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»

Варіант - 17

Виконав
Студент групи ІА-31:
Лисенко В. Є

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. Мета: 3
2. Теоретичні відомості: 3
3. Завдання: 3
4. Хід роботи: 3
Програмна реалізація: 5
5. Висновок 7
6. Контрольні питання: 8

1. Мета:

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

2. Теоретичні відомості:

Composite (Композит): Шаблон для побудови деревоподібних ієрархій "частина-ціле", що дозволяє уніфіковано обробляти як окремі об’єкти (листки), так і їхні контейнери. Ключова ідея: Уніфікована обробка листків і контейнерів. Приклад: Проєкт → Функція → User Story → Task.

Flyweight (Легковаговик): Шаблон для зменшення витрат пам’яті шляхом спільного використання об’єктів з однаковим внутрішнім станом. Ключова ідея: Поділ стану на внутрішній (shared) і зовнішній (context). Приклад: Букви в тексті, графічні примітиви.

Interpreter (Інтерпретатор): Шаблон для реалізації інтерпретації граматики мови, використовуючи рекурсивне дерево абстрактного синтаксису (AST). Ключова ідея: Рекурсивне AST-дерево для обробки виразів. Приклад: Пошук за шаблоном, скриптова мова.

Visitor (Відвідувач): Шаблон для додавання нових операцій до структури об’єктів без зміни їхніх класів. Ключова ідея: Відокремлення логіки операцій від структури об’єктів. Приклад: Розрахунок цін у кошику (з урахуванням знижки або без неї).

3. Завдання:

1) Ознайомитись з короткими теоретичними відомостями.
2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.
5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

4. Хід роботи:

Мій варіант:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor, SOA) Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Паттерн **Visitor** дозволяє:

1. Відокремити алгоритми обробки даних від самих об'єктів даних .
2. Додавати нові операції (наприклад, експорт у JSON) без жодної зміни в існуючих класах метрик (дотримуючись принципу Open/Closed).
3. Об'єднати споріднені операції в одному класі (наприклад, вся логіка аналітики зосереджена в AnalysisVisitor), замість того щоб розкидати її по десятках класів.

