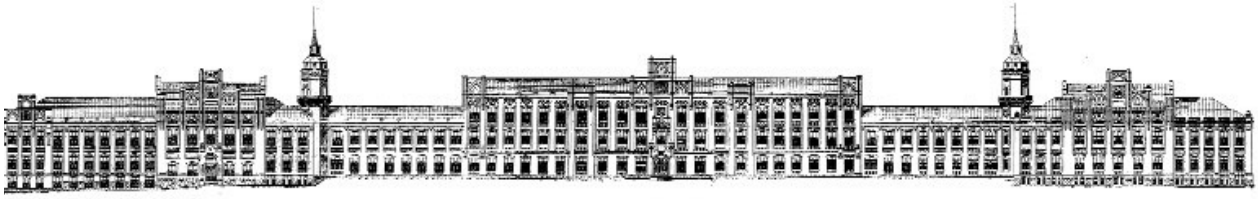


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №8
Тема: «Патерни проектування»

Виконала
студентка групи ІА-32:
Іванова Анастасія
Юріївна

Перевірив:
Мягкий Михайло
Юрійович

Зміст

.....1
Тема проекту..... 3
Теоретичні відомості.....3
Хід роботи.....5
Створення діаграми класів патерну.....5
Висновки:.....9
Контрольні питання..... 10

Тема проекту

Варіант: 26
Опис теми: 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).

Теоретичні відомості

Патерни проєктування (Design Patterns) — це типові, перевірені на практиці способи розв'язання поширених проблем, які виникають під час проєктування програмного забезпечення.
Вони описують загальні принципи побудови структури класів і взаємодії об'єктів, не прив'язуючись до конкретної мови програмування. Патерни не є готовим кодом — це шаблони, за якими можна побудувати гнучке, масштабоване й зрозуміле рішення.
Використання патернів дозволяє:

- підвищити гнучкість і розширюваність системи;
- зменшити дублювання коду й спростити супровід;
- покращити зрозумілість архітектури для інших розробників;
- забезпечити повторне використання перевірених рішень у різних проєктах.

Composite (Компонувальник) — це структурний шаблон проєктування, який використовується для представлення ієрархічних структур типу «частина–ціле». Він дозволяє уніфіковано працювати як з окремими об'єктами (*leaf*), так і з групами об'єктів (composite), не розрізняючи їх під час виконання.
Призначення шаблону
Шаблон Composite використовується для того, щоб:

- будувати деревоподібні структури об'єктів;
- виконувати операції над окремими елементами та над групами елементів однаковим способом;
- спростити роботу клієнтського коду, який не повинен знати, чи перед ним листовий елемент, чи складний об'єкт.

У всіх елементів дерева має бути єдиний інтерфейс.

- Leaf (Лист) — окремий елемент без дочірніх об'єктів. Реалізує базову поведінку.
- Composite (Компонувальник) — містить колекцію елементів (і листів, і інших Composite) та реалізує функції, делегуючи операції всім дочірнім елементам рекурсивно.
- Client (Клієнт) — працює через базовий інтерфейс Component, не знаючи, з яким саме типом елемента має справу.

Переваги:

- Спрощення представлення деревоподібної структури.
- Гнучкість при роботі зі складними об’єктами.
- Легке додавання та видалення об’єктів без зміни логіки клієнта.

Недоліки:

- Потрібні додаткові зусилля для впровадження структури.
- Потрібен добре продуманий загальний інтерфейс, інакше шаблон ускладнить систему.

