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
З дисципліни «Технології розроблення програмного забезпечення»
Тема: **«ШАБЛОНИ «Composite», «Flyweight», «Interpreter», «Visitor»»**
HTTP-сервер

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

Перевірив:
Мягкий М. Ю.

Київ-2024

Зміст

Тема:
Шаблони «Composite», «Flyweight», «Interpreter», «Visitor»

Мета:
Навчитися застосовувати шаблони проєктування «Composite», «Flyweight», «Interpreter», «Visitor» 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. **(Composite)**

Хід роботи

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

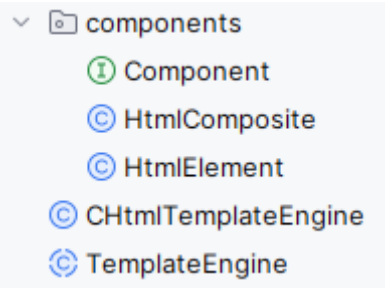

Рис. 1 — Структура класів

У ході виконання лабораторної роботи було створено та оновлено кілька класів, що зображені на рис. 1. Детальний опис кожного з класів:

1. **HtmlComposite.** Цей клас реалізує інтерфейс Component і представляє складовий HTML елемент, що містить інші компоненти (дітей). Він відповідає за формування тегів, їх вкладеність, і рендеринг всього HTML-контенту у вигляді рядка. Клас приймає тег та список компонентів як аргументи, які потім з'єднуються у фінальний HTML-код.
2. **HtmlElement.** Клас реалізує інтерфейс Component і представляє простий HTML-елемент, який не містить вкладених тегів, а лише виводить один

тег із його вмістом. Він приймає тег і вміст як аргументи і рендерить їх у вигляді HTML-рядка.

3. **Component.** Це інтерфейс, що визначає метод render, який повинні реалізовувати всі класи, які будуть рендерити HTML. Цей метод повертає рядок, що представляє HTML-код, який буде використовуватися для генерації сторінок.
4. **TemplateEngine.** Абстрактний клас, що відповідає за основну логіку рендерингу HTML-шаблонів. Містить метод renderHtml, який викликає метод renderTemplate для конкретних реалізацій цього класу. Його завдання — забезпечити процес рендерингу на основі шаблонів, наданих як файли.
5. **CHtmlTemplateEngine.** Конкретна реалізація класу TemplateEngine, що відповідає за рендеринг CHTML-сторінок. Він завантажує вміст файлу з диска, аналізує його, і перетворює у HTML-код з можливістю додавання C# конструкцій (ця частина ще повинна бути додана). Клас містить метод parseHtml, який виконує початковий парсинг HTML-структури і будує дерево компонентів для рендерингу.

