Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №8

«Патерни проектування»

З дисципліни «**Технології розроблення програмного забезпечення**»

Виконав:

Студент групи ІА-31

Дук М. Д.

Перевірив:

Мягкий М. Ю.

Київ 2025

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

Технічне завдання:
Архіватор (strategy, adapter, factory method, facade, visitor)
Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, тощо) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Компонувальник» (Composite)

Призначення: Використовується для складання об'єктів у деревоподібну структуру для представлення ієрархій типу «частина-ціле». Дозволяє клієнтам уніфіковано обробляти як окремі об'єкти, так і їхні групи (складені об'єкти) .

Переваги:

- Спрощує клієнтський код, оскільки клієнту не потрібно знати, чи працює він з простим об'єктом, чи зі складним.
- Полегшує додавання нових видів компонентів, оскільки вони автоматично працюють з існуючими структурами та клієнтським кодом.
- Забезпечує гнучкість у роботі зі складними рекурсивними структурами.

Недоліки:

- Може зробити дизайн надто загальним (важко обмежити типи компонентів, які можуть бути додані до композиту).

Шаблон «Пристосуванець» (Flyweight)

Призначення: Використовується для ефективної підтримки великої кількості дрібних об'єктів шляхом поділу спільного стану між ними (внутрішній стан) замість зберігання всіх даних у кожному об'єкті .

Переваги:

- Значно економить оперативну пам'ять при роботі з великою кількістю схожих об'єктів.

Недоліки:

- Витрачає процесорний час на обчислення або передачу зовнішнього контексту.
- Ускладнює код програми через введення додаткових класів та поділ логіки на внутрішній і зовнішній стани

Шаблон «Інтерпретатор» (Interpreter)

Призначення: Використовується для визначення граматики простої мови та створення інтерпретатора, який розбирає та виконує речення цієї мови. Граматика представляється у вигляді класів для правил .

Переваги:

- Легко розширювати та змінювати граматику, додаючи нові класи виразів.
- Реалізація методів інтерпретації у вузлах синтаксичного дерева зазвичай проста.

Недоліки:

- Складно підтримувати граматики з великою кількістю правил (призводить до вибухового зростання кількості класів).
- Неефективний для складних мов (краще використовувати парсери або компілятори)

Шаблон «Відвідувач» (Visitor)
Призначення: Дозволяє додавати нові операції над об'єктами певної структури, не змінюючи класи цих об'єктів. Досягається шляхом подвійної диспетчеризації.

Переваги:

- Спрощує додавання нових операцій (потрібно лише створити нового відвідувача).
- Об'єднує споріднені операції в одному класі, відокремлюючи алгоритм від структури даних.

Недоліки:

- Важко додавати нові класи елементів в ієрархію, оскільки це вимагає зміни інтерфейсу всіх відвідувачів.
- Порушує інкапсуляцію, оскільки відвідувачу часто потрібен доступ до внутрішнього стану елементів.

Хід Роботи

В рамках даної лабораторної роботи було обрано патерн «Visitor» (Відвідувач).

Призначення: Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів. Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

Основна ідея: Рознести логіку і дані в різні класи. Це дозволяє додавати нові операції (наприклад, експорт, валідацію, хешування) шляхом створення нових класів-відвідувачів, не чіпаючи класи, що зберігають дані.

Переваги: Зручно додавати нові операції.

Недоліки: Важко додавати нові класи елементів в ієрархію, оскільки це вимагає оновлення інтерфейсу всіх відвідувачів.

Необхідно було реалізувати функціонал для роботи з архівами, який вимагає виконання різних операцій над записами архіву (файлами та папками), таких як перевірка цілісності та розрахунок хеш-суми. Для уникнення "забруднення" класу моделі ArchiveEntry логікою бізнес-операцій, було вирішено застосувати патерн Visitor.

Для цього було створено інтерфейс IArchiveVisitor, який визначає контракт для відвідування запису архіву.

Реалізація конкретних відвідувачів:

- ChecksumVisitor: для обчислення SHA256 хешу вмісту.
- TestVisitor: для перевірки можливості читання файлів (тестування архіву).

