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 №7
із дисципліни «Технології Розробки Програмного Забезпечення»
Тема: ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»
Варіант-28

Виконав:
Студент групи ІА-24
Коханчук Михайло Миколайович

Перевірив:
Мягкий Михайло Юрійович

Київ-2024

ЗМІСТ

МЕТА

Метою роботи є розробити частину функціоналу застосунку для роботи з JSON, реалізувавши класи та їхню взаємодію для забезпечення динамічного форматування даних. У процесі реалізації застосувати шаблон проектування Template method для досягнення гнучкості та масштабованості системи

Теоритичні відомості:

Будь-який патерн проектування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проектування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проектування обов'язково має загальновживане найменування. Правильно сформульований патерн проектування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої предметної області. Це є необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проектування.

Відповідне використання патернів проектування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проектування, фактично є структурованим виокремленням тих елементів і зв'язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проектування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови

Застосування патернів проектування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проектування, по суті, являє собою єдиний словник проектування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проектах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

Застосування шаблонів проектування не гарантує, що розроблена архітектура буде кристально чистою і зручною з точки зору програмування. Однак в потрібних місцях застосування шаблонів дозволить досягти наступних вигод:

- Зменшення трудовитрат і часу на побудову архітектури;
- Надання проектованій системі необхідних якостей (гнучкість, адаптованість, ін.);
- Зменшити накладні витрати на подальшу підтримку системи;
- Та інші.

Варто також зазначити, що знання шаблонів проектування допомагає не тільки архітекторам програмних систем, але і розробникам. Коли кожна людина в команді знає значення і властивості шаблонів, архітекторові простіше донести загальну ідею архітектури системи, а розробникам - простіше зрозуміти.

Оскільки, урешті-решт, кожен бізнес зводиться до грошей, шаблони проектування також є економічно виправданим вибором між побудовою власного «колеса», та реалізацією закріплених і гарантованих спільнотою розробників практик і підходів.

Це звичайно ж не означає, що їх необхідно використовувати в кожному проекті на кожну вимогу. Підходи не є догмою, їх потрібно використовувати з головою.

В межах даної та наступних робіт я посилатимусь та цитуватиму книгу Олександра Швецова «Занурення в патерни проектування».

Класифікація патернів

Патерни проектування відрізняються складністю, рівнем деталізації та масштабом застосування до всієї системи, що проектується. Мені подобається аналогія з будівництвом доріг: можна зробити перехрестя безпечнішим, встановивши кілька світлофорів або побудувавши цілу багаторівневу розв'язку з підземними переходами для пішоходів.

Найпростіші та низькорівневі патерни часто називають ідіомами. Зазвичай вони стосуються лише однієї мови програмування.

Найбільш універсальні та високорівневі патерни - це архітектурні патерни. Розробники можуть реалізувати ці патерни практично на будь-якій мові. На відміну від інших патернів, їх можна використовувати для проектування архітектури цілого додатку.

Крім того, всі патерни можна класифікувати за їхнім призначенням. Розглянемо три основні групи патернів:

- Породжувальні патерни(creational patterns) надають механізми створення об'єктів, які підвищують гнучкість і повторне використання існуючого коду.
- Структурні патерни(structural patterns) пояснюють, як збирати об'єкти та класи у більші структури, зберігаючи ці структури гнучкими та ефективними.
- Поведінкові патерни(behavioral) дбають про ефективну комунікацію та розподіл обов'язків між об'єктами.

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

Хід роботи:

У процесі форматування JSON часто потрібно виконувати однакові операції, як-от парсинг JSON, додавання відступів, видалення зайвих символів чи рекурсивну обробку вузлів. Якщо ці операції реалізовувати окремо в кожному форматувальнику, це призводить до дублювання коду, ускладнює його підтримку та збільшує ризик помилок. Водночас необхідно надати можливість легко змінювати чи розширювати специфічні алгоритми форматування.

Рішення

Для вирішення цієї проблеми використано патерн Шаблонний метод, який дозволяє визначити загальний алгоритм роботи в абстрактному класі та реалізовувати специфічну логіку в його підкласах.

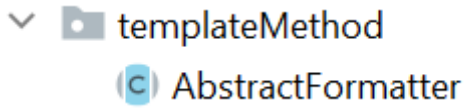

Рис. 1. Ієрархія класів. Реалізація патерну Template Method.

```java
public abstract class AbstractFormatter {
    1 usage
    protected final ObjectMapper objectMapper = new ObjectMapper();

    2 usages
    protected JsonNode parseJson(RawJsonDto rawJsonDto) throws Exception {
        return objectMapper.readTree(rawJsonDto.getRawData());
    }

    4 usages
    protected void appendIndent(StringBuilder builder, int level) { builder.append("  ".repeat(level)); }

    2 usages
    protected void removeTrailingComma(StringBuilder builder) {
        int length = builder.length();
        if (length > 2 && builder.substring( start: length - 2).equals(",\n")) {
            builder.delete(length - 2, length);
        }
    }
}
```

Рис. 2. Абстрактний клас AbstractFormatter для реалізації патерну Template Method.

У класі AbstractFormatter зібрано спільні методи, як-от parseJson, appendIndent і removeTrailingComma, які використовуються у конкретних класах форматування FlatFormatterStrategy і PrettyFormatterStrategy. Цей підхід дозволяє зосередити загальну логіку в одному місці, що спрощує її підтримку та зменшує дублювання. Це зменшує ризик помилок, підвищує масштабованість та забезпечує кращу структуризацію коду.

```
public class PrettyFormatterStrategy extends AbstractFormatter implements FormatterStrategy {

    @Override
    public String format(RawJsonDto rawJsonDto) {
        try {
            JsonNode rootNode = parseJson(rawJsonDto);
            StringBuilder builder = new StringBuilder();
            formatNodeRecursive(rootNode, level: 0, builder);
            return builder.toString().trim();
        } catch (Exception e) {
            throw new InvalidJsonException("Error during formatting", e);
        }
    }

    3 usages
    private void formatNodeRecursive(JsonNode node, int level, StringBuilder builder) {
        if (node.isObject()) {
            builder.append("{\n");
            node.fields().forEachRemaining(entry -> {
                appendIndent(builder, level: level + 1);
                builder.append("\"").append(entry.getKey()).append("\": ");
                formatNodeRecursive(entry.getValue(), level: level + 1, builder);
                builder.append(",\n");
            });
            removeTrailingComma(builder);
            builder.append("\n");
            appendIndent(builder, level);
```

Рис. 2.1.1 Використання методів класу AbstractFormatter у конкретних класах форматування

```
public class FlatFormatterStrategy extends AbstractFormatter implements FormatterStrategy {

    @Override
    public String format(RawJsonDto rawJsonDto) {
        try {
            JsonNode rootNode = parseJson(rawJsonDto);
            StringBuilder builder = new StringBuilder();
            formatNodeRecursive(rootNode, currentPath: "", builder);
            return builder.toString().trim();
        } catch (Exception e) {
            throw new InvalidJsonException("Error during formatting", e);
        }
    }
```

Рис. 2.1.2 Використання методів класу AbstractFormatter у конкретних класах форматування

ВИСНОВОК

Реалізація патерну Template method у межах цього застосунку дозволила зібрати спільні методи які використовуються у конкретних класах форматування. Цей підхід дозволяє зосередити загальну логіку в одному місці, що спрощує її підтримку та зменшує дублювання.