

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 розробки програмного забезпечення»
Патерни програмування

Виконав:
студент групи ІА-34
Швець Роман Вадимович

Перевірив:
асистент кафедри ІСТ
Мякий Михайло Юрійович

Тема: Патерни програмування

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

4. Графічний редактор (proxy, prototype, decorator, bridge, flyweight, SOA)

Графічний редактор повинен вміти створювати / редагувати растрові (або векторні на розсуд студента) зображення в 2-3 основних популярних форматах (bmp, png, jpg), мати панель інструментів для створення графічних примітивів, вибору кольорів, нанесення тексту, додавання найпростіших візуальних ефектів (ч/б растр, інфрачервоний растр, 2-3 на вибір учня), роботи з шарами.

Теоретичні відомості

Шаблони проєктування (патерни) – це загальні рішення типових проблем, що виникають у процесі розроблення програмного забезпечення. Вони не є готовим кодом, а радше описують структуру взаємодії між класами та об'єктами, допомагаючи створювати гнучкі, масштабовані й легко підтримувані системи. Використання шаблонів дозволяє стандартизувати архітектурні рішення, полегшити командну розробку та уникнути повторення вже відомих помилок у проєктуванні.

Шаблон Composite (Компонувальник) – це структурний патерн, який дозволяє групувати об'єкти у деревоподібні структури для відображення ієрархій «частина-ціле». Основна ідея полягає в тому, щоб клієнтський код міг працювати з окремими об'єктами (листочками дерева) та групами об'єктів (вузлами) однаково, через спільний інтерфейс. Це значно спрощує архітектуру застосунків, що працюють зі складними вкладеними структурами, такими як файлові системи або графічні редактори, де група фігур може оброблятися так само як одна фігура (переміщуватися, масштабуватися тощо).

Шаблон Flyweight (Пристосуванець) є структурним патерном, призначеним для ефективної підтримки великої кількості дрібних об'єктів шляхом економії

оперативної пам'яті. Замість того щоб зберігати однакові дані в кожному екземплярі об'єкта, цей патерн пропонує розділяти спільний стан (інтринсивний, внутрішній) між багатьма об'єктами, а унікальний стан (екстринсивний, зовнішній) передавати об'єкту ззовні під час виклику методів. Цей підхід є критично важливим для систем, де кількість об'єктів вимірюється тисячами або мільйонами, наприклад, при рендерингу символів у текстовому редакторі або частинок у графічних системах.

Шаблон Interpreter (Тлумач) – це поведінковий патерн, який використовується для визначення граматики простої мови та інтерпретації речень цієї мови. Він будує абстрактне синтаксичне дерево, де кожен вузол представляє правило граматики. Патерн найчастіше застосовується у спеціалізованих задачах, таких як аналіз математичних виразів, розбір SQL-запитів або регулярних виразів, проте його використання може бути неефективним для складних граматик через велику кількість класів, що створюються для кожного правила.

Шаблон Visitor (Відвідувач) належить до поведінкових патернів і дозволяє додавати нові операції до ієрархії класів без зміни коду самих класів. Це досягається шляхом відокремлення алгоритму від структури об'єктів, над якою він оперує. Використовується техніка подвійної диспетчеризації (double dispatch), коли об'єкт структури приймає «відвідувача» і викликає у нього відповідний метод. Патерн є корисним, коли необхідно виконувати різноманітні операції над об'єктами складної структури (наприклад, експорт у XML, генерація звітів, статистичний аналіз) і при цьому небажано "забруднювати" класи цих об'єктів сторонньою логікою.

Застосування цих шаблонів у проєктуванні програмних систем забезпечує розширюваність, спрощує тестування, дозволяє дотримуватись принципів SOLID та робить архітектуру більш зрозумілою і стійкою до змін.

Хід роботи

Для оптимізації використання оперативної пам'яті при роботі з великою кількістю однотипних графічних елементів (наприклад, точок пензля, частинок текстур або масиву геометричних фігур) використано шаблон проєктування «Пристосуванець» (Flyweight). Його основна ідея полягає у розділенні стану об'єкта на внутрішній

(інтринсивний), який є незмінним і спільним для багатьох об'єктів, та зовнішній (екстринсивний), який є унікальним і передається об'єкту під час виклику методів. У проєкті це реалізовано через інтерфейс `IShapeFlyweight`, який визначає метод малювання, та клас `ConcreteShapeType`, що зберігає «важкі» дані (колір, тип фігури, текстуру). Клас-фабрика `ShapeFactory` керує пулом цих об'єктів, гарантуючи, що для кожного типу фігури в пам'яті існує лише один екземпляр. Контекстний клас `Dot` зберігає лише унікальні координати та посилання на спільний об'єкт-пристосуванець.