2. Реалізувати один з розглянутих шаблонів за обраною темою

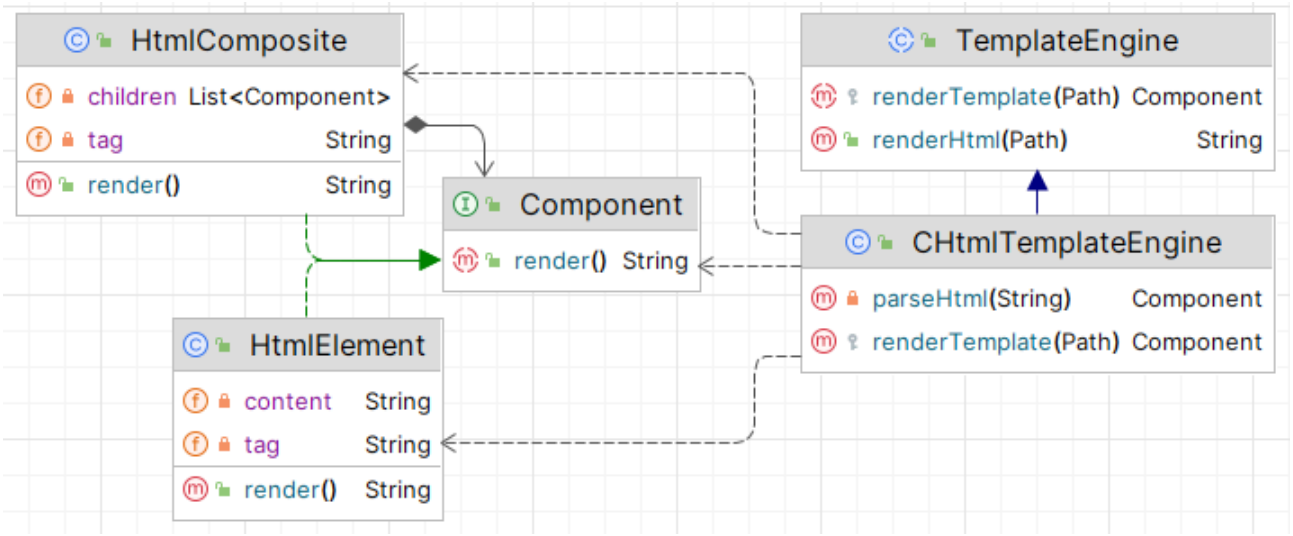

Рис. 2 — Реалізація патерну

У проекті HTTP-серверу патерн Composite був реалізований для вирішення завдання побудови ієрархії HTML-елементів, яка дозволяє легко керувати як простими, так і складними елементами через єдиний інтерфейс. Патерн дозволив зручно рендерити HTML-сторінки з можливістю додавання

динамічних C# конструкцій (CHTML) шляхом об'єднання елементів у дерево компонентів.

Реалізація

```
public interface Component {
    String render();
}
```

Рис. 3 — Реалізація інтерфейсу Component

Короткий опис: Компонент як базовий інтерфейс: Інтерфейс Component визначає загальний метод render, який використовується для рендерингу будь-якого HTML-елемента, незалежно від того, є він простим чи складним. Це дозволяє абстрагуватися від конкретних типів елементів і працювати з усіма однаково.

```
public class HtmlComposite implements Component {
    private String tag;
    private List<Component> children;

    public HtmlComposite(String tag, List<Component> children) {
        this.tag = tag;
        this.children = children;
    }

    @Override
    public String render() {
        StringBuilder html = new StringBuilder();
        html.append("<").append(tag).append(">");
        for (Component child : children) {
            html.append(child.render());
        }
        html.append("</").append(tag).append(">");
        return html.toString();
    }
}
```

Рис. 4 — Реалізація HtmlComposite

Короткий опис: Складові елементи: Клас HtmlComposite відповідає за складні елементи, які можуть містити інші компоненти (дочірні елементи). Таким чином, дерево HTML-елементів може бути побудоване рекурсивно — кожен

складний елемент може включати всередині себе інші елементи (як прості, так і складні), що полегшує рендеринг складних структур, таких як форми, таблиці чи вкладені секції HTML.

```
public class HtmlElement implements Component {
    private String tag;
    private String content;

    public HtmlElement(String tag, String content) {
        this.tag = tag;
        this.content = content;
    }

    @Override
    public String render() { return "<" + tag + ">" + content + "</" + tag + ">"; }
}
```

Рис. 5 — Реалізація HtmlElement

Короткий опис: Прості елементи: Клас HtmlElement відповідає за прості HTML-елементи, які не мають вкладених тегів. Наприклад, це можуть бути теги <p>, <h1>, <img> тощо. Кожен такий елемент рендериться самостійно, без необхідності враховувати вкладеність.

Код проекту: github

Проблеми, які вирішує патерн Composite:

1. Спрощення роботи з вкладеними структурами: Composite дозволяє працювати з ієрархічними HTML-структурами без необхідності окремо обробляти прості та складні елементи. Це значно полегшує роботу з шаблонами HTML та динамічним рендерингом сторінок.
2. Масштабованість: Додавання нових типів елементів можливе без змін у базовій структурі. Це забезпечує легке розширення функціональності для додавання підтримки нових тегів або складніших шаблонів.
3. Єдиний інтерфейс для рендерингу: Завдяки тому, що всі елементи (як прості, так і складні) мають один і той самий інтерфейс, система стає більш гнучкою і легко підтримуваною.

Переваги використання Composite:

1. Легке управління ієрархією HTML: Кожен елемент — це компонент, який може містити інші компоненти, дозволяючи створювати гнучкі і складні HTML-структури.

2. Чистий і зрозумілий код: Composite дозволяє уникнути дублювання логіки рендерингу різних типів елементів, зберігаючи код більш організованим.
3. Гнучкість: Можливість легкого розширення і підтримки нових функцій, таких як додавання підтримки C# конструкцій в CHTML, не змінюючи базової архітектури.

Висновки:

У рамках лабораторної роботи ми реалізували патерн Composite у проекті HTTP-серверу для спрощення рендерингу HTML-сторінок і підтримки динамічних CHTML-шаблонів з C# конструкціями. Патерн дозволяє працювати з HTML-елементами як з єдиною ієрархічною структурою, що складається з простих і складних компонентів. Це рішення забезпечує гнучкість, масштабованість та спрощує обробку вкладених HTML-елементів, що особливо корисно для динамічного формування сторінок та підтримки складних шаблонів.