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 №6
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» »**
HTTP-сервер

Виконав:
Студент групи ІА-22
Ю. Білецький С. В.

Перевірив:
Мягкий М.

Київ-2024

Зміст

Тема:
Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Мета:
Навчитися застосовувати шаблони проєктування «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» у процесі розробки програмного забезпечення. Ознайомитися з їхньою концепцією, особливостями реалізації та можливостями використання для створення гнучких і масштабованих рішень.

Завдання:
HTTP-сервер. Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах. **(Factory Method)**

Хід роботи

1. Реалізувати не менше 3-х класів відповідно до обраної теми

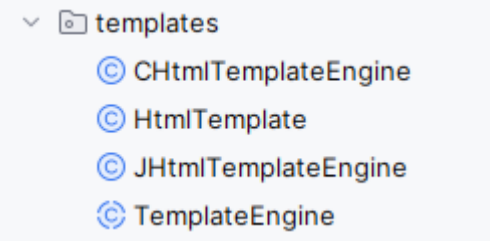

Рис. 1 — Структура класів

У ході виконання лабораторної роботи було реалізовано функціонал 4 класів, що зображені на рис. 1 (певні класи реалізовані частково та мають потенційні можливості для доповнення у зв’язку з тим, що система ще не повністю реалізована). Детальний опис кожного з класів:

1. **TemplateEngine.** Базовий клас для механізму обробки шаблонів. Він визначає основні методи для рендерингу HTML з використанням заданих параметрів. Цей клас служить основою для реалізації конкретних механізмів рендерингу CHTML та HTML-шаблонів, забезпечуючи єдиний інтерфейс для створення шаблонів.
2. **HtmlTemplate.** Клас, який інкапсулює вміст HTML-шаблону і пов'язані з ним змінні (dataBindings), необхідні для підстановки даних. Він містить

методи для отримання вмісту шаблону та переліку змінних, які потрібно заповнити даними. HtmlTemplate виступає як контейнер для шаблону, надаючи зручний інтерфейс для доступу до вмісту та прив'язок даних.

3. **JHtmlTemplateEngine.** Конкретна реалізація TemplateEngine для роботи з шаблонами у форматі HTML. Цей клас відповідає за обробку та рендеринг шаблонів HTML, а також за виявлення та вилучення змінних у шаблоні. Підходить для звичайних HTML-сторінок без специфічних елементів C#.
4. **CHtmlTemplateEngine.** Реалізація TemplateEngine для роботи з CHTML — спеціальним форматом HTML, що підтримує базові конструкції C#. Цей клас використовується для обробки шаблонів, які містять C#-коди, необхідні для динамічного формування сторінок. Він також має методи для виділення змінних, які повинні підставлятися під час рендерингу шаблону.

2. Реалізувати один з розглянутих шаблонів за обраною темою

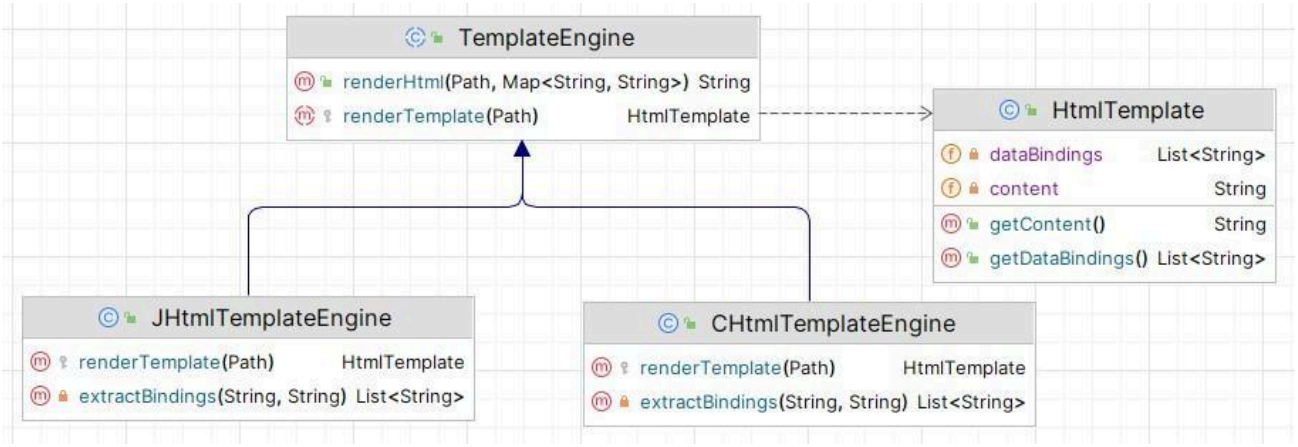

Рис. 2 — Реалізація патерну

У нашому проекті HTTP-сервера ми використали патерн Factory Method для створення об'єктів рендерингу шаблонів (TemplateEngine) — зокрема, для динамічного вибору між JHtmlTemplateEngine і CHtmlTemplateEngine. Це дозволяє серверу вибирати відповідний механізм рендерингу шаблонів залежно від типу сторінки (звичайний HTML або CHTML із вбудованими елементами C#), яка має бути повернута як відповідь на HTTP-запит.

