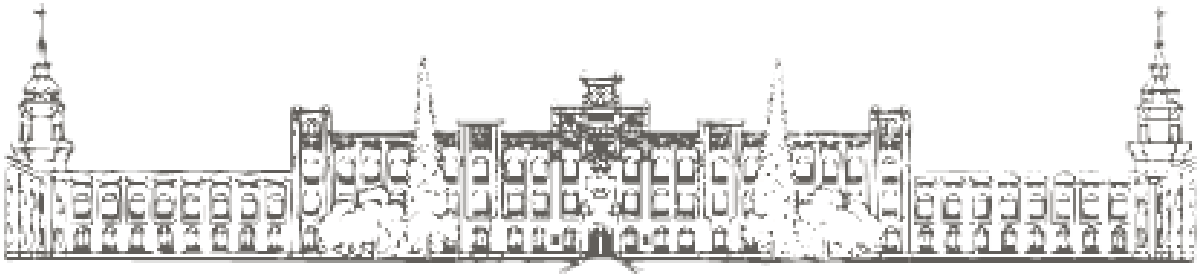


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Звіт лабораторної роботи №8

з курсу

«Технології розроблення програмного забезпечення»

Виконавець:
Нікітченко Наталя Олегівна
студентка групи ІА-33
залікова книжка № ІА-3318

«29» 11 2025 р.

Перевірив: **Мягкий М. Ю.**

Київ – 2025

8. ЛАБОРАТОРНА РОБОТА № 8

Тема: Патерни проектування.

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

Завдання:

Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

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

• 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.

Теоретичні відомості:

Шаблон «Flyweight»

Призначення: Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.

Дуже важливою є концепція «внутрішнього» і «зовнішнього» станів . Внутрішній стан відображає дані, характерні саме поділюваному об'єкту (наприклад, код букви); зовнішній стан несе інформацію про його застосування в додатку (наприклад, рядок і стовпчик). Внутрішній стан зберігається в самому поділюваному об'єкті, зовнішній – в об'єктах додатку (контексту використання поділюваного об'єкта).

