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: «Патерни проектування»

Виконав
Студент групи ІА-31: Перевірив:
Губар Б. О. Мягкий М. Ю.

Київ 2025

Зміст

1. Мета:

Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

2. Теоретичні відомості

Патерн	Призначення	Ключова ідея	Приклад
Composite	Деревоподібні ієрархії "частина-ціле"	Уніфікована обробка листків і контейнерів	Проєкт → Функція → User Story → Task
Flyweight	Зменшення пам’яті через поділ об’єктів	Внутрішній стан (shared) / Зовнішній (context)	Букви в тексті, графічні примітиви
Interpreter	Інтерпретація граматики мови	Рекурсивне AST-дерево	Пошук за шаблоном, скриптова мова
Visitor	Додавання операцій без зміни елементів	Відокремлення логіки від структури	Розрахунок цін у кошику (з/без знижки)

3. Хід роботи

Тема : CI server (state, command, decorator, mediator, visitor, soa)

Обґрунтування застосування патерну

Для системи реалізовано патерн Visitor (Відвідувач) для генерації звітів про результати збірки. У системі існує складна ієрархічна структура даних: Project містить список Builds, які в свою чергу містять BuildLogs. Нам необхідно виконувати над цією структурою операцію експорту даних (в даному випадку — генерацію HTML-сторінки). Додавання методу ToHtml() безпосередньо в класи сутностей (Project, Build) порушило б принцип єдиної відповідальності (SRP), оскільки бізнес-сутності не повинні відповідати за візуальне представлення. Крім того, якщо в майбутньому знадобиться експорт у JSON або XML, довелося б змінювати всі ці класи. Патерн Visitor дозволив винести алгоритм обходу структури та генерації звіту в окремий клас HtmlReportVisitor. Сутності лише надають метод Accept, який дозволяє відвідувачу отримати доступ до їхнього стану.