При обробці HTTP-запитів сервер може стикатися з різними типами шаблонів, що потребують специфічної обробки залежно від формату сторінки. Наприклад, стандартний HTML-шаблон обробляється інакше, ніж CHTML-шаблон, який містить елементи коду C#. Використання Factory Method вирішує цю проблему,

дозволяючи серверу створювати об'єкти потрібного типу шаблону під час виконання програми, не змінюючи основну логіку обробки запитів. Це робить систему більш гнучкою та масштабованою, знижуючи залежність між компонентами.

Реалізація

Абстрактний клас TemplateEngine виступає базовим інтерфейсом для всіх механізмів рендерингу. Завдяки Factory Method ми створюємо конкретні екземпляри JHtmlTemplateEngine або CHtmlTemplateEngine відповідно до типу запиту чи формату запитуваного ресурсу. Factory Method інкапсулює логіку вибору відповідного механізму рендерингу, тож сервер не потребує додаткових змін при роботі з різними типами шаблонів. Це спрощує процес додавання нових форматів шаблонів, оскільки для цього потрібно лише створити новий клас, який успадковує TemplateEngine, і визначити його у фабричному методі.

Переваги використання Factory Method:

1. Гнучкість: Додаючи нові типи шаблонів, ми просто реалізуємо новий клас, що успадковує TemplateEngine, без необхідності змінювати існуючий код.
2. Зручне розширення: Factory Method дозволяє зручно масштабувати систему, адже можна легко додавати підтримку нових форматів (наприклад, інші види динамічних сторінок) без змін у клієнтському коді сервера.
3. Інкапсуляція логіки створення: Логіка вибору та створення конкретного типу TemplateEngine ізольована в одному місці, що полегшує підтримку коду й робить його більш зрозумілим.
4. Зниження зв’язності компонентів: Серверний код не залежить від конкретних реалізацій механізмів рендерингу. Це зменшує взаємозалежність між модулями, що покращує підтримуваність і надійність системи.

```java
public abstract class TemplateEngine {  2 usages  2 inheritors

    public String renderHtml(Path page, Map<String, String> data) {  no usages
        HtmlTemplate template = renderTemplate(page);
        String renderedContent = template.getContent();
        for (String entry : template.getDataBindings()) {
            String placeholder = "{{" + entry + "}}";
            renderedContent = renderedContent.replace(placeholder, data.get(entry));
        }
        return renderedContent;
    }

    protected abstract HtmlTemplate renderTemplate(Path page);  1 usage  2 implementations

}
```

Рис. 3 — Реалізація TemplateEngine

Короткий опис: TemplateEngine слугує основою для створення HTML-сторінок із заміною змінних (плейсхолдерів) на задані значення. Абстрактний метод renderTemplate дозволяє підкласам визначати, як завантажується шаблон.

```java
public class JHtmlTemplateEngine extends TemplateEngine {  no usages   2 1 1

    @Override  1 usage
    protected HtmlTemplate renderTemplate(Path page) {
        try {
            String content = Files.readString(page);
            List<String> dataBindings = extractBindings(content,  regex: "\\{\\{jhtml:([^}]+)\\}\\}");
            return new HtmlTemplate(content, dataBindings);
        } catch (IOException e) {
            System.out.println("Помилка!");
            return new HtmlTemplate( content: "", new ArrayList<>());
        }
    }

    private List<String> extractBindings(String content, String regex) {  1 usage
        List<String> bindings = new ArrayList<>();
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(content);
        while (matcher.find()) {
            bindings.add(matcher.group(1).trim());
        }
        return bindings;
```

Рис. 4 — Реалізація JHtmlTemplateEngine

Короткий опис: Завантаження HTML-шаблону з файлу. Витягування

специфічних плейсхолдерів із формату {{jhtml:bindingName}}. Повернення об'єкта HtmlTemplate, що містить вихідний вміст і список прив'язок даних.

Посилання на код проекту: github

Висновки:

У рамках лабораторної роботи ми опрацювали різні шаблони проєктування, зокрема Factory Method, який застосували для динамічного вибору механізму рендерингу шаблонів у HTTP-сервері. Реалізація цього патерну спростила інтеграцію різних форматів сторінок, підвищила гнучкість і масштабованість системи, полегшивши її розширення та підтримку.