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

Технології розробки програмного забезпечення

“ Патерни проектування.”

“Download manager”

Виконав студент групи ІА–33:
Ященко К.А.

Київ 2025

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

Зміст

Завдання 2
Теоретичні відомості 2
Тема проєкту 4
Хід роботи 4
Основні елементи діаграми 5
Зв’язки та взаємодія між елементами 5
Переваги використання шаблону Composite у проєкті 5
Частина коду програми з Composite 6
Опис програмного коду 8
Висновок 9
Відповіді на контрольні питання 9

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

Composite (Компонувальник)
Цей структурний патерн вирішує проблему побудови складних деревоподібних структур, дозволяючи працювати з ними через єдиний інтерфейс. Ключова ідея полягає в тому, що клієнтський код не повинен

розрізняти прості об'єкти (листки) та складені об'єкти (контейнери). Усі елементи дерева реалізують спільний інтерфейс, який зазвичай містить методи для

виконання основної бізнес-логіки. Для простих компонентів метод просто виконує дію, а для контейнерів він проходить по списку всіх дочірніх елементів, викликає цей метод для кожного з них і підсумовує або агрегує результат. Це дозволяє будувати рекурсивні структури будь-якої глибини вкладеності.
На практиці цей патерн ідеально підходить для реалізації графічних редакторів (де фігури можна групувати в складніші об'єкти), файлових систем (де папки можуть містити файли та інші папки) або меню програмного забезпечення. Головною перевагою є спрощення клієнтського коду, оскільки зникає необхідність перевіряти типи об'єктів перед викликом методів. Однак, недоліком може стати занадто загальний інтерфейс: іноді доводиться додавати методи управління дітьми (додати/видалити) в інтерфейс листка, де вони не мають сенсу, що порушує принцип розділення інтерфейсу.

Flyweight (Легковаговик)
Основна мета цього структурного патерну — економія оперативної пам'яті в системах із величезною кількістю об'єктів. Патерн досягає цього шляхом розділення стану об'єкта на дві частини: внутрішній (intrinsic) та зовнішній (extrinsic). Внутрішній стан — це дані, які є спільними для багатьох об'єктів і не змінюються (наприклад, форма кулі, текстура дерева, шрифт літери). Зовнішній стан — це контекстні дані, унікальні для кожного екземпляра (координати, колір, розмір), які передаються об'єкту лише в момент виконання методу.
Легковаговик — це незмінний об'єкт, який ініціалізується один раз і використовується в різних контекстах. Зазвичай створення таких об'єктів контролюється спеціальною фабрикою, яка кешує вже створені екземпляри: якщо об'єкт із потрібним внутрішнім станом вже існує, фабрика повертає його, замість створення нового. Це критично важливо для ігрових рушіїв (відображення лісу, армії солдатів) або текстових редакторів (де кожна літера є об'єктом). Головний мінус — ускладнення коду програми та необхідність перерахунку зовнішніх даних кожного разу при зверненні до об'єкта, що може незначно вплинути на

процесорний час заради економії пам'яті.

Interpreter (Інтерпретатор)
Цей поведінковий патерн використовується для проектування мовних інтерпретаторів. Він визначає граматику простої мови та будує інтерпретатор для речень цієї мови. Кожне правило граматики (наприклад, змінні, числа, операції додавання чи віднімання) перетворюється на окремий клас. Речення мови представляється у вигляді дерева об'єктів (Абстрактне Синтаксичне Дерево), де вузли — це операції, а листки — термінальні вирази (значення). Щоб виконати програму або обчислити вираз, інтерпретатор рекурсивно проходить по дереву, викликаючи метод interpret для кожного вузла.
Патерн найкраще підходить для специфічних, вузьконаправлених задач: розбору математичних виразів, SQL-подібних запитів, регулярних виразів або конфігураційних файлів. Проте він стає вкрай неефективним для складних мов програмування з великою кількістю правил, оскільки кількість класів зростає лавиноподібно, а дерево стає надто громіздким для обробки. У таких випадках зазвичай використовують повноцінні парсери та компілятори, а не патерн Інтерпретатор.