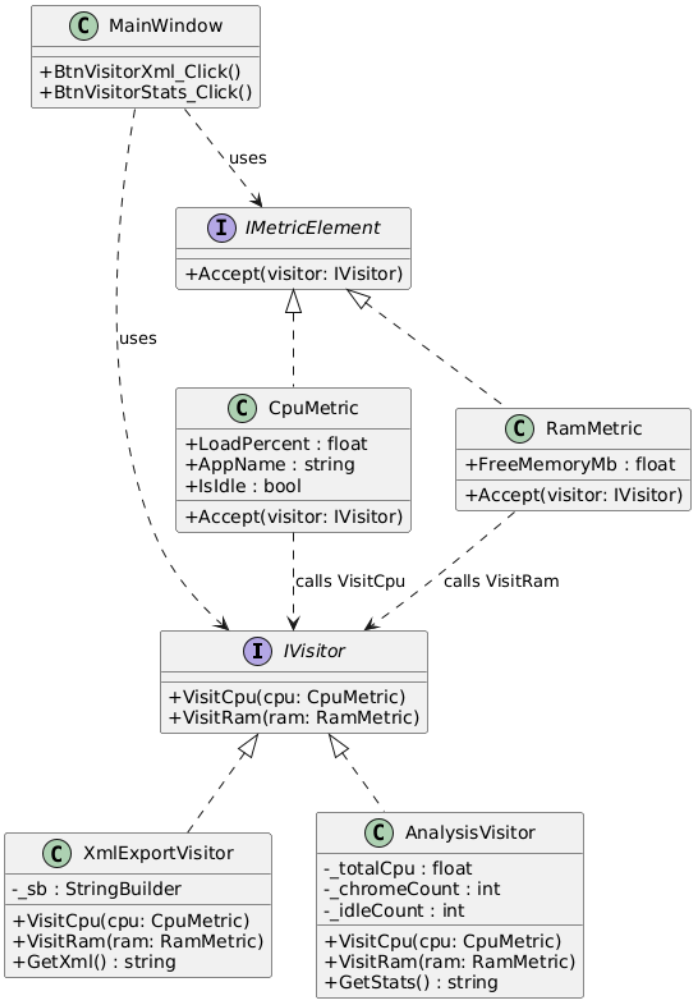


Рисунок 1 - Структура патерну Visitor

IVisitor	Інтерфейс відвідувача	Оголошує методи відвідування для кожного конкретного типу елемента (VisitCpu, VisitRam).
XmlExportVisitor / AnalysisVisitor	Конкретні відвідувачі	Реалізують логіку обробки. XmlExportVisitor формує текстове представлення, а AnalysisVisitor накопичує статистику (сумує навантаження, рахує кількість записів простою).
IMetricElement	Інтерфейс елемента	Оголошує метод Accept(visitor), який приймає відвідувача.
CpuMetric / RamMetric	Конкретні елементи	Класи даних. У методі Accept вони викликають відповідний метод відвідувача (visitor.VisitCpu(this)), реалізуючи техніку подвійної диспетчеризації (Double Dispatch) .
MainWindow	Клієнт	Створює структуру об'єктів (список метрик) і запускає по ній відвідувача.

Програмна реалізація:

IVisitor.cs:

```
namespace SystemActivityMonitor.Data.Patterns.Visitor
{
    public interface IVisitor
    {
        void VisitCpu(CpuMetric cpu);
        void VisitRam(RamMetric ram);
    }
}
```

CpuMetric.cs:

```
namespace SystemActivityMonitor.Data.Patterns.Visitor
{
    public class CpuMetric : IMetricElement
    {
        public float LoadPercent { get; set; }
        public string Time { get; set; }
        public string AppName { get; set; }
        public bool IsIdle { get; set; }

        public CpuMetric(float load, string time, string appName, bool isIdle)
        {
            LoadPercent = load;
            Time = time;
            AppName = appName;
```

```
            IsIdle = isIdle;
        }
        public void Accept(IVisitor visitor)
        {
            visitor.VisitCpu(this);
        }
    }
}
```

AnalysisVisitor.cs:

```
public class AnalysisVisitor : IVisitor
    {
        private float _totalCpu = 0;
        private int _count = 0;
        private int _chromeCount = 0;
        private int _idleCount = 0;
        private float _minRam = float.MaxValue;

        public void VisitCpu(CpuMetric cpu)
        {
            _totalCpu += cpu.LoadPercent;
            _count++;

            if (cpu.AppName == "Google Chrome")
                _chromeCount++;

            if (cpu.IsIdle)
                _idleCount++;
        }

        public void VisitRam(RamMetric ram)
        {
            if (ram.FreeMemoryMb < _minRam)
                _minRam = ram.FreeMemoryMb;
        }

        public string GetStats()
        {
            float avgCpu = _count > 0 ? _totalCpu / _count : 0;
            float browserPercent = _count > 0 ? ((float)_chromeCount / _count) * 100 : 0;
            float idlePercent = _count > 0 ? ((float)_idleCount / _count) * 100 : 0;

            return $"Аналіз завершено:\n" +
                   $" - Середнє CPU: {avgCpu:F2}%\n" +
```

```
                $" - Час у браузері: {browserPercent:F2}%\n" +
                $" - Час простою (Idle): {idlePercent:F2}%\n" +
                $" - Мінімум RAM: {_minRam} MB";
        }
    }
```

MainWindow.xaml.cs:

```
private void BtnVisitorStats_Click(object sender, RoutedEventArgs e)
        {
            var structure = PrepareElements();
            var visitor = new AnalysisVisitor();

            foreach (var element in structure)
            {
                element.Accept(visitor);
            }

            MessageBox.Show(visitor.GetStats(), "Результат Visitor (Stats)");
        }
```