Також було реалізовано механізм прийняття відвідувача (AcceptVisitorAsync) у стратегіях адаптерів (ZipAdapter, TarAdapter) та фасаді ArchiveManager.

Сутність реалізації

В проекті реалізовано патерн Visitor для відділення алгоритмів обробки даних архіву від структури самих даних.

Основні компоненти реалізації (код реалізованого шаблону):

1. Абстракція Відвідувача (IArchiveVisitor): Визначає метод Visit, який приймає метадані запису (ArchiveEntry) та потік даних (Stream).

```
namespace ArchiverCore.Interfaces;

/// <summary>
/// Інтерфейс патерну Відвідувач для виконання операцій над записами архіву.
/// Дозволяє додавати нові операції без зміни класів записів.
/// </summary>
5 usages  2 inheritors  PriMerro23
public interface IArchiveVisitor
{
    /// <summary>
    /// Відвідує запис архіву та виконує операцію.
    /// </summary>
    /// <param name="entry">Метадані запису архіву</param>
    /// <param name="contentStream">Потік для читання вмісту запису (може бути null для директорій)</param>
    3 usages  2 implementations  PriMerro23
    void Visit(Models.ArchiveEntry entry, Stream? contentStream);

    /// <summary>
    /// Повертає результат операції відвідувача (формат залежить від реалізації відвідувача).
    /// </summary>
    2 implementations  PriMerro23
    object GetResult();
}
```

Рис. 1 - IArchiveVisitor

2. Конкретний Відвідувач 1 (ChecksumVisitor): Реалізує логіку обчислення хешу SHA256. Він накопичує дані з усіх відвіданих файлів.

```
using ArchiverCore.Interfaces;
using ArchiverCore.Models;

namespace ArchiverCore.Visitors;

/// <summary>
/// Відвідувач, який обчислює контрольну суму SHA256 вмісту архіву.
/// Реалізує патерн Відвідувач для перевірки цілісності архіву.
/// </summary>
9 usages  PriMerro23
public class ChecksumVisitor : IArchiveVisitor, IDisposable
{
    private readonly SHA256 _sha256 = SHA256.Create();
    private readonly MemoryStream _combinedStream = new();

    5+3 usages  PriMerro23
    public void Visit(ArchiveEntry entry, Stream? contentStream)
    {
        if (entry.IsDirectory || contentStream == null)
            return;

        // Додаємо ім'я запису до контрольної суми
        var nameBytes :byte[] = Encoding.UTF8.GetBytes(entry.FullName);
        _combinedStream.Write( nameBytes);

        // Додаємо вміст до контрольної суми
        contentStream.CopyTo(_combinedStream);
    }

    9 usages  PriMerro23
    public object GetResult()
    {
        _combinedStream.Position = 0;
        var hash :byte[] = _sha256.ComputeHash(_combinedStream);
        return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
    }

    9 usages  PriMerro23
    public void Dispose()
    {
        _sha256?.Dispose();
        _combinedStream?.Dispose();
```

Рис. 2 - ChecksumVisitor

3. Конкретний Відвідувач 2 (TestVisitor): Намагається прочитати потік даних, щоб переконатися, що файл не пошкоджений.

```
using ...

namespace ArchiverCore.Visitors;

/// <summary>
/// Відвідувач, який перевіряє цілісність архіву, намагаючись прочитати всі записи.
/// Реалізує патерн Відвідувач для валідації архіву.
/// </summary>
1 usage  PriMerro23
public class TestVisitor : IArchiveVisitor
{
    private readonly TestResult _result = new() { IsValid = true };

    0+3 usages  PriMerro23
    public void Visit(ArchiveEntry entry, Stream? contentStream)
    {
        try
        {
            if (!entry.IsDirectory && contentStream != null)
            {
                // Намагаємося прочитати потік, щоб перевірити, що він не пошкоджений
                var buffer = new byte[8192];
                while (contentStream.Read(buffer, offset: 0, count: buffer.Length) > 0)
                {
                    // Просто читаємо для перевірки
                }
            }

            _result.EntriesChecked++;
        }
        catch (Exception ex)
        {
            _result.IsValid = false;
            _result.Errors.Add($"Error testing '{entry.FullName}': {ex.Message}");
        }
    }

    1 usage  PriMerro23
    public object GetResult()
    {
        return _result;
    }
```

