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

«Патерни проектування»

Виконав:
студент групи ІА–32
Лось Ярослав

Київ 2025

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

Зміст

Теоретичні відомості

Шаблон «Visitor»

Призначення: 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 (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача).

Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

Проблема: Ви розробляєте онлайн-корзину інтернет магазина. Товари які представлені в магазині є різних типів, наприклад, електроніка, міцні напої, домашня хімія.

Логіка роботи з товарами в корзині є різна, наприклад, розрахунок вартості, формування замовлення.

Ми можемо всі ці методи зробити в товарах, але тоді ми ускладнюємо товари і змішуємо логіку (розрахунок вартості) з даними (товаром та його кількістю).

Якщо ми в подальшому необхідно буде добавити ще логіку розрахунку вартості з врахуванням знижки, то потрібно буде добавити ще цю логіку до товарів. А якщо буде ще сезонна знижка, то ми знову будемо добавляти нову логіку до класів товарів.

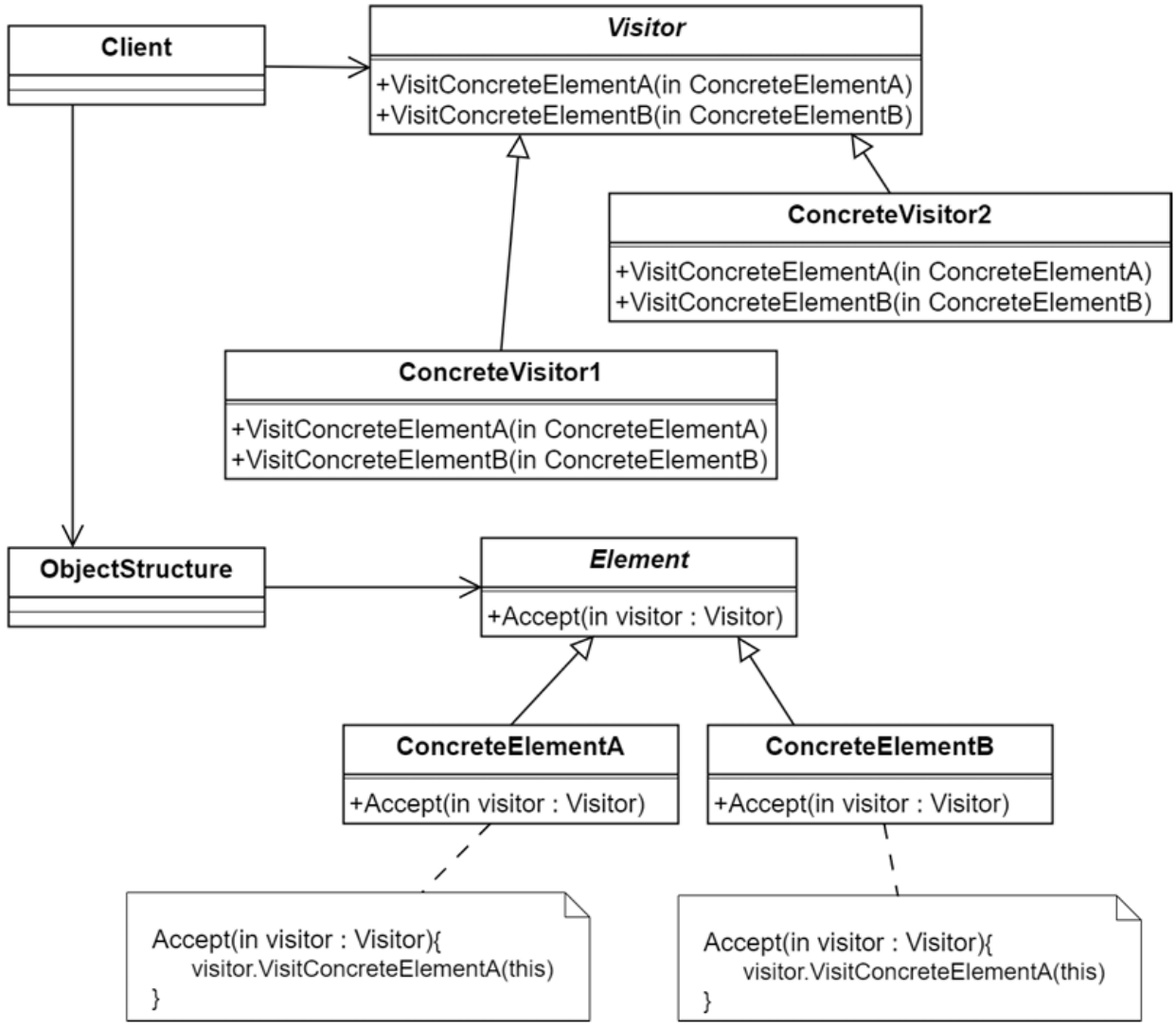


Рисунок 8.5. Структура патерна «Відвідувач»

