Міністерство освіти і науки України
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 розроблення програмного забезпечення**»

Патерни проектування.

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

Вихідний код: https://github.com/Dimon4ick68/MindMapLabs

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.

Варіант:

20. **Mind-mapping software** (strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

Хід роботи

У рамках виконання лабораторної роботи було розроблено модуль для роботи з ієрархічною структурою ментальної карти. Для цього використано структурний патерн **Composite (Компонувальник)**, який дозволяє згрупувати об'єкти у деревоподібну структуру і працювати з нею так само як з одиничним об'єктом.

Було обрано "безпечний" варіант реалізації (Safe Composite), де методи управління дітьми (add, remove) визначені лише в класі-контейнері, а не в загальному інтерфейсі.

Рисунок 1 Структура проекту

Реалізовано такі компоненти патерну:

1. **MapItem (Component — Компонент)**: Спільний інтерфейс для всіх елементів структури (як простих вузлів, так і груп). Він оголошує операції, які є спільними для всіх:

- print(indent): вивести назву елемента з відповідним відступом (візуалізація ієрархії).
- getSize(): отримати "вагу" елемента (в даному випадку — кількість символів у тексті вузла).

2. **MapLeaf (Leaf — Листок)**: Клас, що представляє кінцевий елемент дерева (вузол без підвузлів). Він реалізує методи інтерфейсу MapItem, виконуючи реальну роботу (виводить свій текст, повертає свою довжину).
3. **MapGroup (Composite — Композит)**: Клас, що представляє складений елемент (вузол, який має дітей).
   - Він зберігає список дочірніх компонентів типу MapItem.
   - Метод getSize() працює рекурсивно: він підсумовує результати виклику getSize() для всіх своїх дітей.
   - Метод print() спочатку виводить інформацію про себе (групу), а потім делегує виклик кожному дочірньому елементу.
4. **EditorForm (Client — Клієнт)**: Клас редактора, який ініціює роботу з композитом. Він містить метод-конвертер, який перетворює реальну модель даних (Node) у структуру композита (MapItem) і запускає обчислення статистики або вивід структури.