Рис. 3 - TestVisitor

4. Клієнтський код (використання в ArchiveManager): Менеджер створює потрібного відвідувача і передає його адаптеру.

```
2 usages  PriMerro23
public async Task<string> CalculateChecksumAsync(
    string archivePath,
    CancellationToken cancellationToken = default)
{
    if (!File.Exists(archivePath))
        throw new FileNotFoundException("Archive not found",  fileName: archivePath);

    var adapter :IArchiveStrategy  = ArchiveFactory.CreateFor(archivePath);
    using var checksumVisitor = new ChecksumVisitor();
    await adapter.AcceptVisitorAsync(archivePath, checksumVisitor, cancellationToken: cancellationToken);

    return (string)checksumVisitor.GetResult();
}

3 usages  PriMerro23
private async Task UpdateArchiveMetadataAsync(
    string archivePath,
    string checksum,
    Interfaces.IArchiveStrategy adapter,
    CancellationToken cancellationToken)
{
    if (_repository == null){...}

    try{...}
    catch (Exception ex)
    {
        // Логуємо повну помилку
        System.Diagnostics.Debug.WriteLine($"ERROR updating metadata: {ex.GetType().Name}: {ex.Message}");
        System.Diagnostics.Debug.WriteLine($"Stack trace: {ex.StackTrace}");
        if (ex.InnerException != null)
        {
            System.Diagnostics.Debug.WriteLine($"Inner exception: {ex.InnerException.Message}");
        }
        throw; // Повторно викидаємо помилку для відображення в UI
    }
}
```

Рис. 4 - ArchiveManager

Переваги та недоліки використання в проекті

Переваги:

- Принцип єдиної відповідальності (SRP): Клас ArchiveEntry залишається простим DTO (Data Transfer Object) і не містить складної логіки хешування або валідації.

- Розширюваність (Open/Closed Principle): Додано нову функціональність (тестування архіву TestVisitor) без зміни існуючого коду адаптерів чи моделей. Якщо знадобиться експорт списку файлів у XML/JSON, достатньо буде створити XmlExportVisitor.

- Акумуляція стану: Відвідувачі (ChecksumVisitor, TestVisitor) можуть накопичувати стан (загальний хеш або список помилок) під час обходу архіву, що складно зробити при звичайному ітеративному підході без зовнішніх змінних.