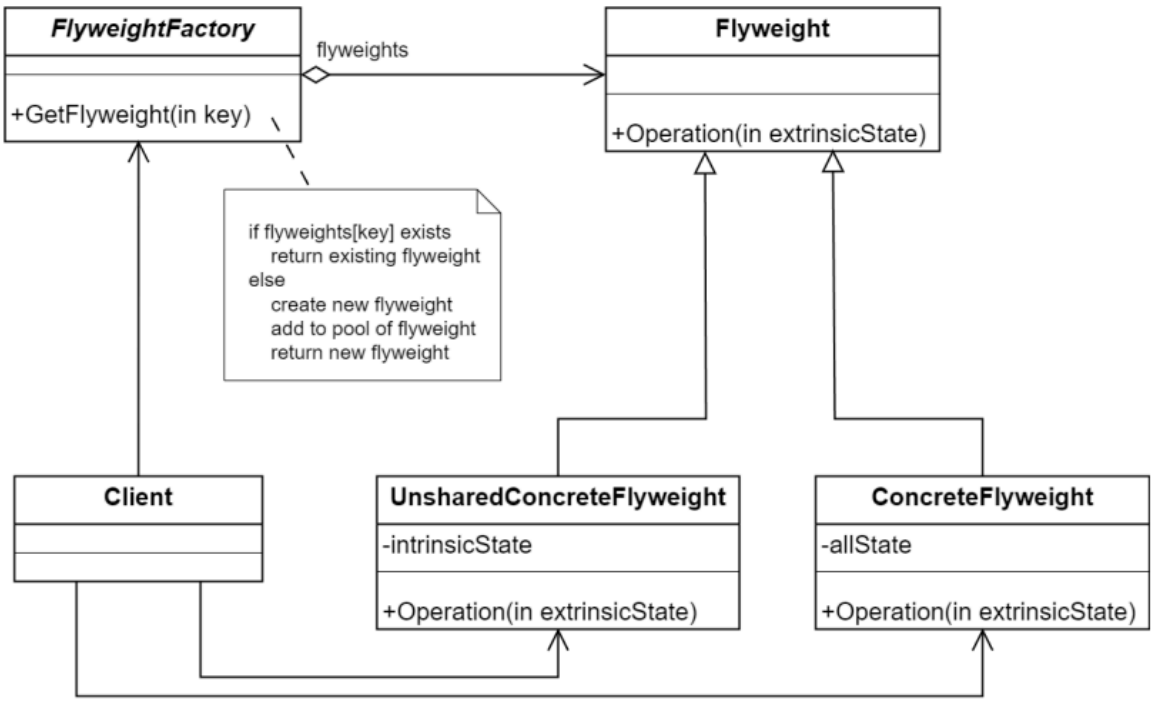


Рисунок 1 - Структура патерну Flyweight (Легковаговик)

Хорошим прикладом є наступне зображення на рисунку 3.

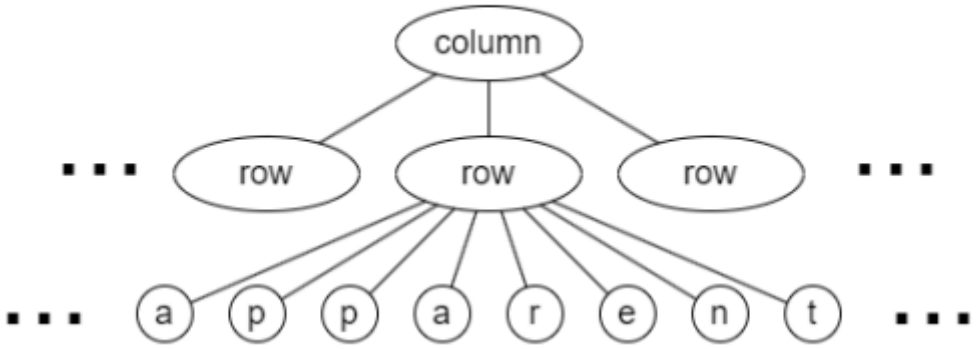


Рисунок 2 - Приклад ситуації де варто використовувати Flyweight

Здається, ніби для кожної літери існує окремий об'єкт. Насправді фізично об'єкт всього один, існує лише безліч посилань на нього. (рисунку 3)

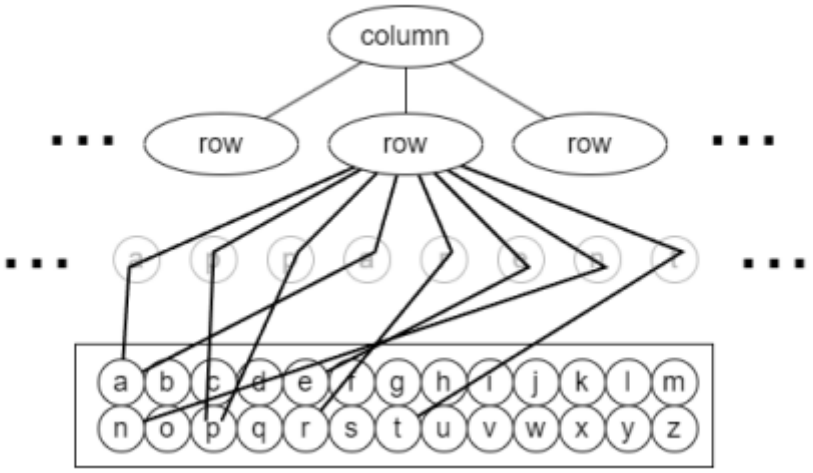


Рисунок 3 - Використання патерну Flyweight

Даний шаблон дуже добре застосовувати у випадках, коли використовується безліч однакових об'єктів (наприклад, графічних примітивів).

Переваги та недоліки:

+ Заощаджує оперативну пам’ять.

- Витрачає процесорний час на пошук/обчислення контексту.

- Ускладнює код програми внаслідок введення безлічі додаткових класів.

Хід роботи:

Тема: “Графічний редактор”

Реалізувати один з розглянутих шаблонів за обраною темою, а саме патерн «Flyweight ».