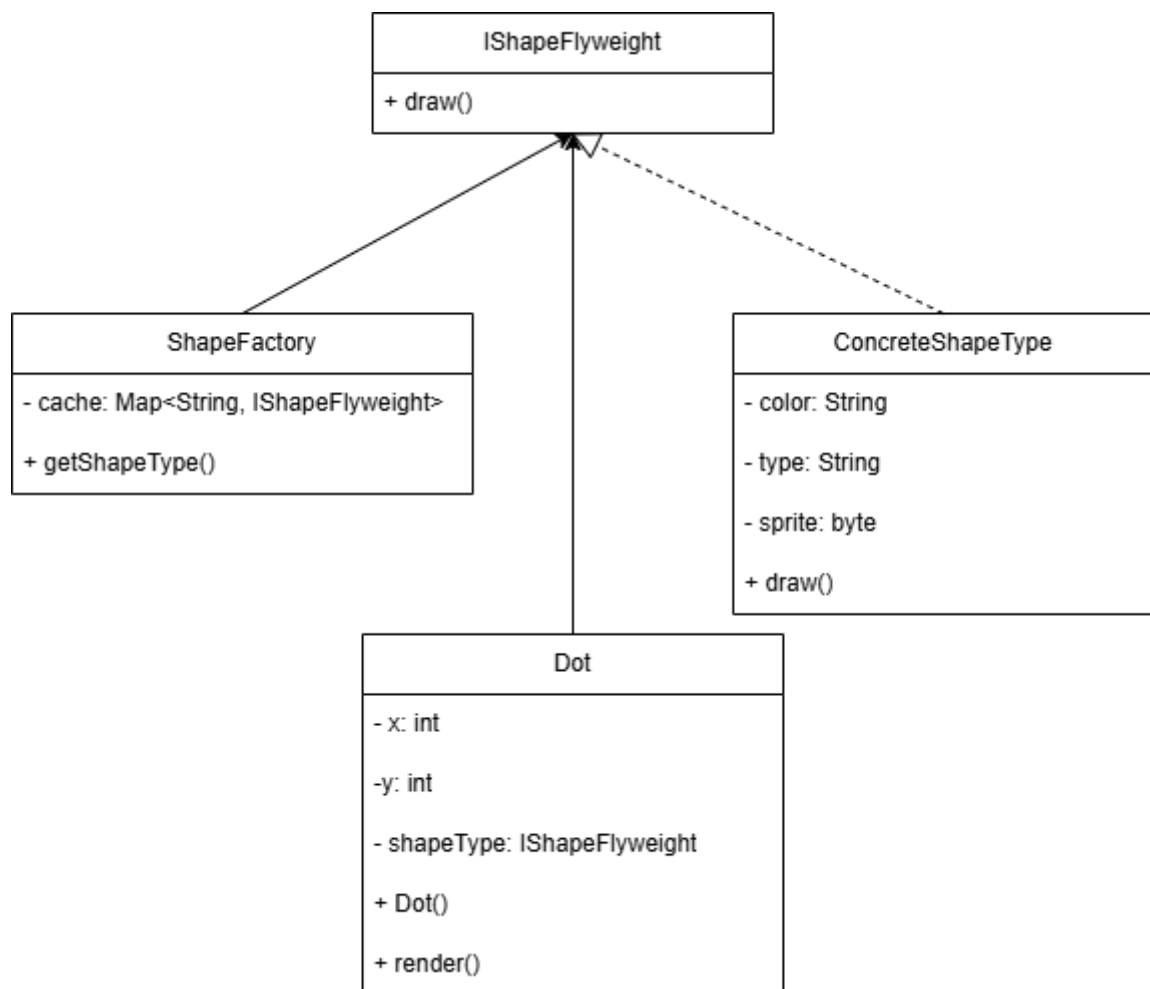


Рисунок 8.1 – Діаграма класів для Flyweight

На поданій діаграмі класів відображено структурну організацію шаблону «Пристосуванець» у системі графічного редактора. Ключовим елементом є `ShapeFactory`, яка через агрегацію володіє колекцією об'єктів `IShapeFlyweight` і забезпечує їх повторне використання. Клас `Dot` (клієнтський контекст) пов'язаний із `IShapeFlyweight` асоціацією: він не дублює дані про колір чи форму, а лише делегує процес відмалювки пристосуванцю, передаючи свої конкретні координати

як аргументи методу. Такий архітектурний підхід робить систему високопродуктивною та масштабованою – дозволяє створювати тисячі візуальних об'єктів із мінімальними витратами пам'яті, оскільки фізично в пам'яті зберігаються лише одиничні екземпляри спільних даних.