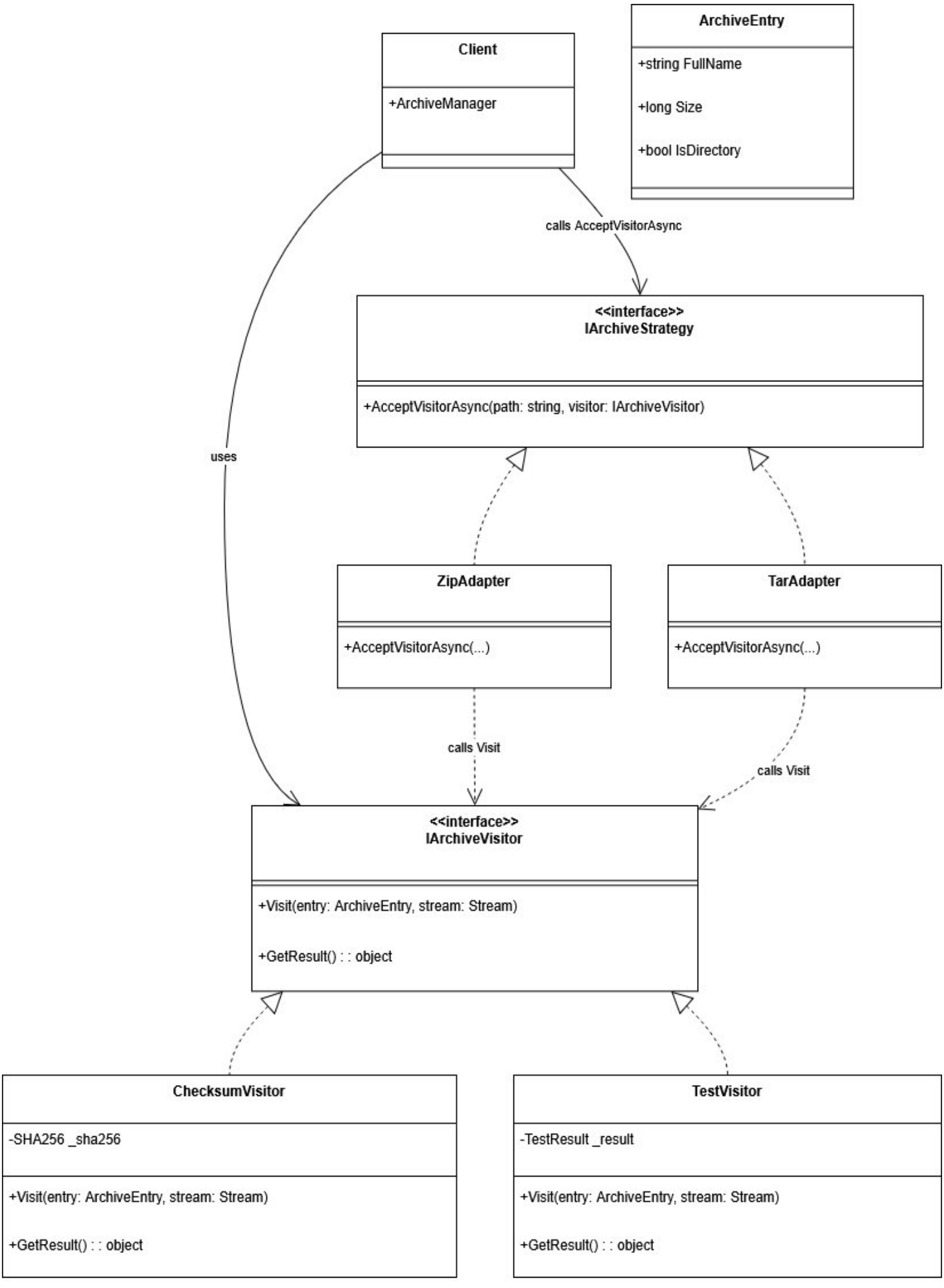
Недоліки:

- Розкриття інкапсуляції: Метод Visit приймає Stream, що дає відвідувачу повний доступ до вмісту файлу. Це необхідно для роботи, але потенційно небезпечно, якщо реалізація відвідувача некоректна (наприклад, закриє потік завчасно).

- Залежність від інтерфейсу: Якщо зміниться сигнатура методу Visit (наприклад, додасться асинхронність VisitAsync), доведеться правити всі існуючі відвідувачі.

Діаграма Класів реалізованого шаблону

Нижче наведена діаграма класів, що ілюструє застосування патерну Visitor у проекті ArchiverSolution.