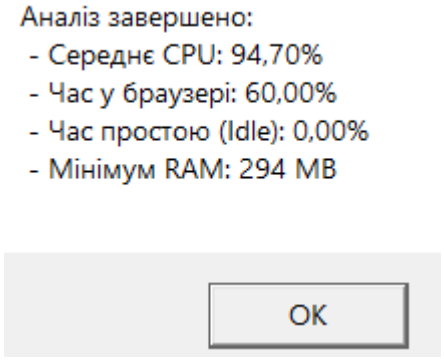


Рисунок 2 – Результат роботи аналітичного відвідувача

5. Висновок

У ході виконання лабораторної роботи було вивчено та практично застосовано поведінковий патерн проєктування Visitor (Відвідувач) . Цей шаблон було використано для вирішення задачі виконання різнорідних операцій над об'єктами метрик системного моніторингу без зміни їхніх класів.

В рамках системи було реалізовано:

Інтерфейс IVisitor та механізм подвійної диспетчеризації (Double Dispatch) через метод Accept у класах даних CpuMetric та RamMetric.

Два типи відвідувачів:

XmlExportVisitor — для серіалізації даних у формат XML.

AnalysisVisitor — для виконання складних розрахунків (середнє навантаження, виявлення простою системи, аналіз активності програм).

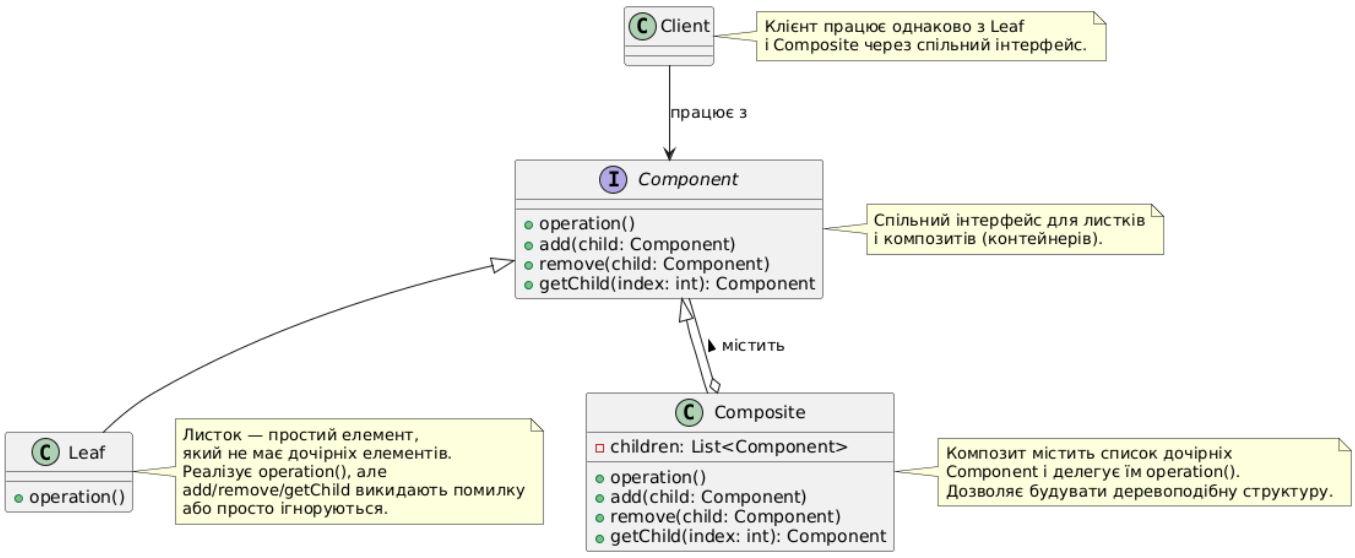
Застосування патерну Visitor дозволило чітко розділити структуру даних і алгоритми їх обробки. Це дало змогу реалізувати нові аналітичні функції, необхідні за варіантом завдання (наприклад, підрахунок % часу у браузері), не порушуючи цілісність існуючих класів та дотримуючись принципу відкритості/закритості (OCP).

6. Контрольні питання:

1. Яке призначення шаблону «Посередник»?

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».