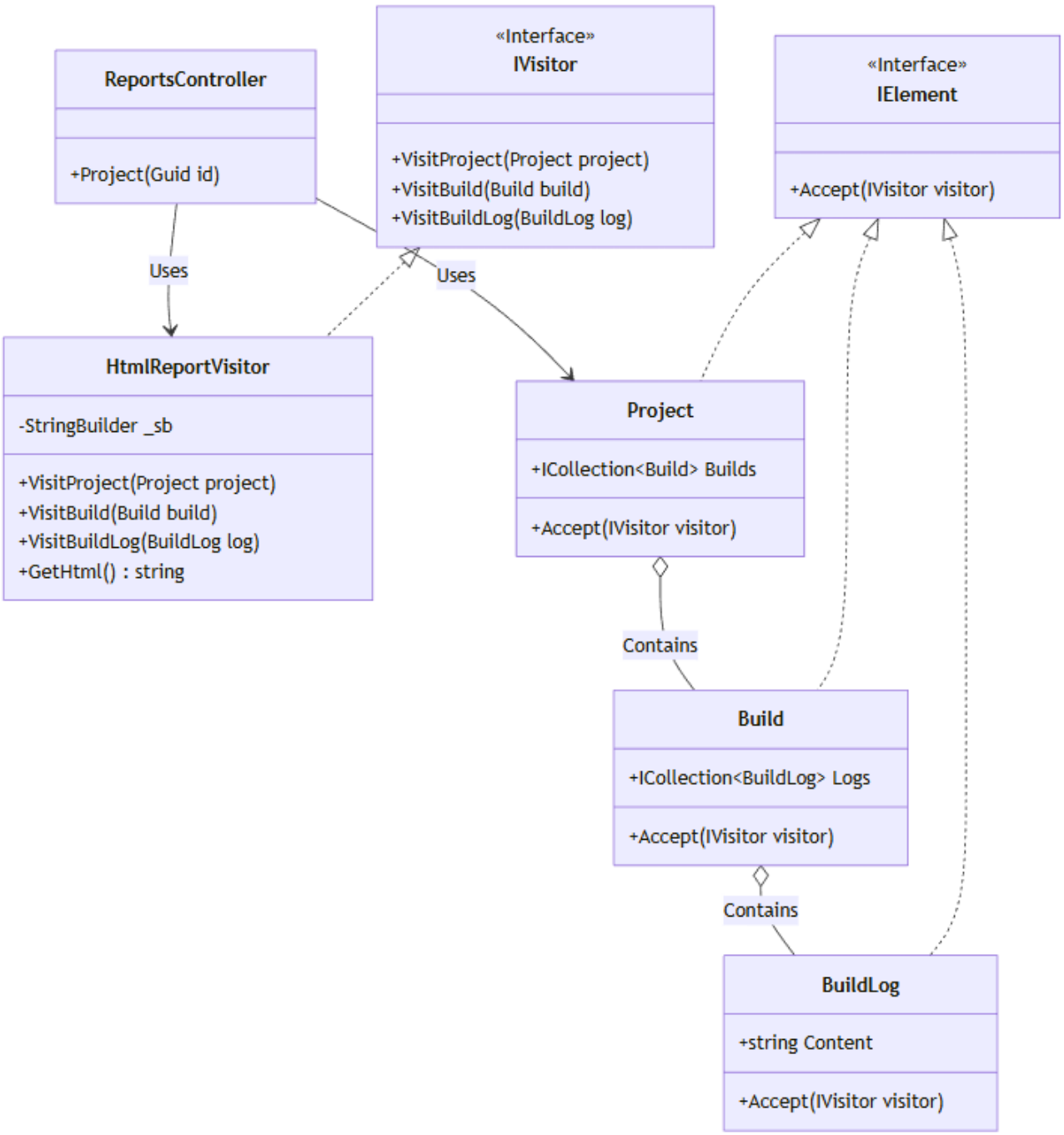


Рисунок 1 - Структура патерну Visitor

- IVisitor (Visitor) – інтерфейс, що оголошує методи відвідування для кожного типу елемента в структурі об'єктів (VisitProject, VisitBuild, VisitBuildLog).
- HtmlReportVisitor (Concrete Visitor) – реалізує логіку генерації звіту. Він накопичує HTML-розмітку у StringBuilder. Для кожного елемента він знає, як його відобразити (наприклад, фарбує статус білда у зелений або червоний колір).
- IElement (Element) – інтерфейс, який зобов'язує класи мати метод Accept(IVisitor visitor).

- Project, Build, BuildLog (Concrete Elements) – класи даних. У методі Accept вони викликають відповідний метод відвідувача (наприклад, visitor.VisitBuild(this)), а потім передають відвідувача своїм дочірнім елементам (наприклад, Build перебирає свої Logs і викликає Accept для кожного). Це забезпечує рекурсивний обхід дерева об'єктів.
- ReportsController (Client) – клієнтський код, який створює відвідувача і запускає процес, викликаючи project.Accept(visitor).

```
using CiServer.Core.Entities;

namespace CiServer.Core.Visitor;

0 references
public interface IVisitor
{
    0 references
    void VisitProject(Project project);
    0 references
    void VisitBuild(Build build);
    0 references
    void VisitBuildLog(BuildLog log);
}
```

Рисунок 2 - Інтерфейс IVisitor.cs

```
namespace CiServer.Core.Visitor;

0 references
public interface IElement
{
    0 references
    void Accept(IVisitor visitor);
}
```

Рисунок 3 - Інтерфейс IElement.cs

```
public void VisitBuild(Build build)
{
    string color = build.Status == BuildStatus.Success ? "green" :
                   build.Status == BuildStatus.Pending ? "gray" :
                   build.Status == BuildStatus.Running ? "blue" : "red";

    _sb.AppendLine("<li>");
    _sb.AppendLine($"<strong>Build ID:</strong> {build.BuildId} <br/>");
    _sb.AppendLine($"Status: <span style='color:{color}'>{build.Status}</span> <br/>");
    _sb.AppendLine($"Time: {build.StartTime}");
    _sb.AppendLine("<div style='margin-left: 20px; font-family: monospace; background: □#f0f0f0; padding: 5px;'>");
}
```

Рисунок 4 – Метод VisitBuild, де відбувається вибір кольору залежно від статусу

```
public void Accept(IVisitor visitor)
{
    visitor.VisitProject(this);
    foreach (var build in Builds)
    {
        build.Accept(visitor);
    }
}
```

Рисунок 5 – Реалізація структури

```
var visitor = new HtmlReportVisitor();
project.Accept(visitor);

return Content(visitor.GetHtml(), "text/html");
```

Рисунок 6 - ReportsController.cs

Project Report: 7654

Repository: https://github.com/gfnf9977/aishki

Build History:

- **Build ID:** 59a74616-1f33-4e1d-bb85-8fbc23c1deb5
  Status: Failed
  Time: 08.12.2025 15:01:08

```
[15:01:10] [Start] CloneRepositoryCommand
[15:01:12] [Success] CloneRepositoryCommand
[15:01:12] [Start] CompileCodeCommand
[15:01:12] [Error] CompileCodeCommand: Compilation failed.
```

Рисунок 7 – Результат

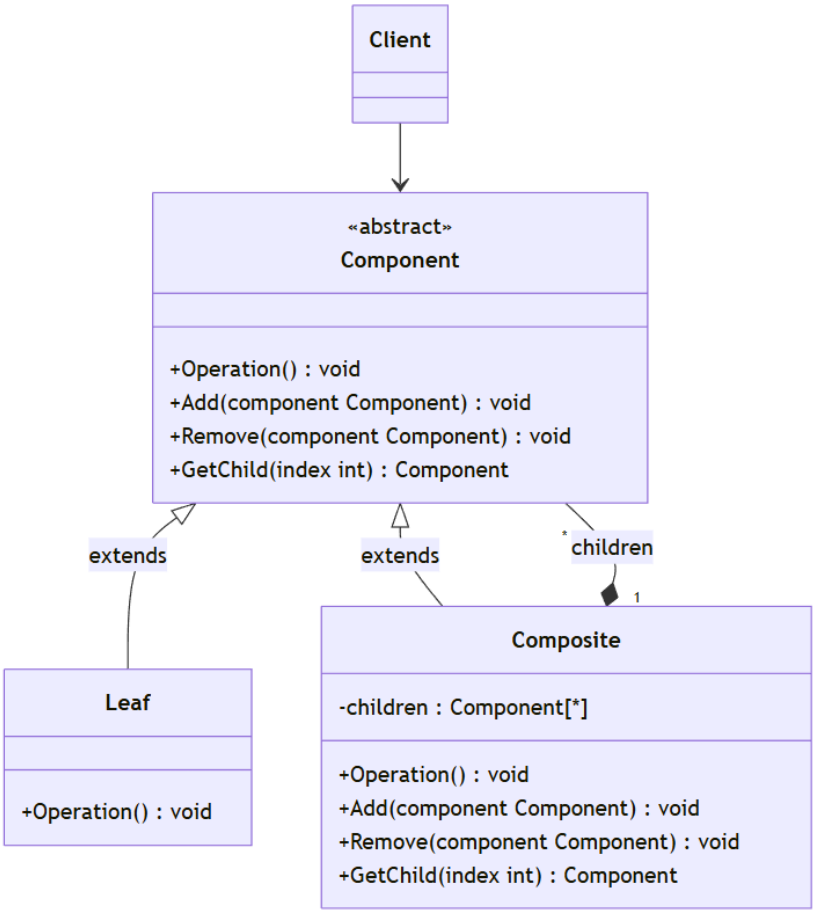
4. Висновок

У ході виконання лабораторної роботи було розглянуто поведінкові шаблони проєктування. На прикладі системи «CI Server» було реалізовано шаблон **Visitor** для генерації детальних звітів про виконання збірок. Завдяки впровадженню цього патерну вдалося відокремити алгоритм обходу складної деревоподібної структури об'єктів (Project -> Build -> BuildLog) від самих класів даних. Клас HtmlReportVisitor інкапсулює всю логіку форматування HTML, включаючи кольорову індикацію статусів. Такий підхід забезпечує дотримання принципу відкритості/закритості (OCP): якщо в майбутньому знадобиться експортувати звіт у формат JSON або PDF, достатньо буде створити новий клас відвідувача (наприклад, JsonReportVisitor), не змінюючи жодного рядка в існуючих класах сутностей.

5. Контрольні питання

1. Яке призначення шаблону «Композит»? Шаблон використовується для складання об’єктів у деревоподібну структуру для подання ієрархій типу «частина — ціле». Дозволяє уніфіковано обробляти поодинокі об’єкти та композиції.

2. Нарисуйте структуру шаблону «Композит».