Вихідні коди класів системи

Лістинг 1 – Інтерфейс IShapeFlyweight

```
package com.mycompany.code;

public interface IShapeFlyweight {
    void draw(int x, int y);
}
```

Лістинг 2 – клас ConcreteShapeType

```
package com.mycompany.code;

public class ConcreteShapeType implements IShapeFlyweight {
    private String type;
    private String color;

    public ConcreteShapeType(String type, String color) {
        this.type = type;
        this.color = color;
    }

    @Override
    public void draw(int x, int y) {
        System.out.println("Відображення [" + type + ", колір: " + color + "] у
точці (" + x + ";" + y + ")");
    }
}
```

Лістинг 3 – клас ShapeFactory

```
package com.mycompany.code;

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

public class ShapeFactory {
    private static final Map<String, IShapeFlyweight> shapeTypes = new
HashMap<>();

    public static IShapeFlyweight getShapeType(String type, String color) {
        String key = type + "_" + color;
        IShapeFlyweight shape = shapeTypes.get(key);

        if (shape == null) {
            shape = new ConcreteShapeType(type, color);
            shapeTypes.put(key, shape);
            System.out.println("--- Створено НОВИЙ спільний об'єкт у пам'яті: " +
key + " ---");
        }
        return shape;
    }
}
```

Лістинг 4 – клас Dot

```
package com.mycompany.code;

public class Dot {
    private int x;
    private int y;
    private IShapeFlyweight shapeType;

    public Dot(int x, int y, IShapeFlyweight shapeType) {
        this.x = x;
        this.y = y;
        this.shapeType = shapeType;
    }

    public void render() {
        shapeType.draw(x, y);
    }
}
```

Лістинг 5 – Демонстрація використання

```
package com.mycompany.code;

import java.util.ArrayList;
import java.util.List;

public class EditorUI {

    public static void main(String[] args) {
        List<Dot> dots = new ArrayList<>();

        IShapeFlyweight redCircle = ShapeFactory.getShapeType("Circle", "Red");
        dots.add(new Dot(10, 10, redCircle));
        dots.add(new Dot(20, 30, redCircle));
        dots.add(new Dot(40, 50, redCircle));

        IShapeFlyweight blueStar = ShapeFactory.getShapeType("Star", "Blue");
        dots.add(new Dot(100, 100, blueStar));
        dots.add(new Dot(120, 110, blueStar));

        IShapeFlyweight moreRedCircle = ShapeFactory.getShapeType("Circle",
"Red");
        dots.add(new Dot(99, 99, moreRedCircle));

        System.out.println("\n--- Рендеринг сцени (6 об'єктів, але лише 2 типи у
пам'яті) ---");
        for (Dot dot : dots) {
            dot.render();
        }
    }
}
```

У поданих лістингах реалізовано застосування шаблону проєктування «Пристосуванець» (Flyweight) у системі графічного редактора. Інтерфейс `IShapeFlyweight` визначає контракт для легковагових об'єктів, декларуючи метод `draw`, який приймає зовнішній стан (координати) як аргументи. Клас

ConcreteShapeType є конкретною реалізацією цього інтерфейсу і відповідає за зберігання внутрішнього, незмінного стану (тип фігури, колір, текстура), який є спільним для багатьох об'єктів. Клас ShapeFactory виступає ключовим елементом керування пам'яттю: він перевіряє наявність вже створеного типу фігури у внутрішньому кеші і повертає його, або створює новий, якщо такий ще не існує. Клас Dot (контекст) зберігає лише унікальні дані конкретної точки (координати x, y) та посилання на спільний об'єкт IShapeFlyweight. Така реалізація дозволяє створювати велику кількість точок на екрані, фактично використовуючи в пам'яті лише декілька об'єктів-пристосувань, що значно оптимізує споживання ресурсів системи.