Рішення: Основна ідея патерна «Відвідувач» це рознести логіку і дані в різні класи та ієрархії. Якщо так зробити для нашої онлайн-корзини, то в класі відвідувача ми маємо функції для обрахунку логіки для кожного типу, а об'єкти в корзині знають свій тип, отримують екземпляр відвідувача і в нього викликають метод відповідно до свого типу. В результаті ми робимо різні відвідувачі для розрахунку вартості: один для звичайних розрахунків, інший для розрахунку вартості зі знижкою, ще один для розрахунку сезонних знижок.

За рахунок того, що логіка відокремлена від наших товарів в корзині ми можемо реалізовувати за необхідності нові класи відвідувачі, а класи товарів та і корзини, в цілому, змінюватися не будуть.

Якщо розвивати далі, то можна зробити і клас відвідувача, який буде формувати замовлення з товарів в корзині в залежності від продавців і можливих варіантів доставки. Це дозволить формувати, наприклад, не одне замовлення, а два одна з самовивозом з магазину, а інше з доставкою Новою Поштою.

Приклад з життя: Прикладом може служити написання компілятора. Припустимо, існують різні об'єкти в синтаксисі мови програмування: виклики методів і умовні вирази. Компліятор перед генерацією коду повинен обійти всі вирази (і виклики методів, і умовні вирази) і перевірити безпеку типів, після чого згенерувати відповідний код. Відповідно буде два відвідувачі – для перевірки безпеки типів і для генерації коду. У кожного з них буде по 2 методи – для викликів методів і для умовних операцій. Таким чином при необхідності додавання нових кроків компіляції досить буде визначити нового «відвідувача» і викликати його у відповідний час.

Хід роботи

1. **Музичний програвач** (iterator, command, memento, facade, visitor, client-server)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам’ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіо-форматів, еквалайзер.

1. Загальний опис роботи

Основна мета роботи полягає у відокремленні алгоритмів обробки даних від самих даних. У музичному програвачі медіа-бібліотека може містити різнорідні елементи: пісні (Song), подкасти (Podcast) та відеокліпи (VideoClip).

Нам необхідно виконувати над ними різні операції, наприклад:

1. **Експорт у XML** (для збереження списку відтворення).
2. **Розрахунок загального розміру** файлів (щоб знати, скільки місця вони займають на диску).

Якби ми додавали методи toXml() та calculateSize() безпосередньо в класи Song чи Podcast, ми б порушили принцип єдиної відповідальності та засмітили б класи даних зайвою логікою. До того ж, додавання нового формату експорту (наприклад, JSON) змусило б нас змінювати код усіх класів медіа-елементів.

Для вирішення цієї проблеми використаємо патерн **Visitor**. Ми створимо ієрархію "Відвідувачів", де кожен клас (XmlExportVisitor, SizeCalculatorVisitor) містить логіку для певної операції над усіма типами елементів.

Програмна система складається з:

- **Element (MediaElement):** Інтерфейс, що дозволяє "приймати" відвідувача.
- **Concrete Elements:** Класи Song, Podcast, VideoClip.
- **Visitor:** Інтерфейс, що описує методи відвідування для кожного типу елемента.
- **Concrete Visitors:** Класи, що реалізують конкретну логіку (експорт, підрахунок).

2. Опис класів програмної системи

Інтерфейс MediaElement та Класи Даних (Concrete Elements)

Опис:

Ця група класів представляє структуру даних нашої програми.