Visitor (Відвідувач)
Цей поведінковий патерн дозволяє відокремити алгоритми від структури об'єктів, над якими вони оперують. Це особливо корисно, коли у вас є стабільна ієрархія класів (наприклад, типи вузлів у дереві документа: текст, картинка, таблиця), яка рідко змінюється, але вам часто потрібно додавати нові операції над цими даними (експорт у XML, генерація звіту, підрахунок статистики). Замість того, щоб додавати новий метод у кожен клас даних (що порушує принцип відкритості/закритості), ви створюєте окремий клас — Відвідувач, який містить методи обробки для кожного типу елемента.
Технічною основою патерну є механізм подвійної диспетчеризації (Double Dispatch). Кожен елемент структури має метод accept, який приймає об'єкт

відвідувача і викликає у нього відповідний метод (наприклад, visitor.visitTable(this)). Таким чином, вибір потрібного методу залежить і від типу відвідувача, і від типу елемента. Це дозволяє додавати скільки завгодно нових операцій, просто створюючи нові класи відвідувачів, не чіпаючи існуючий код класів даних. Недоліком є те, що при додаванні нового типу елемента в структуру (наприклад, "Відео"), доведеться оновлювати всі існуючі класи відвідувачів.

Тема проєкту

26. Download manager (iterator, command, observer, template method, composite, p2p)
Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Паттерн: Composite
Посилання на GitHub: https://github.com/fl1ckyexe/trpz

Хід роботи

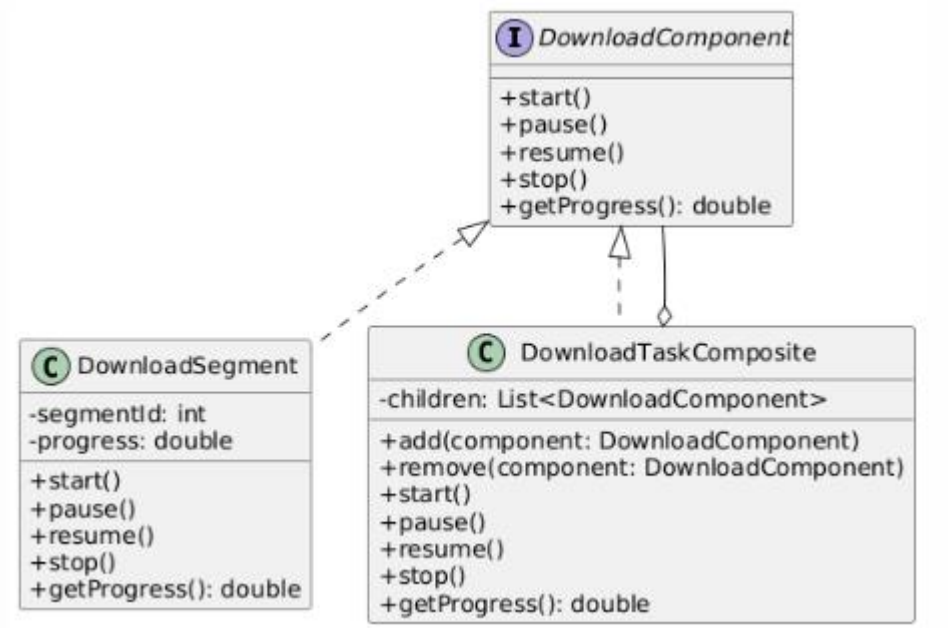


Рис. 1 – Діаграма класів для Composite

Діаграма класів відображає застосування шаблону проєктування Composite для організації структури завантаження файлів у менеджері завантажень. Завдяки цьому шаблону окремі сегменти файлу та ціле завдання завантаження розглядаються як єдині об’єкти з однаковим інтерфейсом керування.

Основні елементи діаграми

Інтерфейс DownloadComponent визначає спільний набір операцій для всіх елементів ієрархії завантаження, таких як запуск, пауза, відновлення, зупинка та отримання прогресу.

Клас DownloadSegment представляє листовий елемент ієрархії. Він відповідає за завантаження окремого сегмента файлу та реалізує всі операції, визначені в інтерфейсі компонента.

Клас DownloadTaskComposite є складеним елементом (Composite), який містить колекцію об’єктів типу DownloadComponent. Він дозволяє об’єднувати кілька сегментів у єдине завдання завантаження.

Зв’язки та взаємодія між елементами

Класи DownloadSegment та DownloadTaskComposite реалізують інтерфейс DownloadComponent, що дозволяє працювати з ними однаковим чином.

DownloadTaskComposite містить список дочірніх компонентів і делегує їм виконання операцій керування. Наприклад, при виклику методу start запуск відбувається для всіх вкладених сегментів.

Метод getProgress у складеному компоненті може обчислювати загальний прогрес на основі прогресу дочірніх сегментів.

Переваги використання шаблону Composite у проєкті

У межах нашого проєкту шаблон Composite був використаний для моделювання сегментного завантаження файлів. Кожен файл розбивається на окремі сегменти, які представлені об’єктами DownloadSegment.