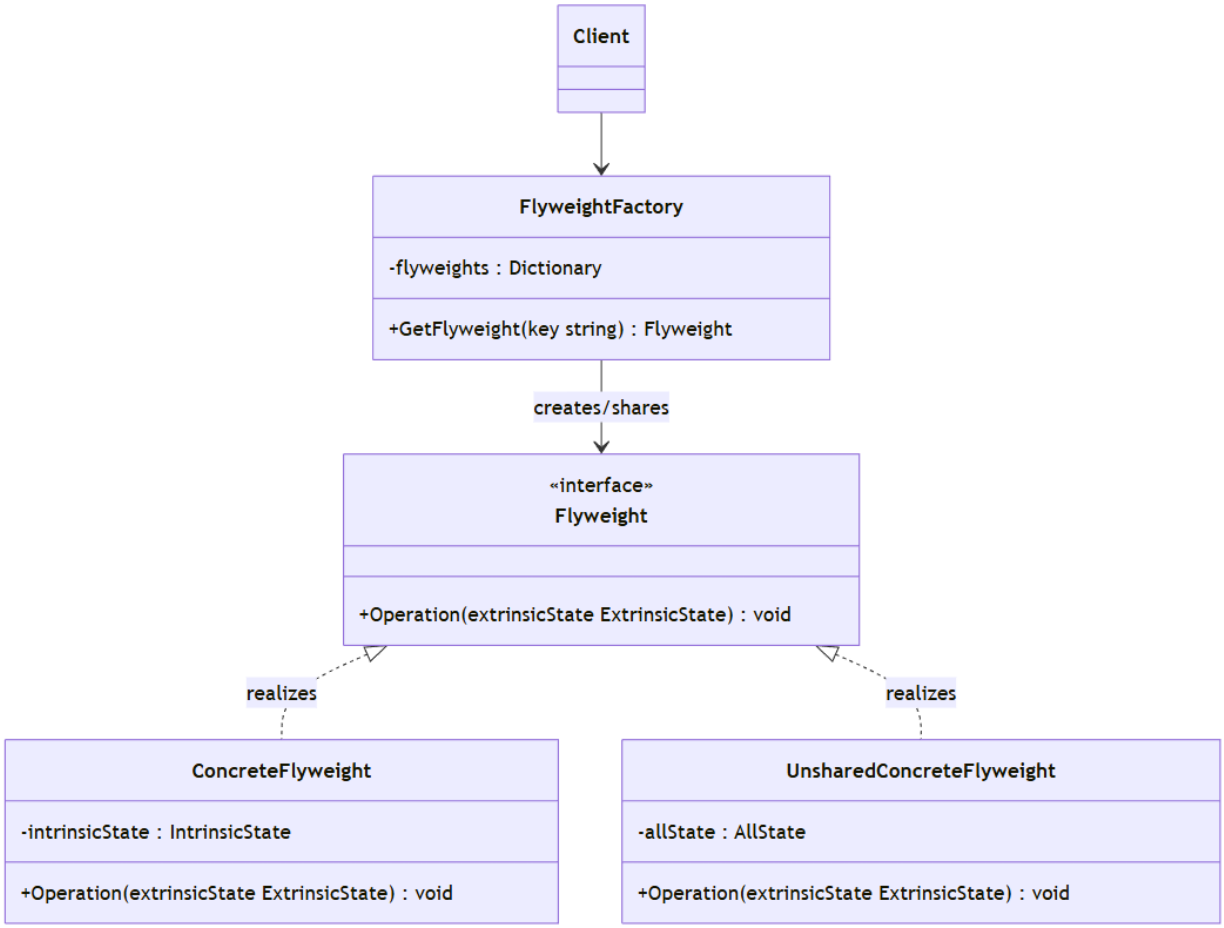


3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

- Component — абстрактний клас/інтерфейс з операціями.
- Leaf — кінцевий елемент, реалізує операції.
- Composite — контейнер, містить children, рекурсивно делегує операції. Взаємодія: клієнт працює з Component, не розрізняючи Leaf і Composite.

4. Яке призначення шаблону «Легковаговик»? Зменшення кількості об’єктів у пам’яті шляхом поділу спільного (внутрішнього) стану між кількома екземплярами.

5. Нарисуйте структуру шаблону «Легковаговик».