Висновок:

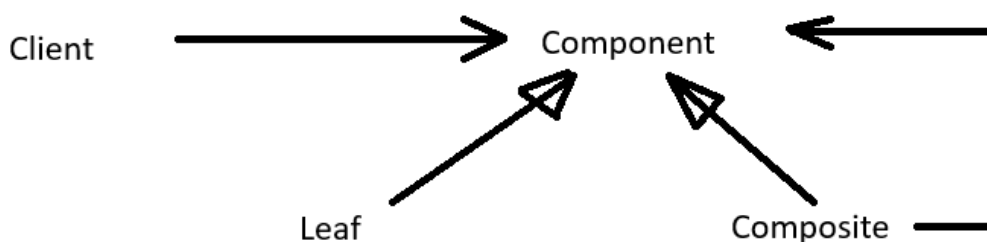
У ході виконання лабораторної роботи було реалізовано шаблон проєктування Flyweight (Пристосуванець) у контексті системи графічного редактора. Цей шаблон дозволив вирішити проблему надмірного споживання оперативної пам'яті при роботі з великою кількістю однотипних графічних елементів. У проєкті створено архітектуру, що розділяє стан об'єктів на внутрішній (спільний) та зовнішній (унікальний). Клас ShapeFactory забезпечує ефективне керування пулом спільних ресурсів, а клієнтські об'єкти Dot делегують логіку відображення пристосуванням. Такий підхід спростив масштабування системи, дозволивши оперувати тисячами об'єктів без суттєвого впливу на продуктивність.

Відповіді на контрольні запитання:

1. Яке призначення шаблону «Композит»?

Дозволяє згрупувати об'єкти у деревоподібні структури для відображення ієрархії «частина-ціле» та дає клієнтам можливість працювати з окремими об'єктами й групами об'єктів однаково через спільний інтерфейс.

2. Нарисуйте структуру шаблону «Композит».



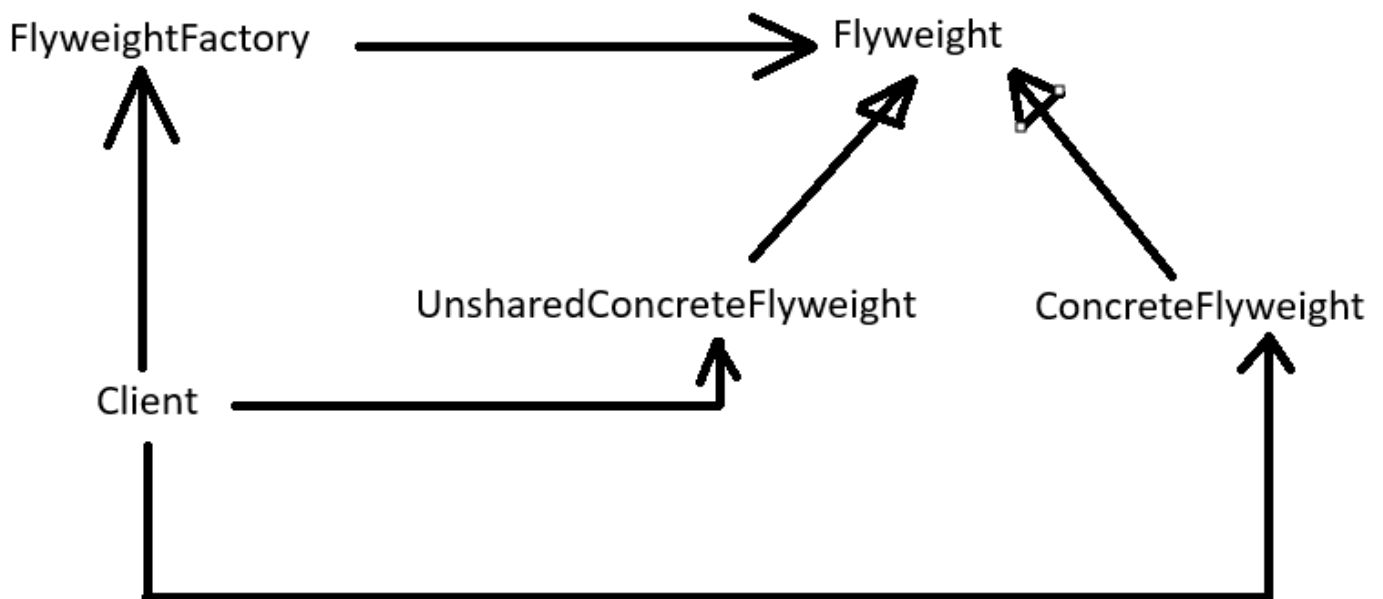
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

Класи: Component (спільний інтерфейс), Leaf (простий елемент, що не має підлеглих), Composite (контейнер, що містить дочірні елементи), Client. Composite зберігає колекцію дочірніх компонентів (як Leaf, так і інших Composite) і делегує їм виконання запитів, а Client взаємодіє з усіма елементами через інтерфейс Component.