- **MediaElement** — це спільний інтерфейс для всіх об'єктів медіатеки. Він оголошує єдиний метод accept(Visitor v), який є точкою входу для будь-якого зовнішнього алгоритму (відвідувача).
- **Song, Podcast, VideoClip** — це конкретні класи, що зберігають інформацію. Головна особливість патерну в тому, що ці класи не містять бізнес-логіки обробки даних (наприклад, логіки експорту в XML чи стиснення). Вони діють як носії даних (Data Transfer Objects) з гетерами, плюс механізм маршрутизації викликів через метод accept.

```
public interface MediaElement {
    1 usage  3 implementations
    void accept(Visitor v);
}
```

Рисунок 1 – Код класу MediaElement

```java
public class Song implements MediaElement {
    2 usages
    private String title;
    2 usages
    private String artist;
    2 usages
    private double sizeMb;

    2 usages
    public Song(String title, String artist, double sizeMb) {
        this.title = title;
        this.artist = artist;
        this.sizeMb = sizeMb;
    }

    1 usage
    public String getTitle() { return title; }
    1 usage
    public String getArtist() { return artist; }
    1 usage
    public double getSizeMb() { return sizeMb; }

    1 usage
    @Override
    public void accept(Visitor v) {
        v.visitSong( s: this);
    }
}
```

Рисунок 2 – Код класу Song

```java
public class Podcast implements MediaElement {
    2 usages
    private String title;
    2 usages
    private int episodeNr;
    2 usages
    private double sizeMb;

    1 usage
    public Podcast(String title, int episodeNr, double sizeMb) {
        this.title = title;
        this.episodeNr = episodeNr;
        this.sizeMb = sizeMb;
    }

    1 usage
    public String getTitle() { return title; }
    1 usage
    public int getEpisodeNr() { return episodeNr; }
    1 usage
    public double getSizeMb() { return sizeMb; }

    1 usage
    @Override
    public void accept(Visitor v) {
        v.visitPodcast( p: this);
    }
}
```

Рисунок 3 – Код класу Podcast

```java
public class VideoClip implements MediaElement {
    2 usages
    private String title;
    2 usages
    private String resolution;
    2 usages
    private double sizeMb;

    1 usage
    public VideoClip(String title, String resolution, double sizeMb) {
        this.title = title;
        this.resolution = resolution;
        this.sizeMb = sizeMb;
    }

    1 usage
    public String getTitle() { return title; }
    1 usage
    public String getResolution() { return resolution; }
    1 usage
    public double getSizeMb() { return sizeMb; }

    1 usage
    @Override
    public void accept(Visitor v) {
        v.visitVideoClip( v: this);
    }
}
```

Рисунок 4 – Код класу VideoClip

Інтерфейс Visitor

Опис:

Інтерфейс Visitor визначає контракт для всіх можливих операцій, які можуть бути виконані над елементами медіатеки. Він оголошує набір методів visit(), де кожен метод відповідає певному класу конкретного елемента.

Характеристики:

- Це вузьке місце патерну. Якщо ми додамо новий тип елемента (наприклад, AudioBook), нам доведеться змінити цей інтерфейс, що призведе до помилок компіляції у всіх існуючих відвідувачів.
- Інтерфейс гарантує, що будь-який алгоритм (відвідувач) знає, як обробити кожен тип даних.

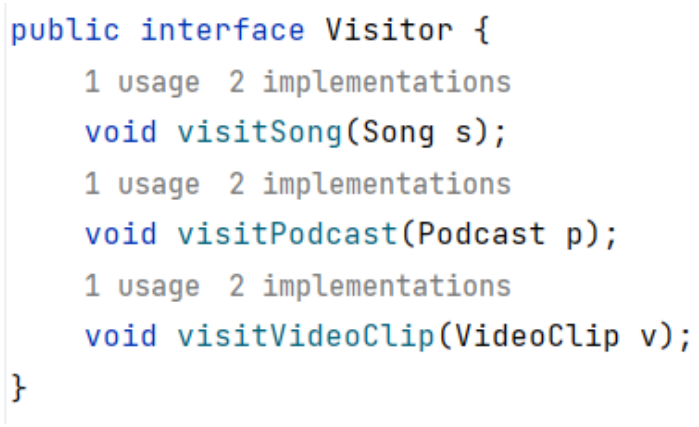


Рисунок 5 – Код інтерфейсу Visitor

Класи XmlExportVisitor та SizeCalculatorVisitor (Concrete Visitors)

Опис:

Це класи, де інкапсульована реальна бізнес-логіка. Замість того, щоб розширювати код експорту XML по класах Song, Podcast і VideoClip, ми збираємо його в одному місці - XmlExportVisitor.