Розглянемо структуру цього прототипу(Рис.4):

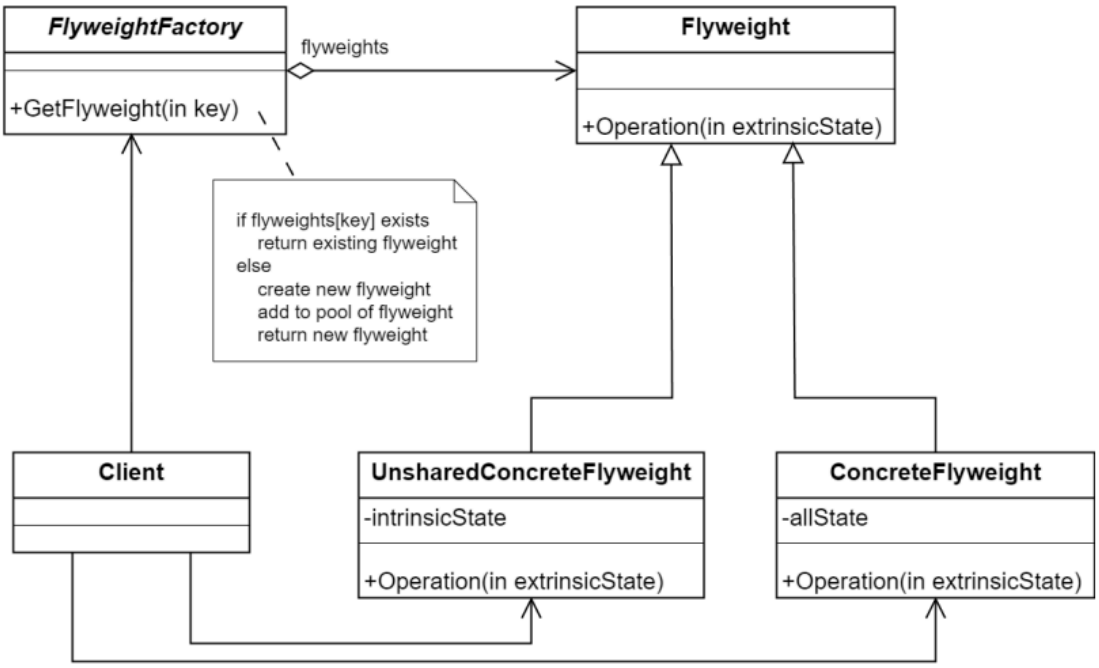


Рис.4 - Структура патерну «Легковаговик)»

Реалізація цього протипу відповідно даної теми(Рис.5):