Хід роботи

Створення діаграми класів патерну

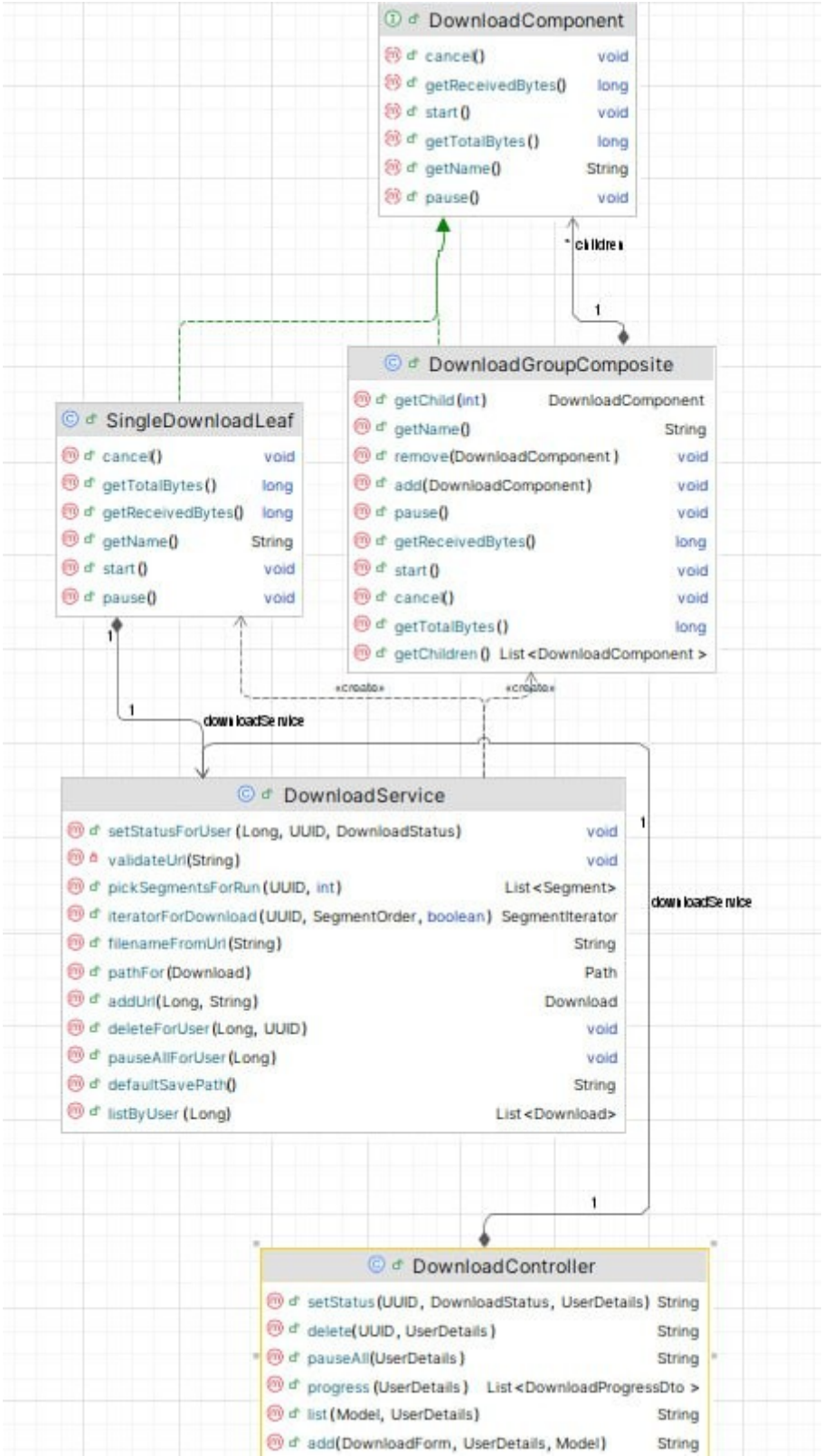


Рис. 1. Діаграма класів патерну Composite

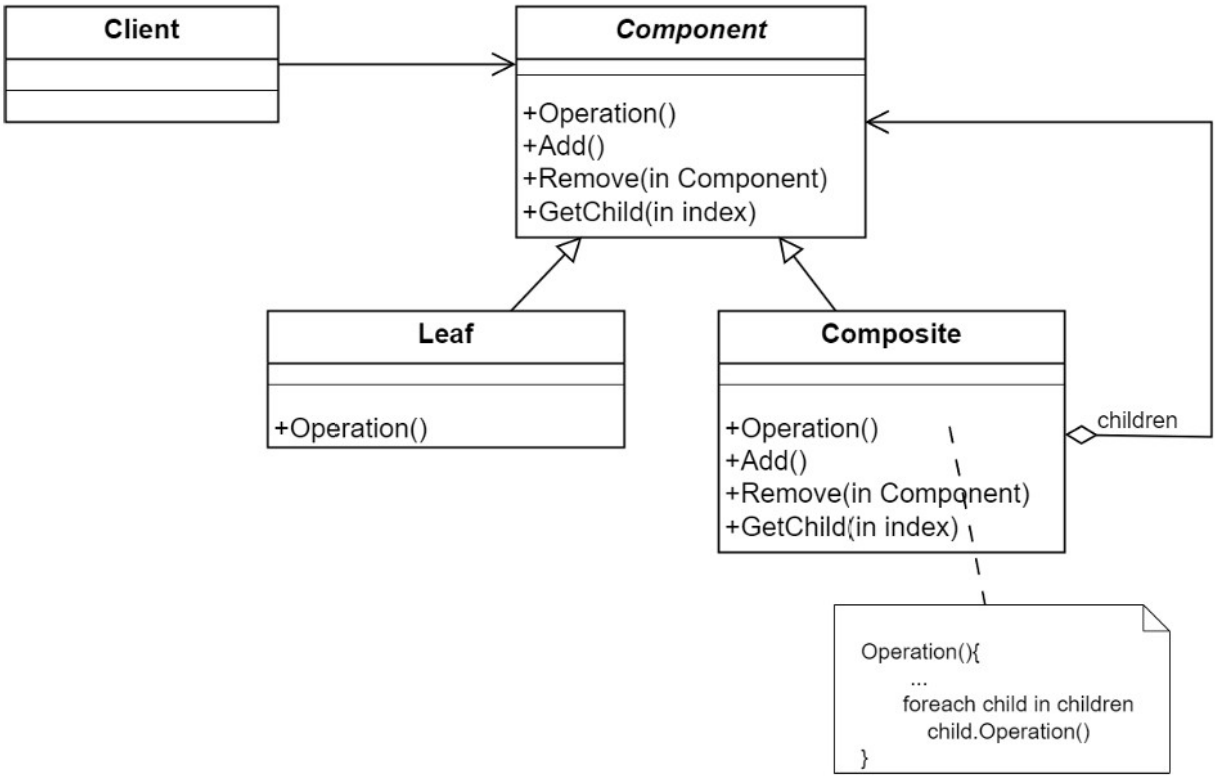


Рис. 2. Шаблон Composite

1. Component → DownloadComponent:

Задає єдиний інтерфейс для будь-якого елемента дерева завантажень — як для окремого завантаження, так і для групи завантажень.

Фрагмент коду:

```
public interface DownloadComponent {

    void start();
    void pause();
    void cancel();

    long getTotalBytes();
    long getReceivedBytes();
    String getName();
}
```

Роль:

Цей інтерфейс використовується однаково як для окремих завантажень (Leaf), так і для груп (Composite).
Він визначає набір операцій, які можна виконувати над завантаженнями.

2. Leaf → SingleDownloadLeaf:

Представляє одне конкретне завантаження — кінцевий елемент дерева, що не має дочірніх вузлів.

Фрагмент коду:

```
public class SingleDownloadLeaf implements DownloadComponent {

   @Override
   public void start() {
      downloadService.setStatusForUser(userId, downloadId,
DownloadStatus.RUNNING);
   }

   @Override
   public void pause() {
      downloadService.setStatusForUser(userId, downloadId,
DownloadStatus.PAUSED);
   }

   @Override
   public void cancel() {
      downloadService.setStatusForUser(userId, downloadId,
DownloadStatus.CANCELED);
   }

   @Override
   public long getTotalBytes() { ... }

   @Override
   public long getReceivedBytes() { ... }

   @Override
   public String getName() { ... }
}
```

Роль:
Leaf виконує Operation() напряму.

3. Composite → DownloadGroupComposite:

Представляє групу завантажень, яка може містити інші компоненти: як SingleDownloadLeaf, так і інші DownloadGroupComposite.

Фрагмент коду:

```
public class DownloadGroupComposite implements DownloadComponent {
```

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

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

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

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

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

    @Override
    public void cancel() {
        for (DownloadComponent child : children)
            child.cancel();
    }

    @Override
    public long getTotalBytes() {
        return
children.stream().mapToLong(DownloadComponent::getTotalBytes).sum();
    }
}
```

Роль:
DownloadGroupComposite реалізує той самий набір операцій, що і Leaf, але зберігає список дочірніх елементів (children) та реалізує поведінку методом рекурсивної делегації.

4. Client → DownloadService + DownloadController:

Клієнтський код працює лише з інтерфейсом DownloadComponent, не розрізняючи, чи це один елемент, чи група.

Фрагмент коду:

```
public void pauseAllForUser(Long userId) {
    List<Download> list = listByUser(userId);
    DownloadGroupComposite root = new DownloadGroupComposite("All
downloads");

    for (Download d : list) {
        root.add(new SingleDownloadLeaf(userId, d.getId(), this, downloads));
    }

    root.pause();
}
```

Роль:
Клієнт працює з колекцією елементів через єдиний інтерфейс, не залежачи від того, чи є об'єкт складним чи простим.

Робота патерну в застосунку:

Кнопка «Усі» зупиняє всі активні завантаження.

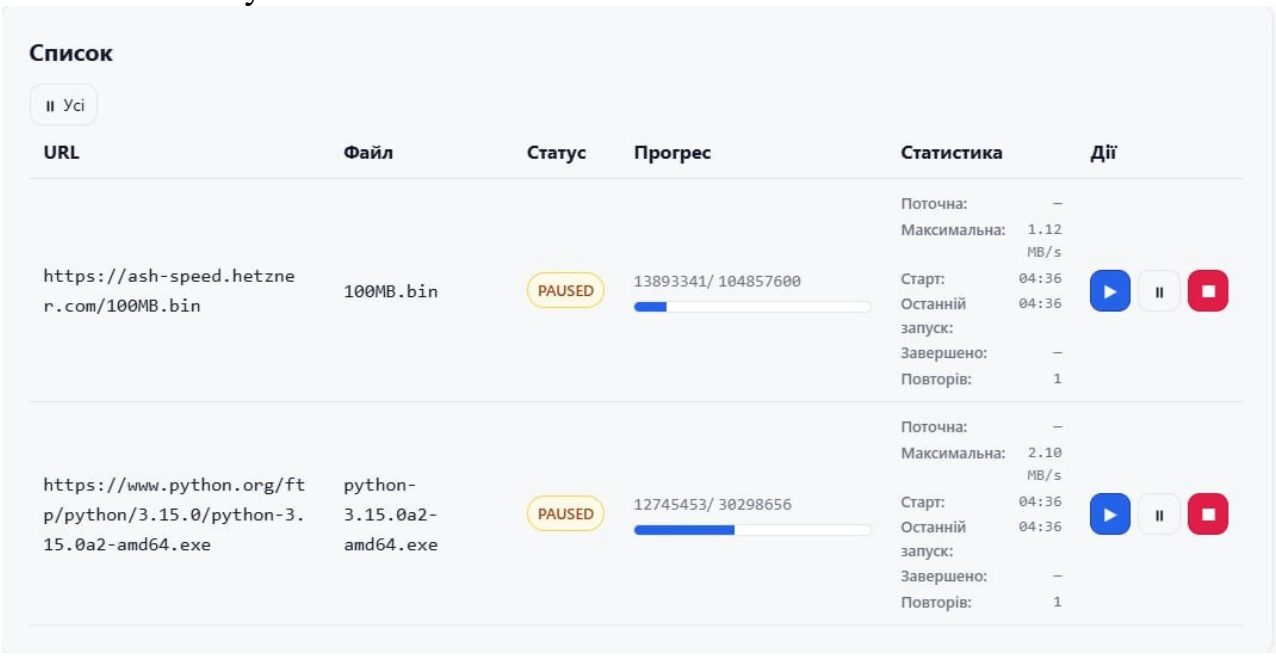


Рис. 3. Робота патерну у застосунку.

Висновки:

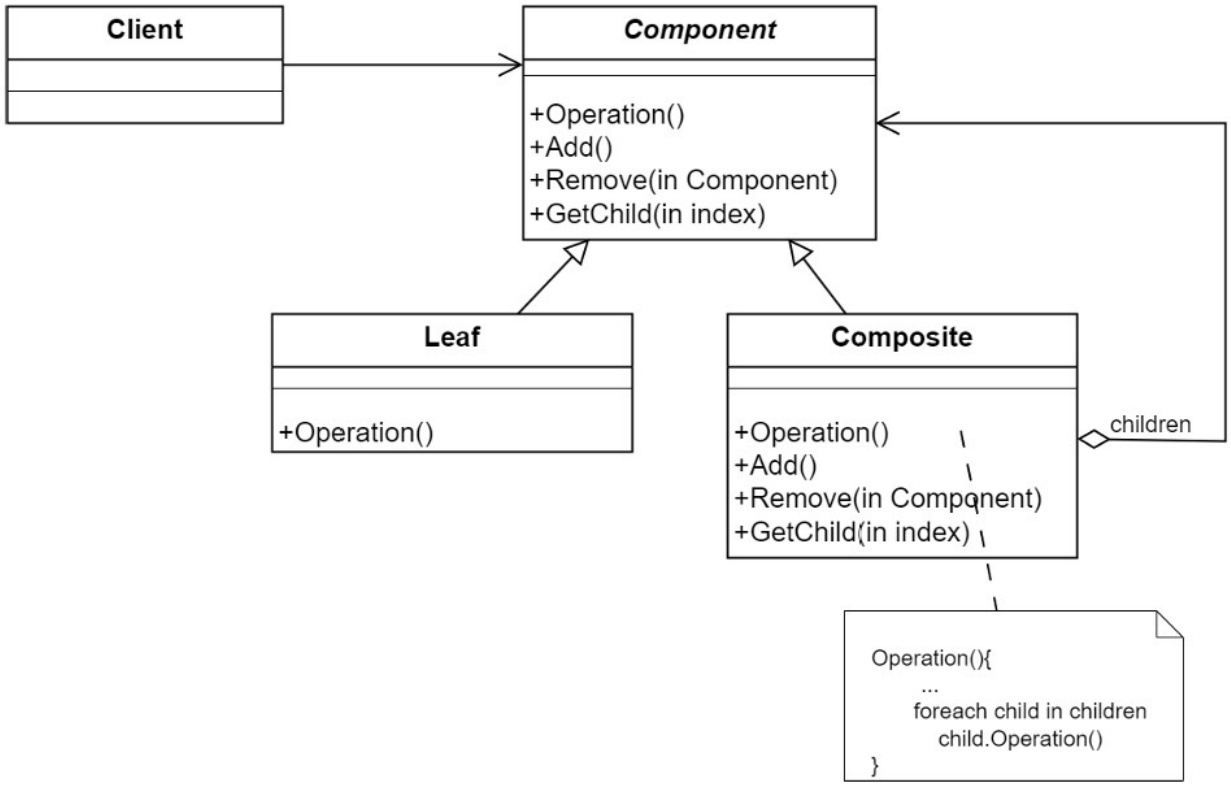
У моєму застосунку патерн Composite використовується для реалізації групових операцій над завантаженнями.
Інтерфейс DownloadComponent задає єдиний контракт для роботи як з одиночними завантаженнями (SingleDownloadLeaf), так і з групами (DownloadGroupComposite).
Leaf виконує операції безпосередньо над одним завантаженням, тоді як Composite делегує операцію всім дочірнім елементам, забезпечуючи рекурсивне виконання (паузу всіх завантажень).