Завдання завантаження файлу реалізовано як об’єкт DownloadTaskComposite, що об’єднує всі сегменти та дозволяє керувати ними як єдиним цілим. Це дало змогу реалізувати запуск, паузу та зупинку завантаження незалежно від кількості сегментів.

Застосування шаблону Composite спростило архітектуру системи, зробило її більш гнучкою та дозволило обробляти складні ієрархії об’єктів завантаження через єдиний інтерфейс.

Частина коду програми з Composite

```
// DownloadComponent.java
public interface DownloadComponent {
    void start();
    void pause();
    void resume();
    void stop();
    double getProgress();
}

// DownloadSegment.java
public class DownloadSegment implements DownloadComponent {

    private int segmentId;
    private double progress;

    public DownloadSegment(int segmentId) {
        this.segmentId = segmentId;
        this.progress = 0.0;
    }

    @Override
    public void start() {
        // Початок завантаження сегмента
        progress = 10.0;
    }

    @Override
    public void pause() {
        // Призупинення завантаження сегмента
    }

    @Override
    public void resume() {
        // Відновлення завантаження сегмента
    }

    @Override
    public void stop() {
        // Зупинка завантаження сегмента
        progress = 0.0;
    }

    @Override
    public double getProgress() {
        return progress;
    }
```

```
}

// DownloadTaskComposite.java
import java.util.ArrayList;
import java.util.List;

public class DownloadTaskComposite implements DownloadComponent {

    private List<DownloadComponent> children = new ArrayList<>();

    public void add(DownloadComponent component) {
        children.add(component);
    }

    public void remove(DownloadComponent component) {
        children.remove(component);
    }

    @Override
    public void start() {
        for (DownloadComponent component : children) {
            component.start();
        }
    }

    @Override
    public void pause() {
        for (DownloadComponent component : children) {
            component.pause();
        }
    }

    @Override
    public void resume() {
        for (DownloadComponent component : children) {
            component.resume();
        }
    }

    @Override
    public void stop() {
        for (DownloadComponent component : children) {
            component.stop();
        }
    }

    @Override
    public double getProgress() {
        if (children.isEmpty()) return 0.0;
```

```
        double total = 0.0;
        for (DownloadComponent component : children) {
            total += component.getProgress();
        }
        return total / children.size();
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {

        DownloadSegment segment1 = new DownloadSegment(1);
        DownloadSegment segment2 = new DownloadSegment(2);
        DownloadSegment segment3 = new DownloadSegment(3);

        DownloadTaskComposite downloadTask = new DownloadTaskComposite();
        downloadTask.add(segment1);
        downloadTask.add(segment2);
        downloadTask.add(segment3);

        downloadTask.start();

        System.out.println("Progress: " + downloadTask.getProgress() + "%");
    }
}
```

Опис програмного коду

Поданий програмний код реалізує шаблон проєктування Composite у межах десктопного менеджера завантажень файлів. Основною метою використання даного шаблону є уніфікація роботи з окремими сегментами файлу та з усім завданням завантаження загалом.

Інтерфейс DownloadComponent визначає спільний набір операцій керування завантаженням. Це дозволяє працювати з окремими сегментами та групами сегментів однаковим чином.

Клас DownloadSegment представляє листовий елемент ієрархії та відповідає за завантаження окремої частини файлу. Клас DownloadTaskComposite є складеним компонентом, який містить колекцію об’єктів DownloadComponent та делегує їм виконання керуючих операцій.

Метод getProgress у складеному компоненті обчислює загальний прогрес завантаження на основі прогресу всіх вкладених сегментів. Це дозволяє отримувати узагальнену інформацію про стан завантаження незалежно від кількості сегментів.

Висновок

У ході виконання лабораторної роботи було реалізовано шаблон проєктування Composite для моделювання сегментного завантаження файлів у менеджері завантажень. Застосування даного шаблону дозволило уніфікувати керування окремими сегментами файлу та цілим завданням завантаження.