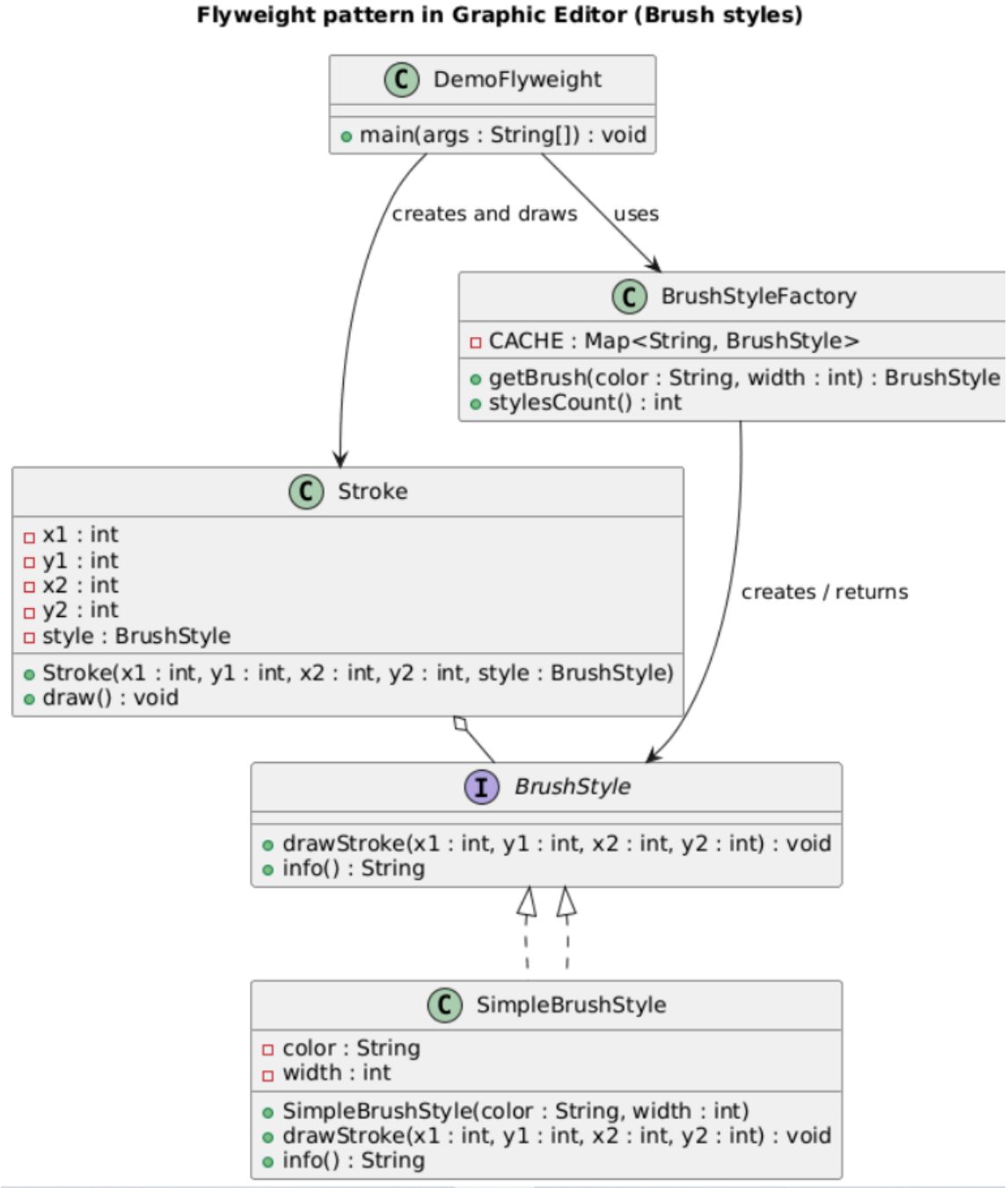


Рис.5- діаграму класів, яка представляє використання шаблону в реалізації системи

Діаграма Flyweight (Brush Styles) показує, що:

- BrushStyle — інтерфейс спільного стилю пензля (колір, товщина).

- SimpleBrushStyle — конкретний flyweight, який зберігає внутрішній стан, що повторюється (color, width).

- BrushStyleFactory — фабрика, яка містить кеш стилів. Якщо стиль вже існує — повертає його, якщо ні — створює і додає в кеш.

- Stroke — об’єкт лінії, що містить зовнішній стан (координати x1,y1,x2,y2) та посилання на спільний стиль BrushStyle.

- DemoFlyweight — клієнт, який створює багато Stroke, але завдяки кешу використовує лише кілька унікальних стилів.

```
package com.example.graphiceditor.model.flyweight;

public interface BrushStyle {  8 usages  1 implementation  new *

    void drawStroke(int x1, int y1, int x2, int y2);  1 usage  1 implementation

    String info();  2 usages  1 implementation  new *
}
```

Рис.6 - BrushStyle

BrushStyle – інтерфейс “легковаги”. Описує спільну поведінку стилю пензля: drawStroke(...) малює лінію, info() повертає опис стилю.

```java
package com.example.graphiceditor.model.flyweight;

import java.util.HashMap;
import java.util.Map;

public class BrushStyleFactory {  3 usages  new *

    private static final Map<String, BrushStyle> CACHE = new HashMap<>();  3 usages

    public static BrushStyle getBrush(String color, int width) {  1 usage  new *
        String key = color + "#" + width;
        BrushStyle style = CACHE.get(key);
        if (style == null) {
            style = new SimpleBrushStyle(color, width);
            CACHE.put(key, style);
            System.out.println("[Factory] create new " + style.info());
        } else {
            System.out.println("[Factory] reuse " + style.info());
        }
        return style;
    }

    public static int stylesCount() {  1 usage  new *
        return CACHE.size();
    }
}
```