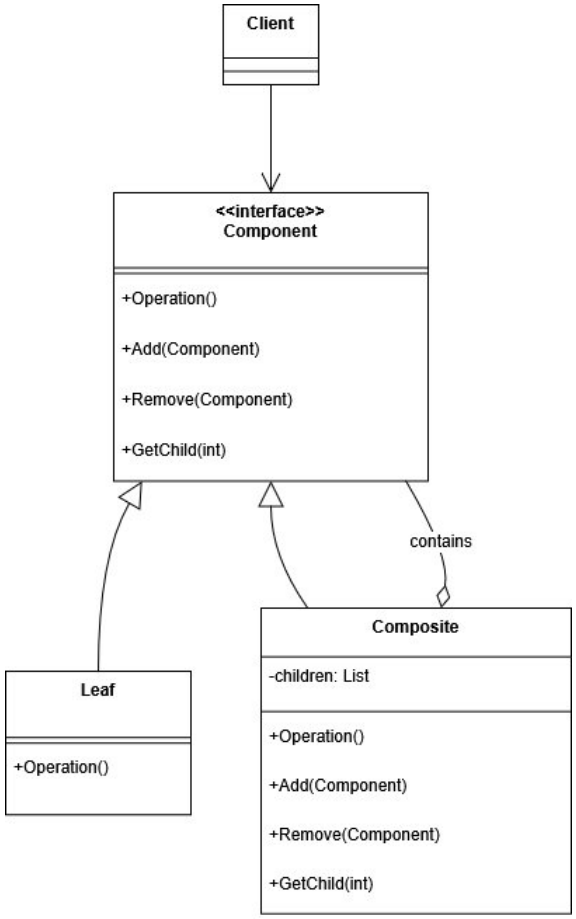

ArchiveEntry
+string FullName
+long Size
+bool IsDirectory
Client
+ArchiveManager
calls AcceptVisitorAsync
<<interface>>
IArchiveStrategy
+AcceptVisitorAsync(path: string, visitor: IArchiveVisitor)
uses
ZipAdapter
+AcceptVisitorAsync(...)
TarAdapter
+AcceptVisitorAsync(...)
calls Visit
calls Visit
<<interface>>
IArchiveVisitor
+Visit(entry: ArchiveEntry, stream: Stream)
+GetResult() : : object
ChecksumVisitor
-SHA256 _sha256
+Visit(entry: ArchiveEntry, stream: Stream)
+GetResult() : : object
TestVisitor
-TestResult _result
+Visit(entry: ArchiveEntry, stream: Stream)
+GetResult() : : object

Висновок: В ході виконання лабораторної роботи було проаналізовано та практично застосовано патерн проектування Visitor (Відвідувач). Реалізація цього патерну в проекті архіватора дозволила винести операції перевірки цілісності та розрахунку контрольних сум в окремі класи. Це значно спростило підтримку коду та дозволило дотримуватися принципів SOLID, зокрема принципу відкритості/закритості, оскільки додавання нових операцій над архівом тепер не потребує модифікації класів, що відповідають за структуру архіву.

Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?
   Призначення шаблону «Композит» полягає у групуванні об'єктів у деревоподібні структури для представлення ієрархій «частина-ціле». Це дозволяє клієнтському коду працювати з одиничними об'єктами та їх групами (композитами) однаково, через спільний інтерфейс .

2. Нарисуйте структуру шаблону «Композит».