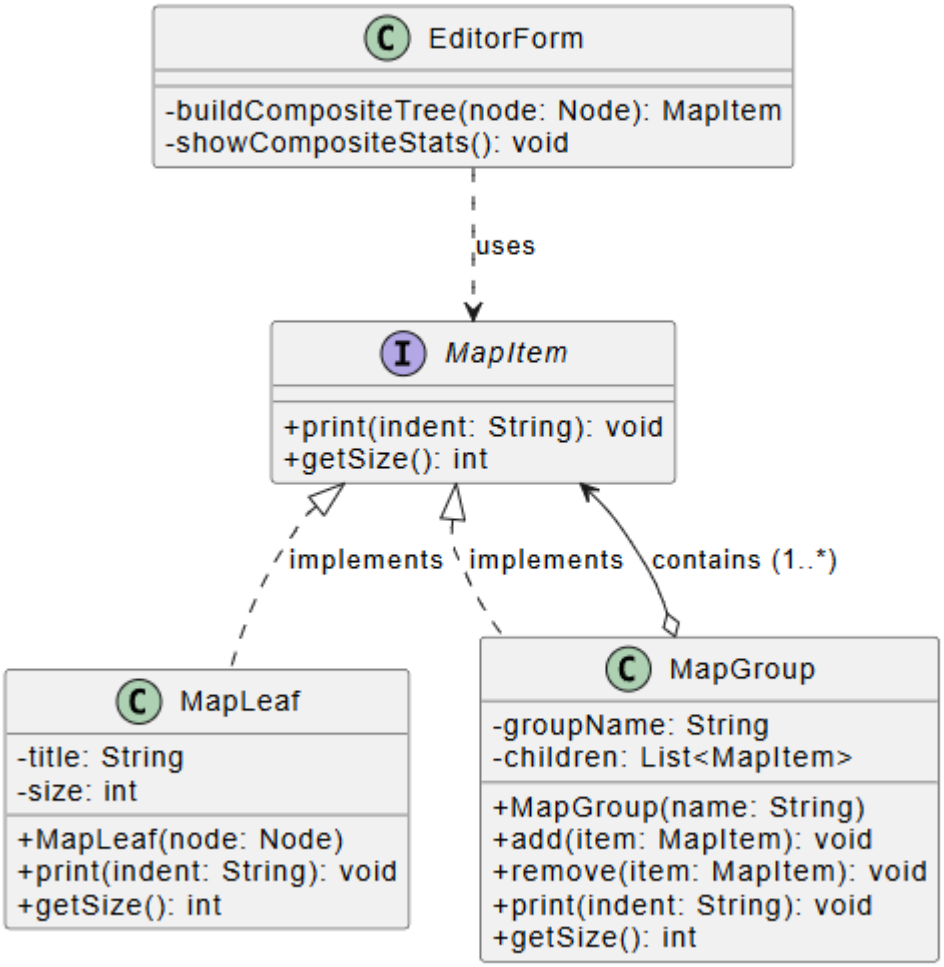


Рисунок 2 Діаграма класів

Проблеми, які вирішує патерн

- **Складність обробки рекурсивних структур:** Ментальна карта за своєю суттю є деревом. Без використання патерну Composite клієнтському коду довелося б постійно перевіряти тип об'єкта (чи це група, чи окремий вузол) і писати складні цикли для обходу. Патерн приховує цю складність за поліморфізмом.
- **Необхідність уніфікованої обробки:** Часто виникає потреба виконати дію над "виділеним елементом", не знаючи заздалегідь, чи це один вузол, чи ціла гілка. Composite дозволяє ігнорувати цю різницю.

Переваги використання

- **Спрощення клієнтського коду:** Клієнт працює лише з інтерфейсом MapItem. Йому неважливо, чи працює він з простим елементом, чи зі складним деревом об'єктів.
- **Легкість розширення:** Можна легко додати нові типи компонентів (наприклад, MapImage або MapLink), просто реалізувавши інтерфейс MapItem. Це не зламає існуючий код підрахунку статистики чи виводу.
- **Відповідність принципу відкритості/закритості (Open/Closed Principle):** Можна додавати нові класи елементів без зміни клієнтського коду, який працює з деревом.

Висновок

У ході лабораторної роботи було успішно реалізовано патерн **Composite** для системи "MindApp". Це дозволило створити зручний механізм для роботи з ієрархічною структурою карти, уніфікувавши операції над окремими вузлами та їх групами. Реалізований підхід значно спрощує додавання нових функцій аналізу та обробки структури карти в майбутньому..

Код

MapItem.java

```java
package com.mindapp.client.patterns;

// [Component]
public interface MapItem {
    void print(String indent); // Операція: вивести структуру
    int getSize();             // Операція: отримати "вагу" (наприклад, к-сть символів або файлів)
}

```

MapGroup.java

```java
package com.mindapp.client.patterns;

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

// [Composite] - Контейнер
public class MapGroup implements MapItem {
    private final String groupName;
    private final List<MapItem> children = new ArrayList<>();

    public MapGroup(String name) {
        this.groupName = name;
    }

    public void add(MapItem item) {
        children.add(item);
    }

    public void remove(MapItem item) {
        children.remove(item);
    }

    @Override
    public void print(String indent) {
        System.out.println(indent + "📂 ГРУППА: " + groupName);
        for (MapItem child : children) {
            child.print(indent + "    ");
        }
    }

    @Override
    public int getSize() {
        int totalSize = 0;
        for (MapItem child : children) {
            totalSize += child.getSize();
        }
        return totalSize;
    }
}

```

MapLeaf.java

client > demo > src > main > java > com > mindapp > client > patterns > J MapLeaf.java > ...

```java
package com.mindapp.client.patterns;

import com.mindapp.client.models.Node;

// [Leaf] - Кінцевий елемент
public class MapLeaf implements MapItem {
    private final String title;
    private final int size;

    // Ми приймаємо Node, але беремо з нього тільки дані
    public MapLeaf(Node node) {
        this.title = node.getText();
        this.size = node.getText().length(); // Наприклад, вага = довжина тексту
    }

    @Override
    public void print(String indent) {
        System.out.println(indent + "📄 " + title + " (" + size + " байт)");
    }

    @Override
    public int getSize() {
        return size;
    }
}