1. XmlExportVisitor:

Відповідає за представлення даних у форматі XML.

- Кожен метод visit знає специфіку полів конкретного об'єкта (наприклад, для подкасту виводиться тег <episode>, а для відео - <resolution>).
- Цей клас не зберігає стан між викликами, він просто виконує дію (вивід у консоль).

2. SizeCalculatorVisitor:

Відповідає за підрахунок статистики (загального розміру файлів).

- Накопичення стану: На відміну від XML-відвідувача, цей клас має внутрішнє поле totalSize.
- Кожен виклик visit додає розмір поточного елемента до загальної суми.
- Це демонструє здатність відвідувача збирати зведений результат під час обходу складної структури.

```java
public class XmlExportVisitor implements Visitor {
    1 usage
    @Override
    public void visitSong(Song s) {
        System.out.println("  <song>");
        System.out.println("    <title>" + s.getTitle() + "</title>");
        System.out.println("    <artist>" + s.getArtist() + "</artist>");
        System.out.println("  </song>");
    }

    1 usage
    @Override
    public void visitPodcast(Podcast p) {
        System.out.println("  <podcast>");
        System.out.println("    <title>" + p.getTitle() + "</title>");
        System.out.println("    <episode>" + p.getEpisodeNr() + "</episode>");
        System.out.println("  </podcast>");
    }

    1 usage
    @Override
    public void visitVideoClip(VideoClip v) {
        System.out.println("  <video>");
        System.out.println("    <title>" + v.getTitle() + "</title>");
        System.out.println("    <resolution>" + v.getResolution() + "</resolution>");
        System.out.println("  </video>");
    }
}
```

Рисунок 6 – Код класу XmlExportVisitor

```java
public class SizeCalculatorVisitor implements Visitor {
    4 usages
    private double totalSize = 0;

    1 usage
    @Override
    public void visitSong(Song s) {
        totalSize += s.getSizeMb();
    }

    1 usage
    @Override
    public void visitPodcast(Podcast p) {
        totalSize += p.getSizeMb();
    }

    1 usage
    @Override
    public void visitVideoClip(VideoClip v) {
        totalSize += v.getSizeMb();
    }

    1 usage
    public double getTotalSize() {
        return totalSize;
    }
}
```

Рисунок 7 – Код класу SizeCalculatorVisitor

Клас MediaLibrary (Object Structure)

Опис:

Клас MediaLibrary виконує роль контейнера або Структури Об'єктів (Object Structure). У складніших системах це могло б бути дерево (шаблон Composite), але тут це список. Його завдання - зберігати посилання на всі елементи та надавати спосіб для відвідувача обійти їх.

Характеристики:

- Зберігає колекцію List<MediaElement>. Список поліморфний - там лежать і пісні, і відео.

- Метод accept(Visitor v): Це ітератор для відвідувача. Він проходить по списку і для кожного елемента викликає його метод accept. Таким чином, клієнту не потрібно писати цикли перебору в основному коді.

```
public class MediaLibrary {
    2 usages
    private List<MediaElement> elements = new ArrayList<>();

    4 usages
    public void add(MediaElement e) {
        elements.add(e);
    }

    2 usages
    public void accept(Visitor v) {
        for (MediaElement element : elements) {
            element.accept(v);
        }
    }
}
```

Рисунок 8 – Код класу MediaLibrary

Клас VisitorClient (Client)

Опис:

Клас Клієнта поєднує всі частини системи. Він створює структуру даних (бібліотеку), наповнює її різними об'єктами, а потім застосовує до цієї структури різні алгоритми (відвідувачів).