Клієнтський код у DownloadService та DownloadController працює лише з інтерфейсом Component, не розрізняючи тип елемента.
Патерн Composite виконує такі функції у застосунку:
• забезпечує можливість представляти множину завантажень у вигляді деревоподібної структури;
• дозволяє виконувати групові операції єдиною командою;
• спрощує клієнтський код і робить його незалежним від внутрішньої організації завантажень;
• дає можливість легко додавати нові рівні ієрархії без модифікації логіки роботи сервісів;

Контрольні питання

1. Яке призначення шаблону «Композит»?
Патерн Композит (Composite) використовується для представлення об’єктів у вигляді деревоподібної структури «частина–ціле», що дозволяє однаково обробляти як поодинокі об’єкти, так і складені структури.

2. Нарисуйте структуру шаблону «Композит».

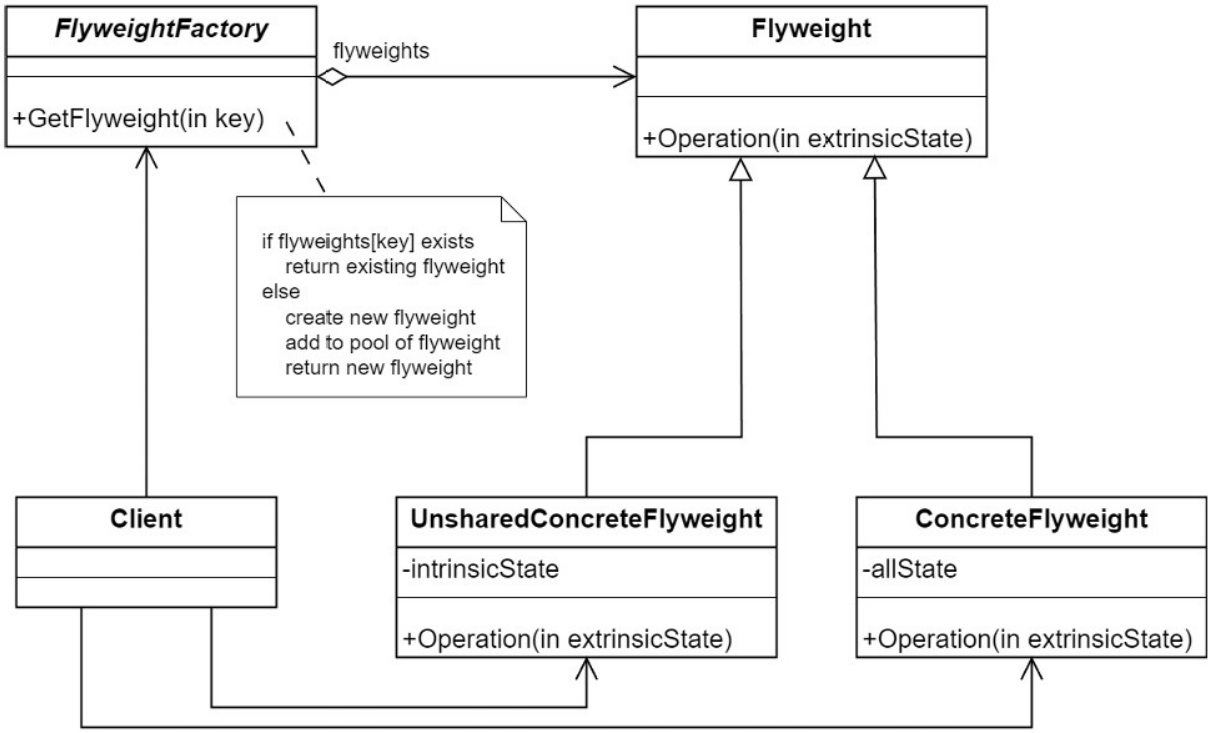


3. Які класи входять у шаблон «Композит», та яка між ними взаємодія?
Component — загальний інтерфейс для листів і композицій.