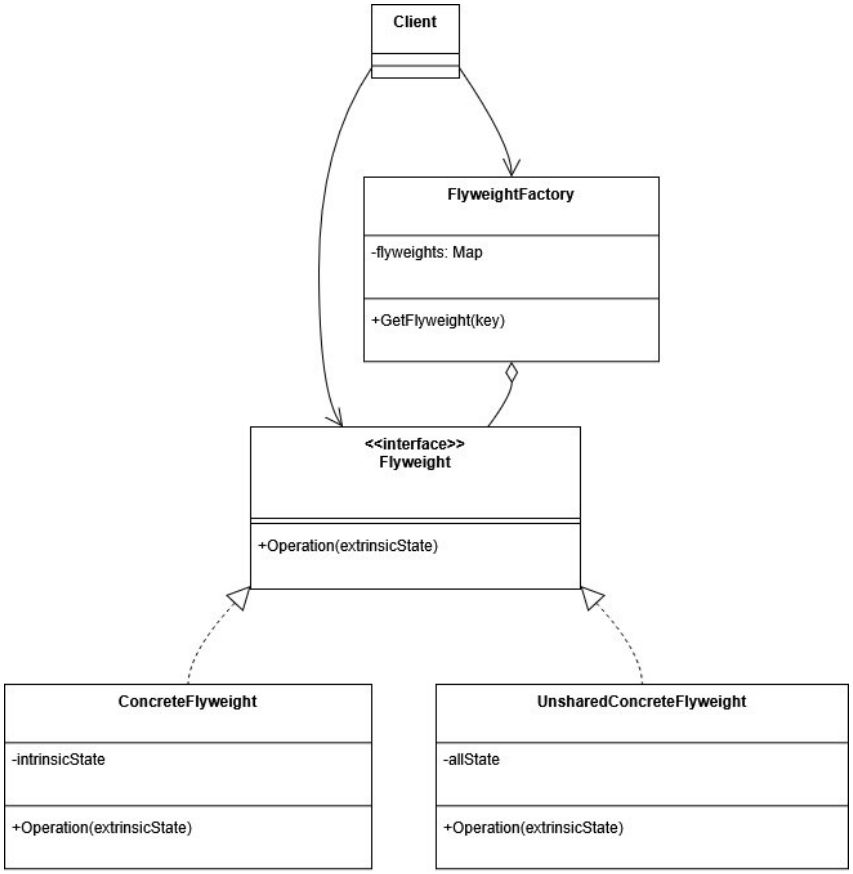


3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

   - Component (Компонент): Оголошує спільний інтерфейс для простих і складних об'єктів структури.
   - Leaf (Лист): Представлє кінцевий об'єкт, який не має підлеглих. Реалізує основну логіку.
   - Composite (Композит): Зберігає колекцію дочірніх компонентів (і листків, і інших композитів) та реалізує методи управління ними (Add, Remove). У методі Operation він зазвичай делегує виконання своїм дочірнім елементам.
   - Client (Клієнт): Працює з усіма елементами через інтерфейс Component.

4. Яке призначення шаблону «Легковаговик»?
   Шаблон «Легковаговик» (Flyweight) призначений для зменшення кількості об'єктів у додатку шляхом спільного використання їхнього стану. Він розділяє стан об'єкта на внутрішній (незмінний, спільний) та зовнішній (контекстний, передається ззовні), що дозволяє економити пам'ять.

5. Нарисуйте структуру шаблону «Легковаговик».



