styleCache.put(colorCode, style);
        }
        return style;
    }


    public Style createNewStyle(String colorCode) {  2 usages
        // Тут ми створюємо новий стиль
        Style style = new StyleContext().addStyle(colorCode,  parent: null);
        Color highlightColor = parseColor(colorCode);  // Використовуємо метод
        StyleConstants.setForeground(style, highlightColor);
        return style;
    }
```

Рис. 3 - Частина класу StyleFactory

Ця частина коду класу **StyleFactory** відповідає принципам шаблону **Flyweight** через використання **кешування стилів**, щоб уникнути створення нових об'єктів стилю для однакових параметрів (у цьому випадку — кольору).

Для зберігання вже створених стилів використовується кеш у вигляді **styleCache**. Це дозволяє зберігати існуючі стилі, щоб повторно використовувати їх, замість того, щоб створювати нові об'єкти стилів для кожного елемента тексту з тим самим кольором.

getStyle(String colorCode): цей метод намагається знайти стиль для заданого кольору у кеші. Якщо стиль вже існує, він повертається і використовується. Якщо ж стиль для цього кольору відсутній, створюється новий стиль і додається в кеш для подальшого використання.

Це гарантує, що для кожного унікального кольору створюється лише один об'єкт стилю, який може бути використаний повторно в різних частинах тексту.

createNewStyle(String colorCode): цей метод створює новий стиль для заданого кольору, якщо такого стилю ще не існує. Стиль створюється за допомогою **StyleContext.addStyle()** і його колір встановлюється за допомогою **StyleConstants.setForeground()**.

Якщо для заданого кольору не існує стилю, він буде створений один раз і збережений в кеші, після чого використовуватиметься повторно для інших елементів тексту з тим самим кольором.

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).