Етапи роботи:

1. Створюється бібліотека, куди додаються пісні, подкасти, відео.
2. Створюється екземпляр XmlExportVisitor. Викликається library.accept(exportVisitor). Результат – в консолі з’являється XML.
3. Створюється екземпляр SizeCalculatorVisitor. Викликається library.accept(calcVisitor). Відвідувач проходить по списку, накопичує суму.
4. Клієнт запитує результат у відвідувача: calc.getTotalSize().

```java
public class VisitorClient {

    public static void main(String[] args) {
        MediaLibrary library = new MediaLibrary();
        library.add(new Song( title: "Bohemian Rhapsody",  artist: "Queen",  sizeMb: 5.5));
        library.add(new Podcast( title: "Tech Talk",  episodeNr: 42,  sizeMb: 45.0));
        library.add(new VideoClip( title: "Thriller",  resolution: "1080p",  sizeMb: 120.0));
        library.add(new Song( title: "Imagine",  artist: "John Lennon",  sizeMb: 3.2));

        System.out.println("- Exporting to XML -");
        Visitor exportVisitor = new XmlExportVisitor();
        System.out.println("<library>");
        library.accept(exportVisitor);
        System.out.println("</library>");

        System.out.println("\n- Calculating Statistics -");
        SizeCalculatorVisitor calcVisitor = new SizeCalculatorVisitor();
        library.accept(calcVisitor);
        System.out.println("Total library size: " + calcVisitor.getTotalSize() + " MB");
    }

}
```

Рисунок 9 – Код класу VisitorClient

3. Опис результатів коду

Після запуску програми консоль відображає результати роботи двох різних алгоритмів над одним і тим самим набором даних.

```
- Exporting to XML -
<library>
  <song>
    <title>Bohemian Rhapsody</title>
    <artist>Queen</artist>
  </song>
  <podcast>
    <title>Tech Talk</title>
    <episode>42</episode>
  </podcast>
  <video>
    <title>Thriller</title>
    <resolution>1080p</resolution>
  </video>
  <song>
    <title>Imagine</title>
    <artist>John Lennon</artist>
  </song>
</library>

- Calculating Statistics -
Total library size: 173.7 MB
```

Рисунок 10 – Результат виконання програми

Аналіз результатів:

1. Спершу спрацював XmlExportVisitor. Він пройшов по кожному елементу і, залежно від типу, вивів відповідні теги (<song>, <podcast>, <video>).
2. Потім спрацював SizeCalculatorVisitor. Він просумував розміри всіх файлів

Отже, ми змогли додати дві абсолютно різні операції (експорт тексту та математичний розрахунок), не змінюючи жодного рядка коду в класах Song, Podcast чи VideoClip.

4. Діаграми класів

Для візуалізації архітектури розробленої системи та взаємозв'язків між її компонентами побудуємо UML-діаграми класів.