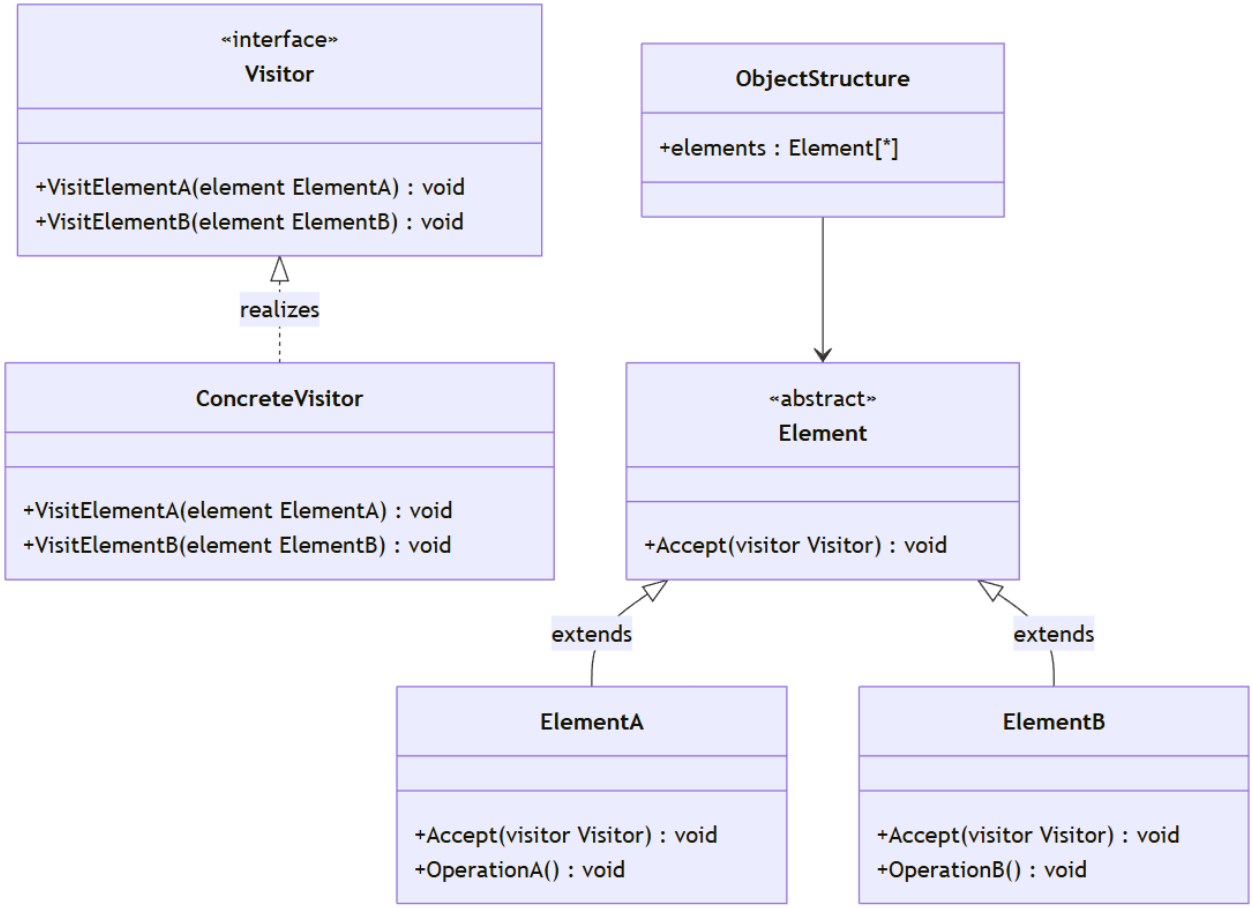
6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

- Flyweight — інтерфейс з операцією, що приймає зовнішній стан.
- ConcreteFlyweight — зберігає внутрішній стан, спільний.
- UnsharedConcreteFlyweight — не поділюваний варіант.
- FlyweightFactory — кешує та повертає Flyweight за ключем. Взаємодія: клієнт отримує Flyweight з фабрики, передає зовнішній стан у метод.

7. Яке призначення шаблону «Інтерпретатор»? Подання граматики мови та інтерпретатора для обчислення виразів у термінах абстрактного синтаксичного дерева (AST).

8. Яке призначення шаблону «Відвідувач»? Дозволяє додавати нові операції над елементами ієрархії без зміни їх класів, відокремлюючи логіку від структури.

9. Нарисуйте структуру шаблону «Відвідувач».



10. Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

- Visitor — інтерфейс з методами Visit... для кожного типу елемента.
- ConcreteVisitor — реалізує операції.
- Element — інтерфейс з Accept(Visitor).
- ConcreteElement — викликає відповідний Visit... у відвідувача.
- ObjectStructure — колекція елементів. Взаємодія: елемент викликає visitor.Visit(this), відвідувач виконує логіку за типом.