Рис.7- BrushStyleFactory

BrushStyleFactory – фабрика та кеш.Тримає Map<String, BrushStyle> CACHE.

getBrush(color, width):

- робить ключ color#width,
- якщо такого стилю ще нема – створює SimpleBrushStyle і кладе в кеш;
- якщо вже є – повертає існуючий об’єкт. Так кількість стилів обмежена, а не тисячі дубльованих.

```java
package com.example.graphiceditor.model.flyweight;

public class SimpleBrushStyle implements BrushStyle {  1 usage  new *

    private final String color;  3 usages
    private final int width;  3 usages

    public SimpleBrushStyle(String color, int width) {  1 usage  new *
        this.color = color;
        this.width = width;
    }

    @Override  1 usage  new *
    public void drawStroke(int x1, int y1, int x2, int y2) {
        System.out.printf(
                "[BrushStyle %s] draw line (%d,%d) -> (%d,%d), width=%d%n",
                color, x1, y1, x2, y2, width
        );
    }

    @Override  2 usages  new *
    public String info() {
        return "BrushStyle{color=" + color + ", width=" + width + "}";
    }
}
```

Рис.8 - SimpleBrushStyle

SimpleBrushStyle – конкретний flyweight. Зберігає колір і товщину лінії (color, width) і в drawStroke(...) реально малює (тут просто пише в консоль). Один такий об’єкт можна використовувати для багатьох ліній.

```java
package com.example.graphiceditor.model.flyweight;

public class Stroke {  4 usages  new *

    private final int x1, y1, x2, y2;  2 usages
    private final BrushStyle style;  // Flyweight  2 usages

    public Stroke(int x1, int y1, int x2, int y2, BrushStyle style) {  1 usage  new
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
        this.style = style;
    }

    public void draw() {  1 usage  new *
        style.drawStroke(x1, y1, x2, y2);
    }
}
```

Рис.9 - Stroke

Stroke – контекст навколо flyweight. Містить змінні дані конкретного штриха: координати x1,y1,x2,y2 та посилання на спільний BrushStyle style. У draw() просто делегує малювання в style.drawStroke(...).

Висновки:

На ці лабораторній роботі було розглянуто класичну реалізацію Flyweight, адаптовану під тему графічного редактора:

- Є чіткий поділ на Flyweight (BrushStyle), ConcreteFlyweight (SimpleBrushStyle), контекст (Stroke) і фабрику (BrushStyleFactory).
- Внутрішній стан стилю (колір, товщина) спільно використовується багатьма об’єктами, а зовнішній стан (координати) зберігається окремо.