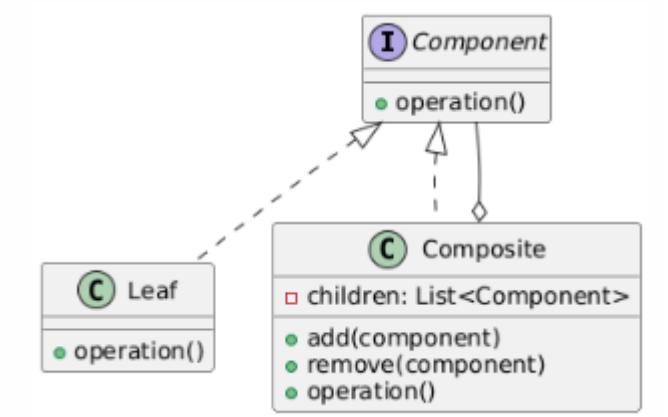
Використання Composite спростило архітектуру системи, зменшило залежності між компонентами та підвищило гнучкість реалізації. Отримане рішення демонструє коректне застосування шаблону та створює надійну основу для подальшого розвитку функціональності менеджера завантажень, зокрема реалізації багатопотокового та P2P-завантаження.

Відповіді на контрольні питання

Питання та відповіді до лабораторної роботи

1. Яке призначення шаблону «Композит»?
Шаблон проєктування «Композит» призначений для побудови ієрархій об’єктів типу «частина–ціле», у яких окремі об’єкти та групи об’єктів можуть оброблятися однаковим чином. Він дозволяє клієнту працювати з одиночними об’єктами та їх композиціями через єдиний інтерфейс.

2. Нарисуйте структуру шаблону «Композит».

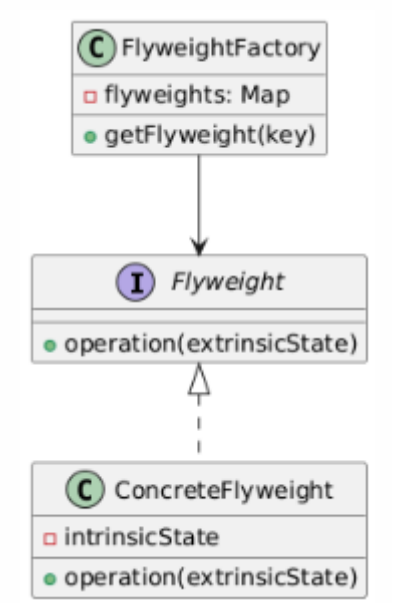


3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?
У шаблоні «Композит» використовуються інтерфейс Component, листові об’єкти Leaf та складені об’єкти Composite. Composite містить колекцію Component і делегує їм виконання операцій. Клієнт працює з усіма об’єктами через інтерфейс Component, не розрізняючи одиночні та складені елементи.

4. Яке призначення шаблону «Легковаговик»?
Шаблон «Легковаговик» (Flyweight) призначений для зменшення споживання пам’яті шляхом спільного використання об’єктів, які мають однаковий внутрішній стан. Він дозволяє ефективно працювати з великою кількістю дрібних

об’єктів.

5. Нарисуйте структуру шаблону «Легковаговик».

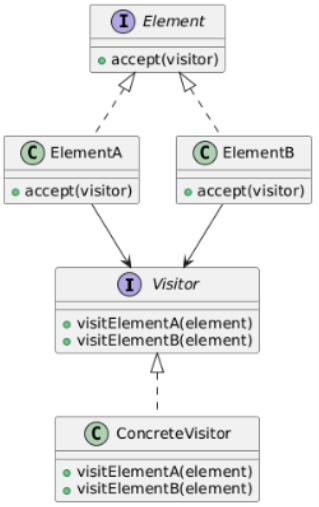


6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?
У шаблоні використовуються інтерфейс Flyweight, клас ConcreteFlyweight та фабрика FlyweightFactory. Фабрика створює та зберігає об’єкти легковаговиків і повертає вже існуючі екземпляри при повторних запитах. Клієнт передає зовнішній стан об’єкта під час виклику операцій.

7. Яке призначення шаблону «Інтерпретатор»?
Шаблон «Інтерпретатор» призначений для визначення граматики простої мови та реалізації механізму інтерпретації виразів цієї мови. Він часто використовується для обробки формальних мов, правил або простих скриптів.

8. Яке призначення шаблону «Відвідувач»?
Шаблон «Відвідувач» (Visitor) призначений для винесення операцій над об’єктами в окремі класи без зміни самих об’єктів. Це дозволяє додавати нову функціональність до структури об’єктів без модифікації їх класів.

9. Нарисуйте структуру шаблону «Відвідувач».



10. Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?
У шаблоні беруть участь інтерфейс Visitor, конкретні відвідувачі, інтерфейс Element та конкретні елементи. Кожен елемент приймає відвідувача через метод accept і передає себе як параметр. Відвідувач виконує відповідну операцію для конкретного типу елемента.