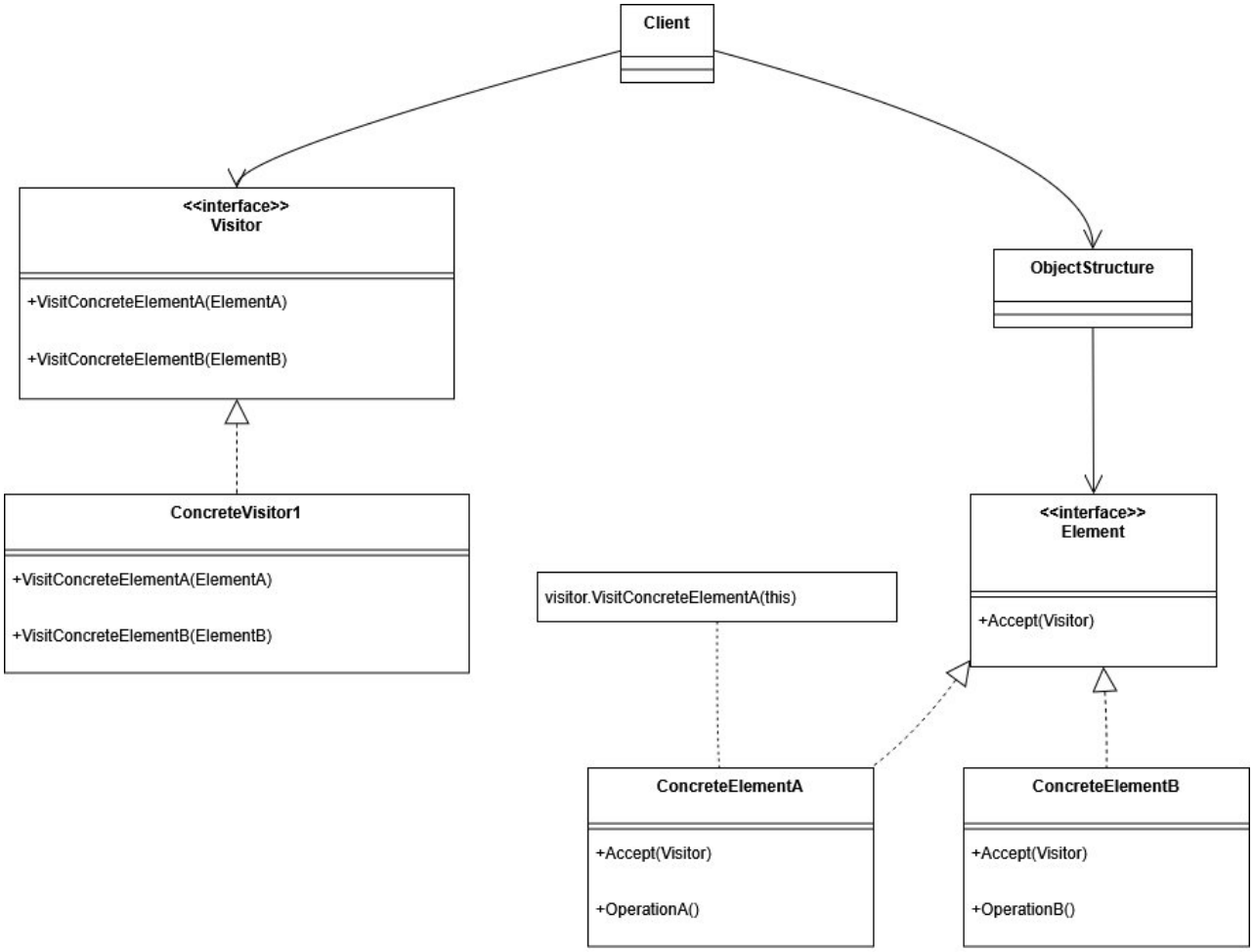
6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

- Flyweight (Легковаговик): Інтерфейс, через який пристосуванці можуть отримувати зовнішній стан і діяти відповідно до нього.
- ConcreteFlyweight: Реалізує інтерфейс і зберігає внутрішній стан (спільний для багатьох контекстів).
- UnsharedConcreteFlyweight: Не всі підкласи зобов'язані бути розділюваними. Цей клас зберігає весь стан у собі.
- FlyweightFactory: Створює та управляє пулом пристосуванців. Перевіряє наявність вже створеного об'єкта перед створенням нового.
- Client: Зберігає посилання на пристосуванців і обчислює або зберігає зовнішній стан.

7. Яке призначення шаблону «Інтерпретатор»?
Призначенням шаблону є визначення представлення граматики для заданої мови разом з інтерпретатором, який використовує це представлення для інтерпретації речень цієї мови. Використовується для часто повторюваних задач, які можна описати мовою (наприклад, регулярні вирази або математичні формули).

8. Яке призначення шаблону «Відвідувач»?
Шаблон «Відвідувач» призначений для відокремлення алгоритму від структури об'єктів, над якою він оперує. Це дозволяє додавати нові операції над класами без необхідності змінювати код цих класів, що спрощує розширення функціоналу .

9. Нарисуйте структуру шаблону «Відвідувач».



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

- Visitor (Відвідувач): Оголошує методи Visit для кожного конкретного класу елемента, який може бути відвіданий.
- ConcreteVisitor: Реалізує методи Visit, виконуючи конкретну логіку для кожного типу елемента (наприклад, у вашому проекті ChecksumVisitor, TestVisitor).
- Element: Оголошує метод Accept, який приймає об'єкт відвідувача як аргумент.
- ConcreteElement: Реалізує метод Accept, викликаючи відповідний метод відвідувача (наприклад, visitor.Visit(this)). Це техніка подвійної диспетчеризації.
- ObjectStructure: Містить колекцію елементів і дозволяє відвідувачу обійти їх (наприклад, список файлів в архіві).