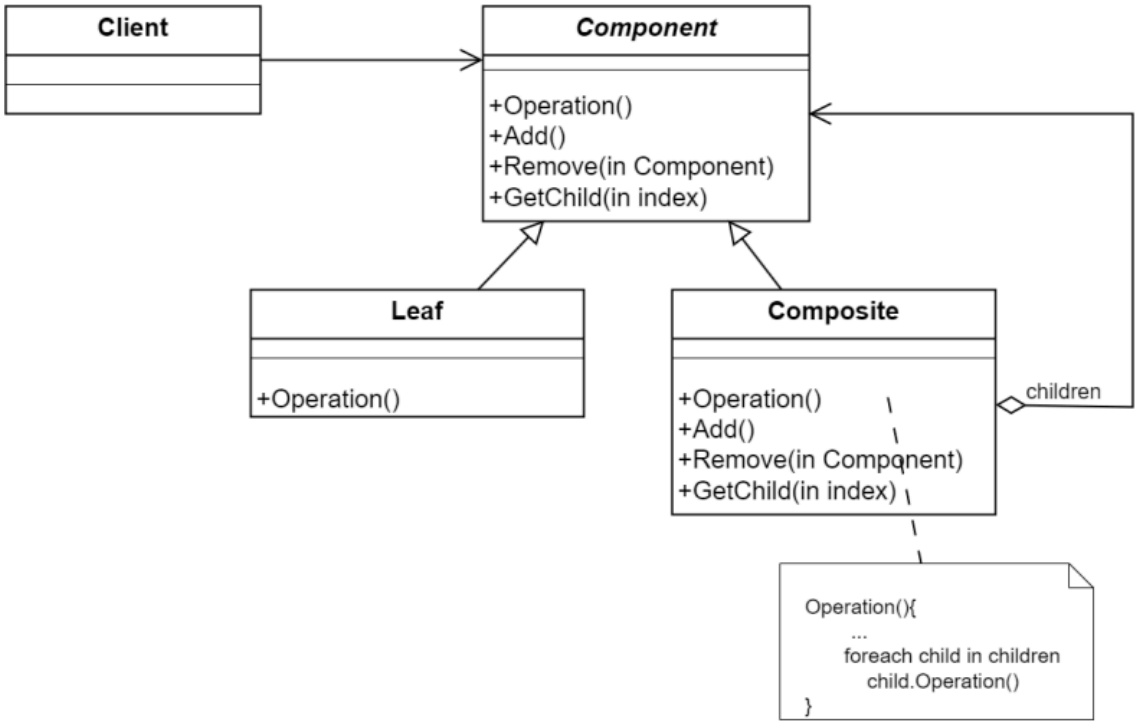
- Фабрика гарантує, що для кожної комбінації параметрів створюється рівно один об’єкт стилю, а інші клієнти його повторно використовують.
- Така архітектура робить редактор легким по пам’яті, зручним для розширення та добре підходить для демонстрації патерну Flyweight.

Контрольні питання:

1. Яке призначення шаблону «Композит»?

Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю.

2. Нарисуйте структуру шаблону «Композит».



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

Класи які входять в шаблон «Композит»:

- component(абстрактний клас або інтерфейс)

Визначає спільний інтерфейс для всіх об'єктів у композиції. Містить методи: Operation(), Add(Component), Remove(Component), GetChild(index)

- leaf(конкретний клас)

Представляє кінцеві об'єкти, які не мають дітей. Реалізує лише Operation().

- composite(конкретний клас)

Представляє складені об'єкти, які можуть містити інші Component (і Leaf, і Composite). Реалізує всі методи інтерфейсу та зберігає колекцію дітей.

- client(клієнтський код)

Працює з об'єктами через інтерфейс Component, не розрізняючи Leaf чи Composite.

Взаємодія між класами:

Client викликає методи через інтерфейс Component, не знаючи, чи це Leaf, чи Composite.

Composite містить список об'єктів типу Component — це можуть бути як Leaf, так і інші Composite.

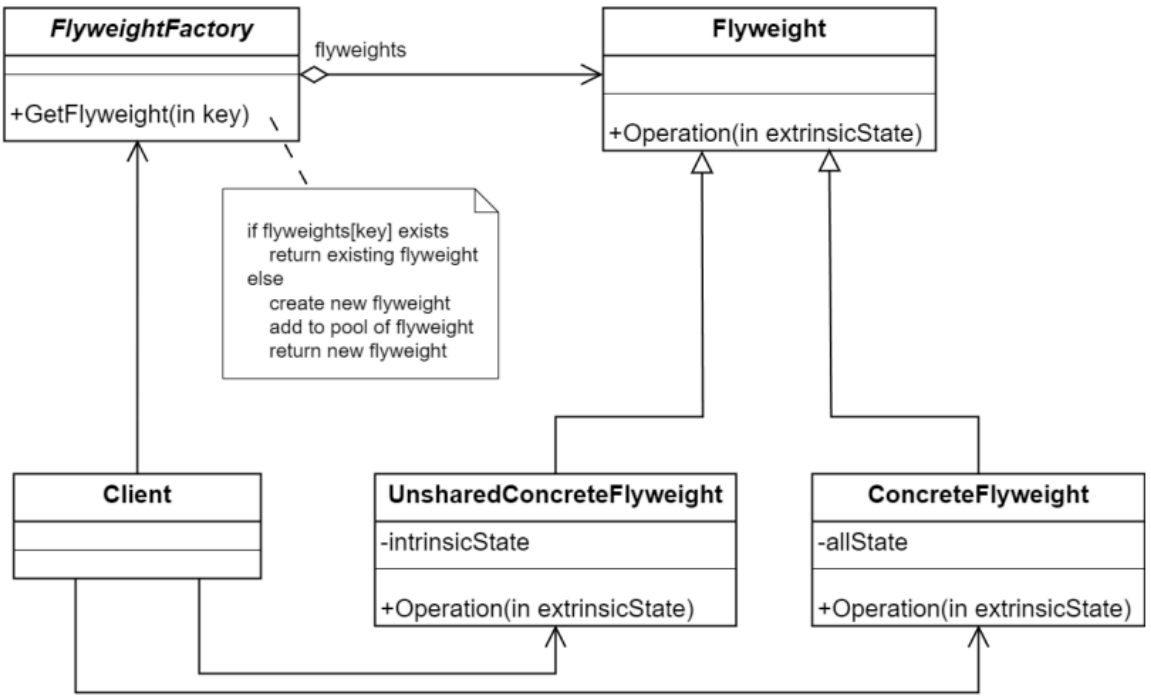
Коли **Client** викликає Operation() на Composite, той виконує операцію для кожного свого дочірнього елемента:

Operation() { foreach (Component child : children) child.Operation(); } **Leaf** реалізує Operation() без підтримки дітей — методи Add, Remove, GetChild можуть бути порожніми або кидати винятки.

4. Яке призначення шаблону «Легковаговик»?

Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Класи які входять в шаблон «Легковаговик»:

- Flyweight(Інтерфейс)

Визначає метод Operation(extrinsicState) для об'єктів, які можуть бути спільно використані.

- ConcreteFlyweight(Реалізація Flyweight)

Зберігає внутрішній стан (intrinsicState), який спільний для багатьох об'єктів. Реалізує Operation(extrinsicState).

- UnsharedConcreteFlyweight(Альтернативна реалізація)

Не ділиться з іншими. Має власний внутрішній стан, використовується для унікальних випадків.

- FlyweightFactory(Фабрика)
Керує пулом Flyweight-об'єктів. Повертає існуючий об'єкт або створює новий, якщо його ще немає.

- Client(Клієнтський код)

Використовує Flyweight-об'єкти через фабрику, передаючи зовнішній стан (extrinsicState) при виклику Operation().

Взаємодія між класами:

Client звертається до FlyweightFactory, щоб отримати Flyweight-об'єкт за ключем.

FlyweightFactory перевіряє, чи існує об'єкт у полі.

Flyweight(через ConcreteFlyweight) виконує операцію, використовуючи зовнішній стан, який передає клієнт.

UnsharedConcreteFlyweight використовується, коли об'єкт не може бути спільним — наприклад, має унікальні дані.