4. Яке призначення шаблону «Легковаговик»?

Призначений для оптимізації роботи з великою кількістю дрібних об'єктів шляхом розділення їхнього стану на внутрішній (спільний для багатьох об'єктів) та зовнішній (унікальний), що дозволяє суттєво економити оперативну пам'ять.

5. Нарисуйте структуру шаблону «Легковаговик».



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

Класи: Flyweight (інтерфейс), ConcreteFlyweight (зберігає внутрішній стан), FlyweightFactory (створює та керує пулом об'єктів), Client (зберігає зовнішній стан). Клієнт запитує об'єкт у Фабрики (яка повертає існуючий або створює новий) і викликає методи Flyweight, передаючи унікальний (зовнішній) стан як аргументи.

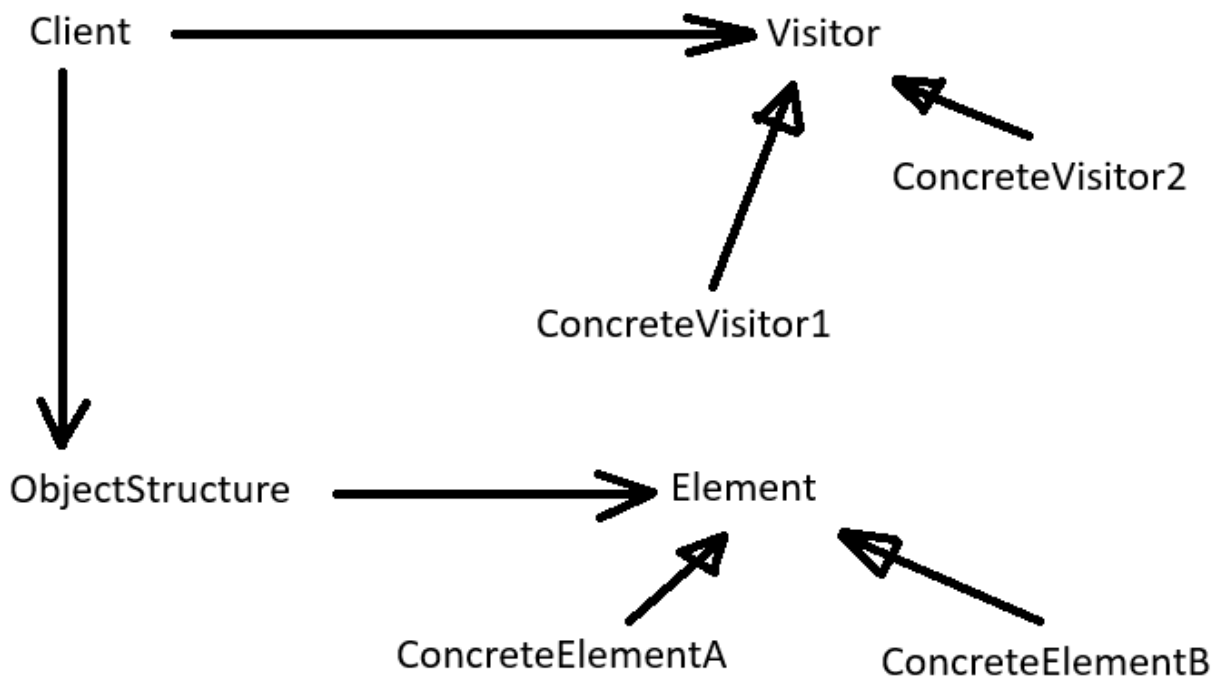
7. Яке призначення шаблону «Інтерпретатор»?

Використовується для визначення граматики простої мови та створення інтерпретатора, який розбирає та виконує речення цієї мови (наприклад, математичні вирази або пошукові запити).

8. Яке призначення шаблону «Відвідувач»?

Дозволяє додавати нові операції до ієрархії класів без зміни вихідного коду цих класів, відокремлюючи алгоритм від структури об'єктів, над якою він виконується.

9. Нарисуйте структуру шаблону «Відвідувач».



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

Класи: **Visitor** (інтерфейс з методами `visit` для кожного типу елемента), **ConcreteVisitor** (реалізація алгоритму), **Element** (інтерфейс з методом `accept`), **ConcreteElement**. Об'єкт **Element** приймає об'єкт **Visitor** через метод `accept()`, який, у свою чергу, викликає відповідний метод `visit()` у відвідувача, передаючи посилання на себе (техніка подвійної диспетчеризації).