Leaf — кінцевий елемент без дочірніх об’єктів.
Composite — містить колекцію Component і реалізує операції, делегуючи їх дочірнім елементам рекурсивно.
Client — працює через інтерфейс Component.

Composite викликає методи Leaf та інших Composite однаковим чином.

4. Яке призначення шаблону «Легковаговик» (Flyweight)?
Патерн використовується для зменшення кількості об’єктів, створюючи поділювані екземпляри при роботі з великою кількістю однакових об’єктів. Він розділяє стан на внутрішній (зберігається в об’єкті) та зовнішній (зберігається в контексті).

5. Нарисуйте структуру шаблону «Легковаговик».



6. Які класи входять у шаблон «Легковаговик», та яка між ними взаємодія?
Flyweight — інтерфейс поділюваних об’єктів.

ConcreteFlyweight — реалізація об’єкта з внутрішнім станом.

UnsharedConcreteFlyweight — неподовжуваний об’єкт (не обов’язковий).

FlyweightFactory — створює та кешує екземпляри Flyweight.

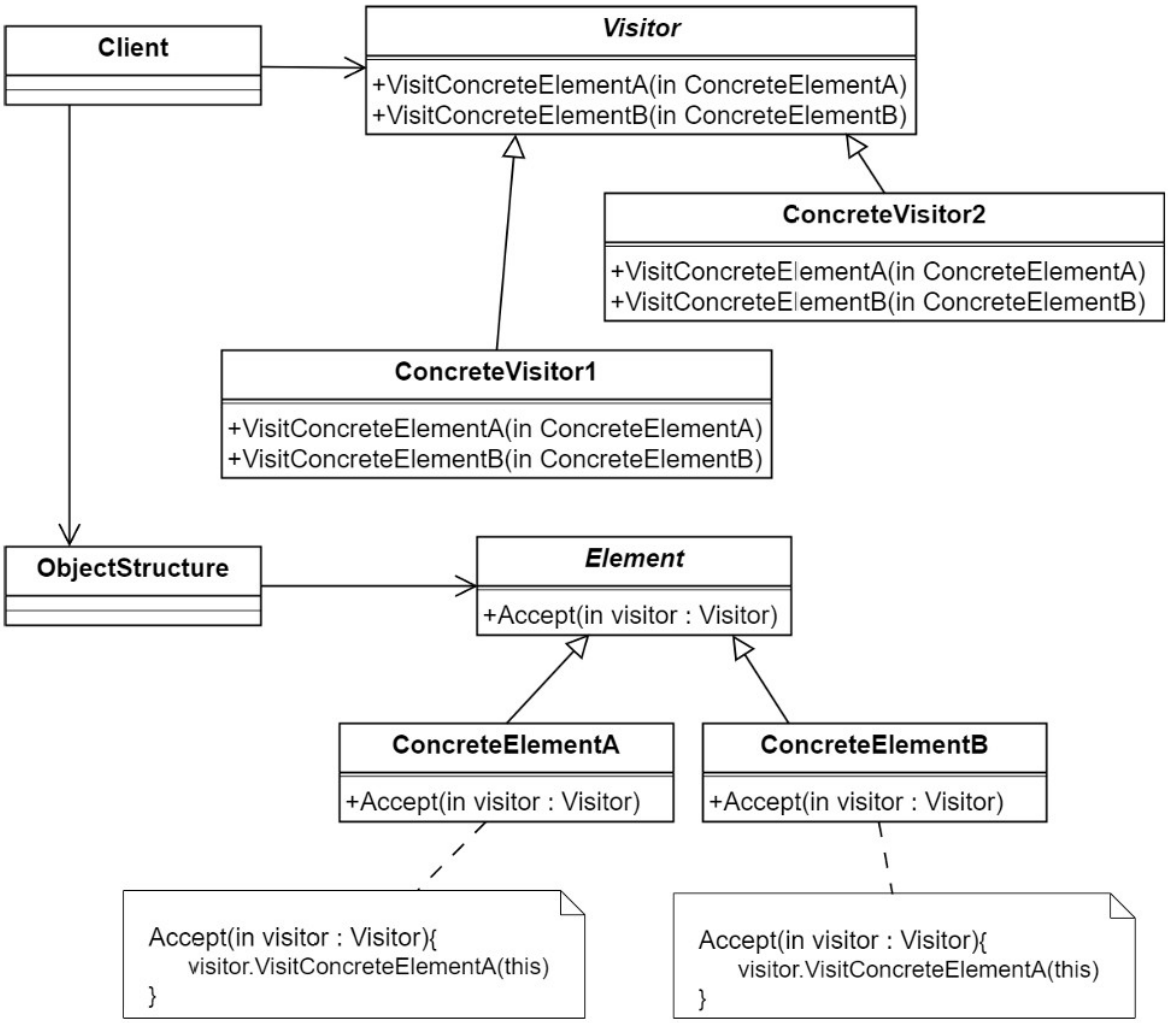
Client — зберігає зовнішній стан і використовує Flyweight.

Factory повертає існуючі об’єкти або створює нові при потребі.

7. Яке призначення шаблону «Інтерпретатор»?
Патерн застосовується для опису граматики мови та створення інтерпретатора, який може обчислювати вирази, представлені у вигляді абстрактного синтаксичного дерева.

8. Яке призначення шаблону «Відвідувач» (Visitor)?
Патерн дозволяє визначати нові операції над об’єктами складної структури без зміни їхніх класів, відокремлюючи логіку обробки від структури даних.

9. Нарисуйте структуру шаблону «Відвідувач».



10. Які класи входять у шаблон «Відвідувач», та яка між ними взаємодія?
Visitor — інтерфейс з методами відвідування елементів різних типів.
ConcreteVisitor — реалізує конкретні операції над елементами.
Element — інтерфейс елементів, які приймають відвідувача.
ConcreteElement — елемент, що викликає відповідний метод Visitor.
ObjectStructure — зберігає колекцію елементів і дозволяє виконати обхід.
Елемент викликає visitor.VisitConcreteElementX(this) → відвідувач обробляє об’єкт відповідно до його типу.