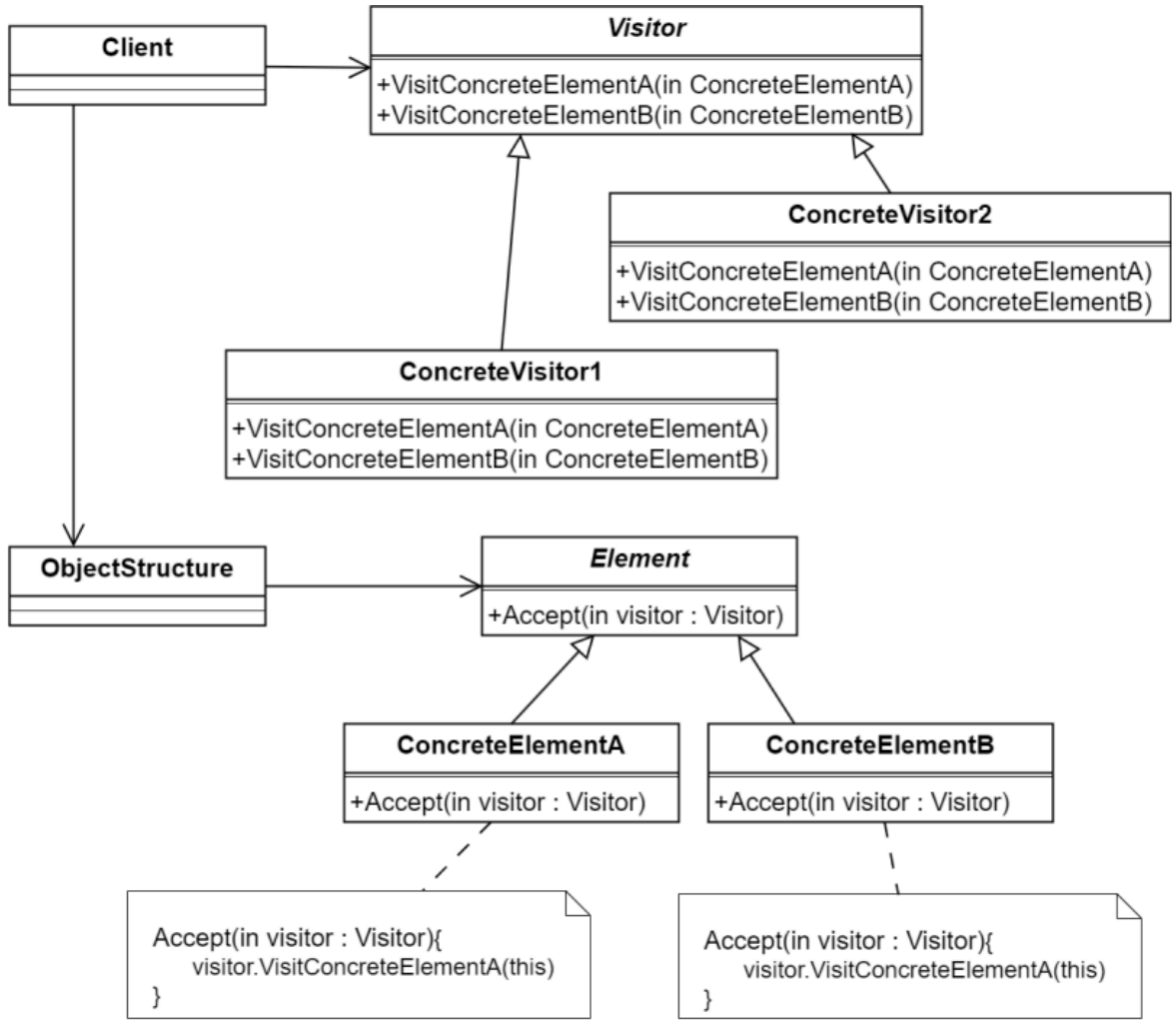
7. Яке призначення шаблону «Інтерпретатор»?
Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової) .

Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту.

8. Яке призначення шаблону «Відвідувач»?

Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію. Шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Класи які входять в шаблон «Відвідувач»:

- Visitor(Інтерфейс)

Визначає методи для відвідування кожного типу елементів: VisitConcreteElementA(), VisitConcreteElementB().

- ConcreteVisitor1/2(Реалізація Visitor)
Реалізують специфічні дії для кожного типу елементів.

- Element(Абстрактний клас або інтерфейс)

Має метод Accept(Visitor) — дозволяє відвідувачу взаємодіяти з елементом.

- ConcreteElementA/B(Реалізація Element)

Реалізують Accept(Visitor) так, щоб викликати відповідний метод VisitConcreteElementX(this).

- ObjectStructure(Контейнер)
Містить колекцію елементів і дозволяє відвідувачу пройтись по них.

- Client(Клієнтський код)

Ініціює процес відвідування, передаючи відвідувача до ObjectStructure.

Взаємодія між класами:

Client створює Visitor і передає його до ObjectStructure.

ObjectStructure проходить по всіх Element і викликає Accept(visitor) для кожного.

Кожен **ConcreteElement** викликає відповідний метод VisitConcreteElementX(this) у Visitor.

ConcreteVisitor реалізує логіку, яку потрібно виконати для кожного типу елемента.