```

EditorForm.java

```java
// Метод для демонстрації Composite (альтернативний варіант)
private void showCompositeStats() {
    if (map.getRootNode() == null) return;

    // 1. Будуємо дерево Композита з нашої моделі Node
    MapItem compositeRoot = buildCompositeTree(map.getRootNode());

    // 2. Використовуємо його
    System.out.println(x: "\n--- COMPOSITE PATTERN OUTPUT ---");
    compositeRoot.print(indent: "");
    System.out.println(x: "--------------------------------");

    int totalSize = compositeRoot.getSize();
    showAlert("Статистика (Composite):\nЗагальний об'єм тексту: " + totalSize + " символів.");
}

// Рекурсивний метод-конвертер: Node -> MapItem
private MapItem buildCompositeTree(Node node) {
    // Якщо у вузла немає дітей - це Листок
    if (node.getChildren().isEmpty()) {
        return new MapLeaf(node);
    } else {
        // Якщо є діти - це Група
        MapGroup group = new MapGroup(node.getText());
        for (Node child : node.getChildren()) {
            // Рекурсивно додаємо дітей
            group.add(buildCompositeTree(child));
        }
        return group;
    }
}
```

```java
Button btnStats = new Button(text: "📊 Статистика");
btnStats.setOnAction(e -> showCompositeStats());
```

```java
ToolBar toolbar = new ToolBar(
        new Label(text: "Назва:"), titleField, btnSave,
        new Separator(),
        btnAddChild, btnAddImg, btnAddVid, btnUrgent, btnArea,
        new Separator(),
        btnExport,
        btnStats,
        btnTheme,
        btnLineStyle);
```

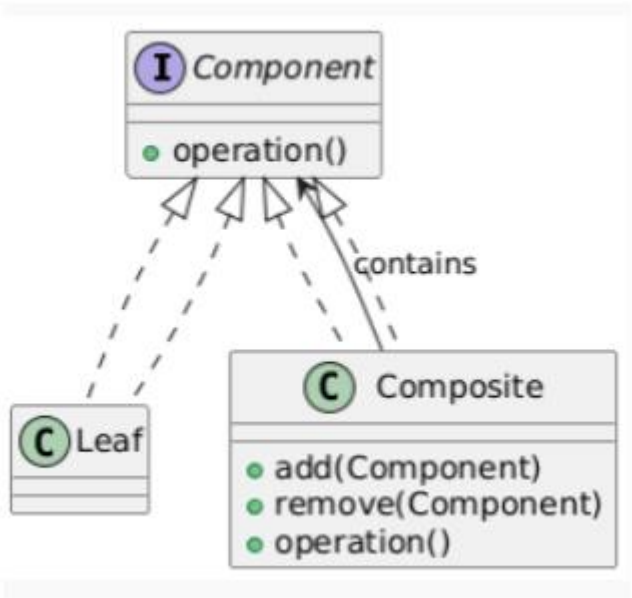
Відповіді на контрольні питання:

1. Яке призначення шаблону «Композит»?

Шаблон Композит (Composite) використовується для представлення об’єктів у вигляді ієрархічної деревоподібної структури, де окремі об'єкти та групи

об'єктів обробляються однаково. Це дозволяє клієнту працювати з одиничними елементами та колекціями через спільний інтерфейс.

2. Нарисуйте структуру шаблону «Композит».



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

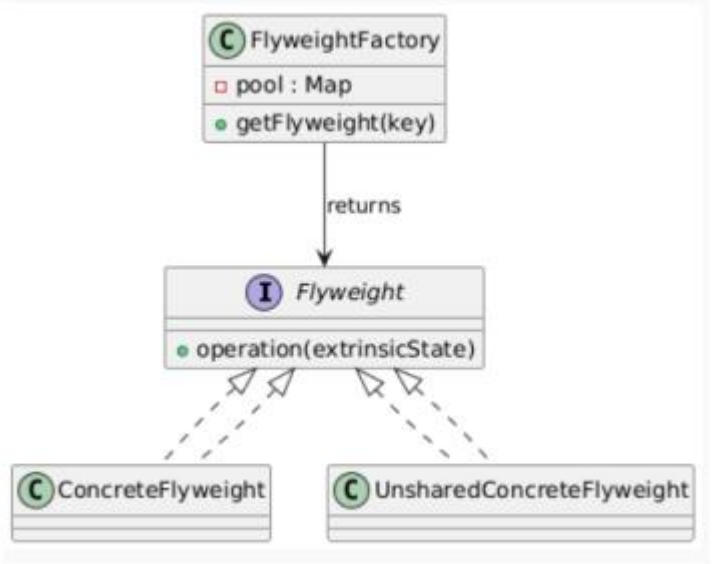
- Component - спільний інтерфейс для листків і контейнерів
- Leaf - елементарний об’єкт без дочірніх елементів
- Composite - вузол, який містить інші компоненти