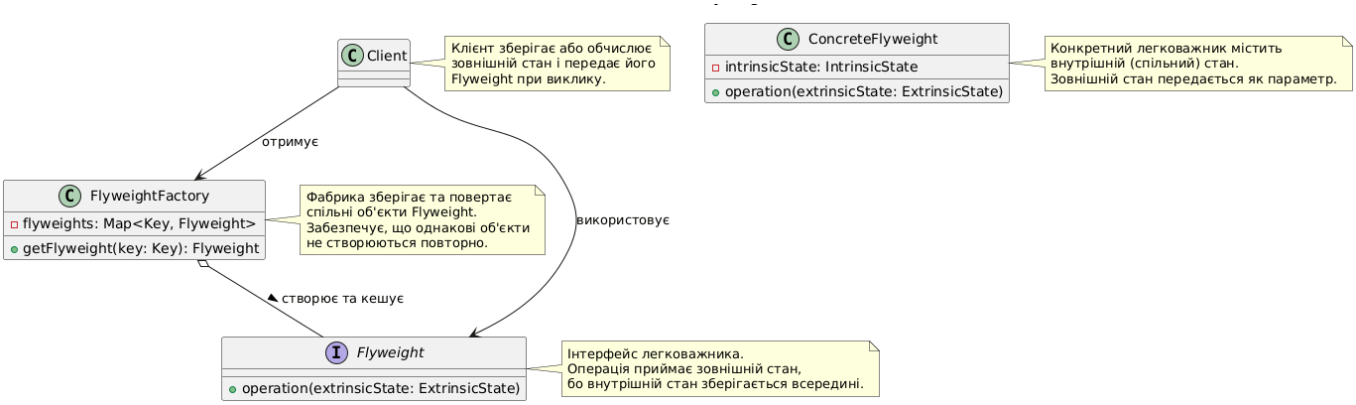


3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

- 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».



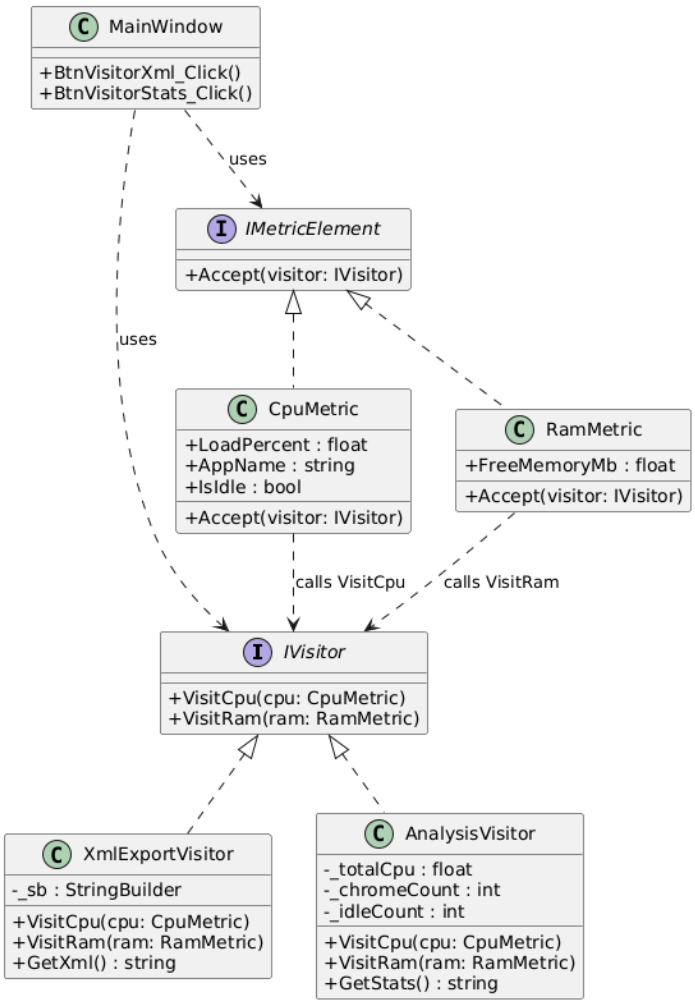
6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

- 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.