Composite містить список Component і делегує виклики дочірнім елементам. 8

4. Яке призначення шаблону «Легковаговик»?

Шаблон Легковаговик дозволяє економити пам’ять шляхом повторного використання спільних об’єктів, розділяючи стан на внутрішній (shared) і зовнішній (у клієнта). Використовується при великій кількості дрібних однотипних об’єктів.

5. Нарисуйте структуру шаблону «Легковаговик».



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

- Flyweight - інтерфейс об'єктів з внутрішнім станом
- ConcreteFlyweight - спільний об’єкт
- UnsharedFlyweight - об’єкт, який не розділяється
- FlyweightFactory - повертає існуючі екземпляри або створює нові

Зовнішній стан передається в методи замість зберігання в об’єкті.

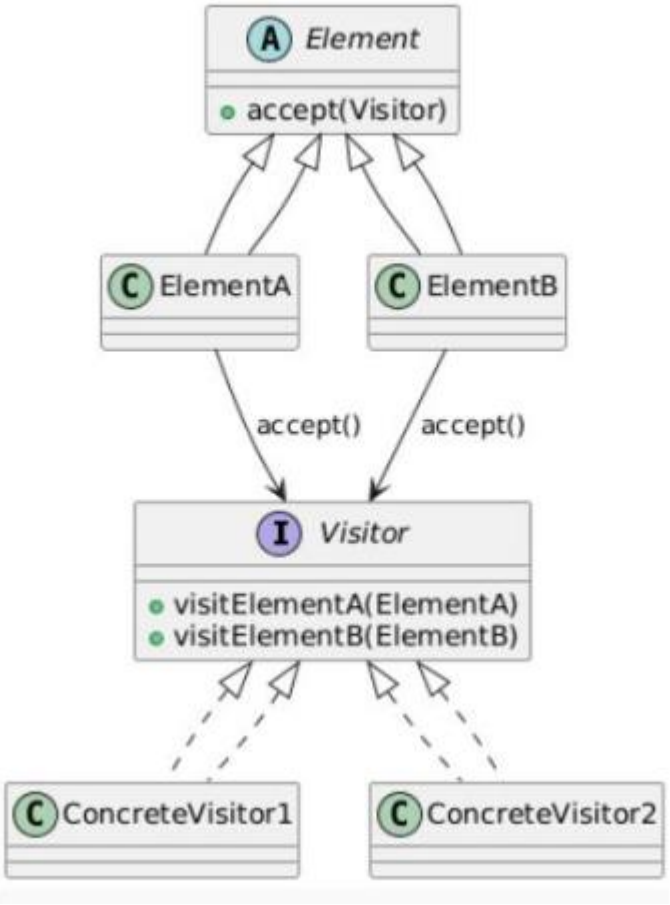
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон Інтерпретатор визначає спосіб опису граматики та інтерпретації 9 мовоподібних команд. Використовується для побудови простих мов, обробки виразів, формул, логічних правил.

8. Яке призначення шаблону «Відвідувач»?

Шаблон Відвідувач (Visitor) дозволяє додавати нові операції до об’єктів без модифікації їхніх класів, переміщуючи поведінку в окремий клас-відвідувач. Це корисно, коли треба виконувати багато різних операцій над об’єктами складної структури.

9. Нарисуйте структуру шаблону «Відвідувач».



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

- Element - елемент структури
- ConcreteElement - конкретні елементи
- Visitor - інтерфейс операцій • ConcreteVisitor — реалізації операцій

Взаємодія через double dispatch:

element.accept(visitor) → visitor.visit(element).