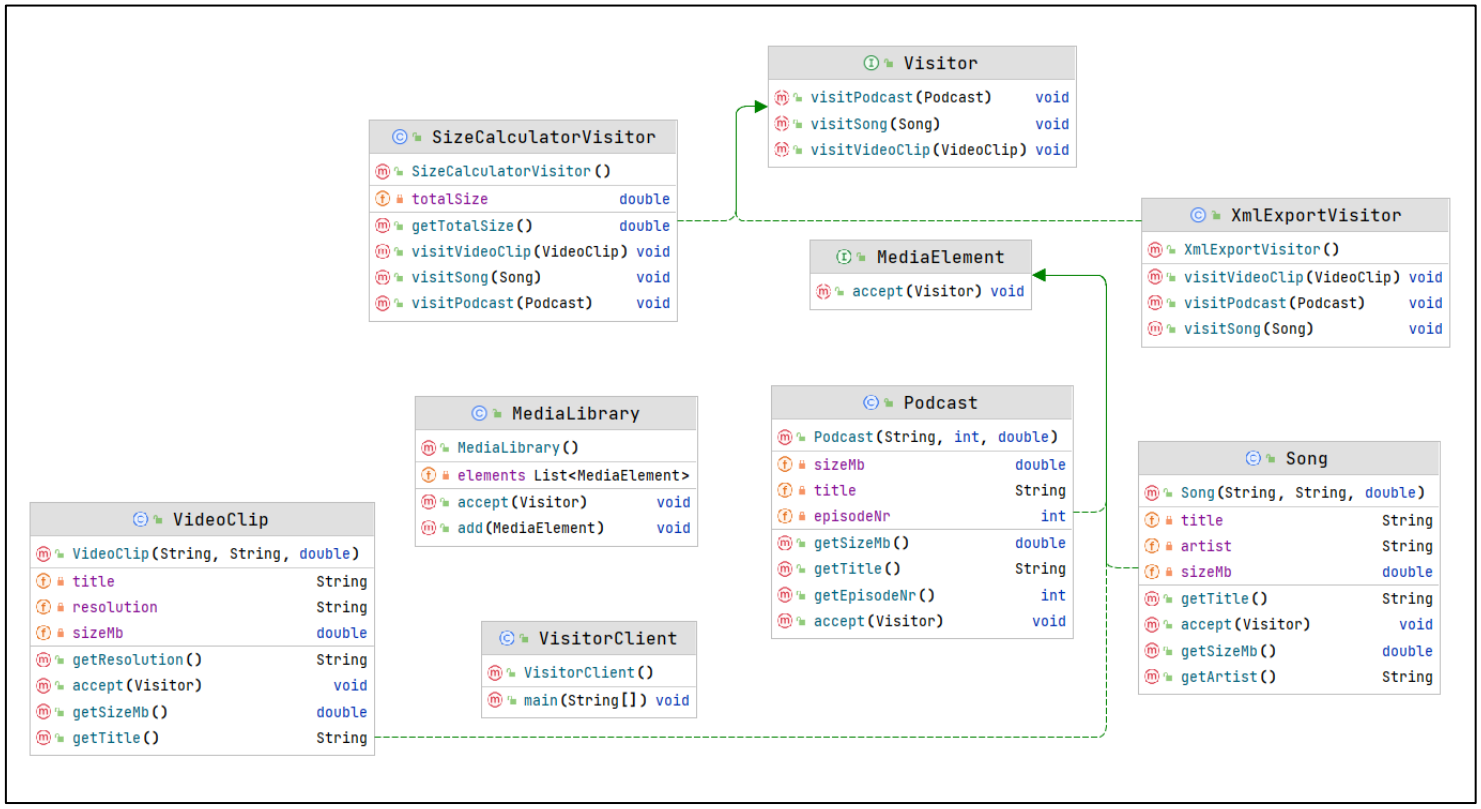


Рисунок 11 – Діаграма класів (спрощена

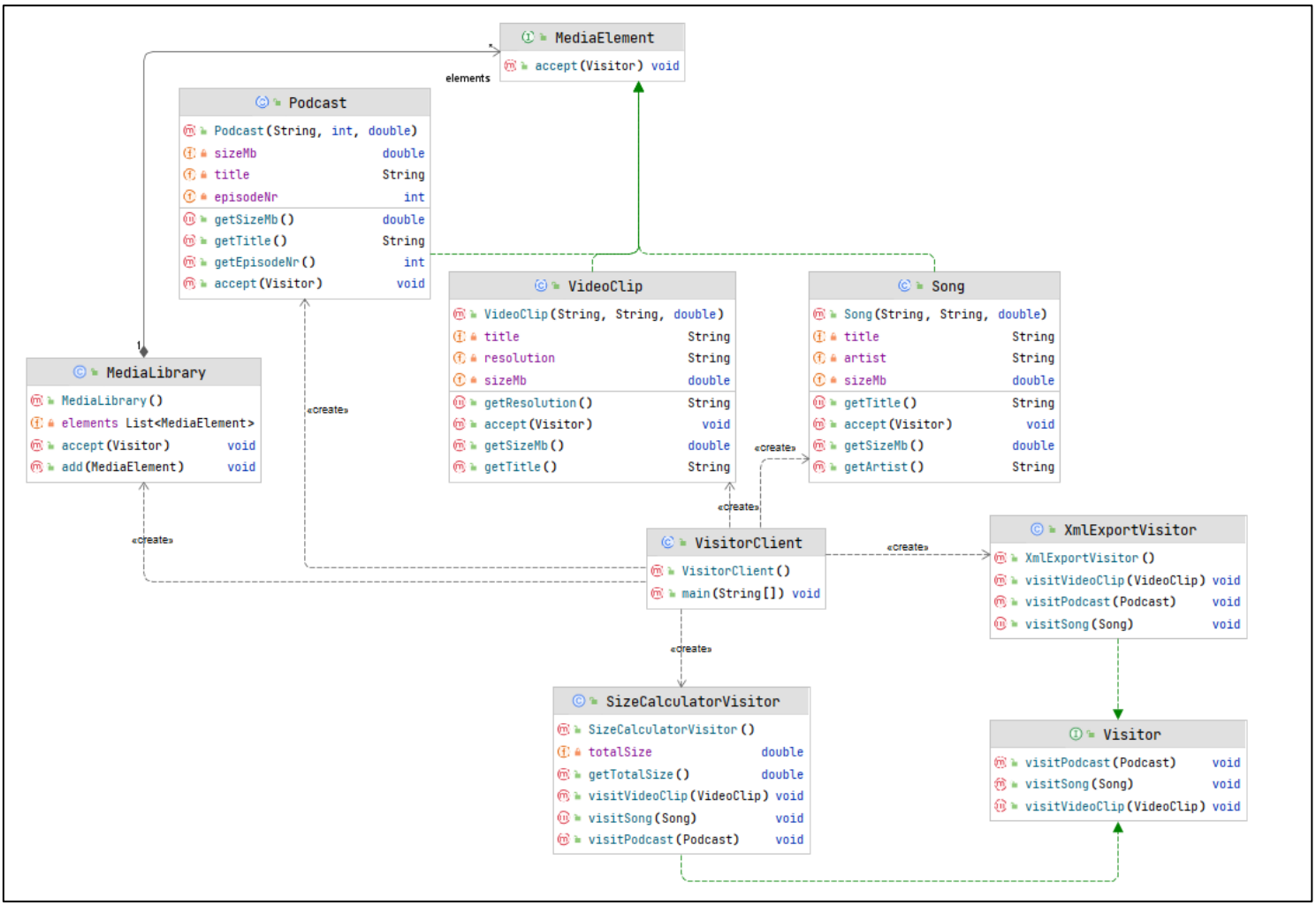


Рисунок 12 – Діаграма класів (повна)

Висновки

В ході лабораторної роботи було реалізовано патерн «Відвідувач» (Visitor) для системи музичної бібліотеки. Цей патерн дозволив винести операції експорту даних та збору статистики в окремі класи-відвідувачі. Основна перевага, яку ми отримали - дотримання принципу відкритості/закритості: ми можемо додавати необмежену кількість нових операцій, створюючи нові класи відвідувачів, і при цьому не потрібно змінювати код існуючих класів медіа-елементів.

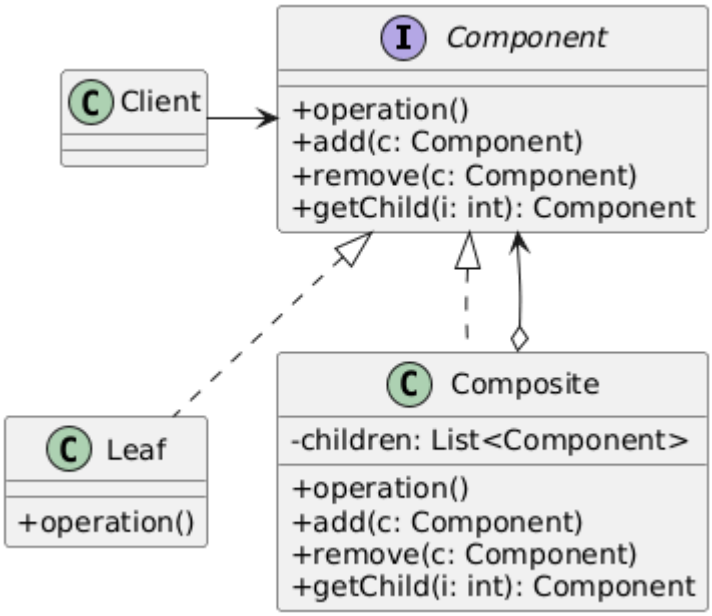
Недоліком, який було виявлено теоретично, є складність додавання нових типів елементів (наприклад, якщо з'явиться AudioBook). У такому разі доведеться змінювати інтерфейс Visitor і всі його реалізації, додаючи метод visitAudioBook. Тому цей патерн найкраще підходить для систем зі стабільною структурою класів, але часто змінюваними операціями над ними.

Контрольні запитання

1. Яке призначення шаблону «Композит»?

Шаблон «Композит» (або «Компонувальник») дозволяє згрупувати об'єкти у деревоподібну структуру для представлення ієрархій «частина-ціле». Цей патерн дозволяє клієнтам виконувати операції над окремими об'єктами та їхніми колекціями (групами) однаковим чином. (Приклад: файли та папки).

2. Нарисуйте структуру шаблону «Композит».



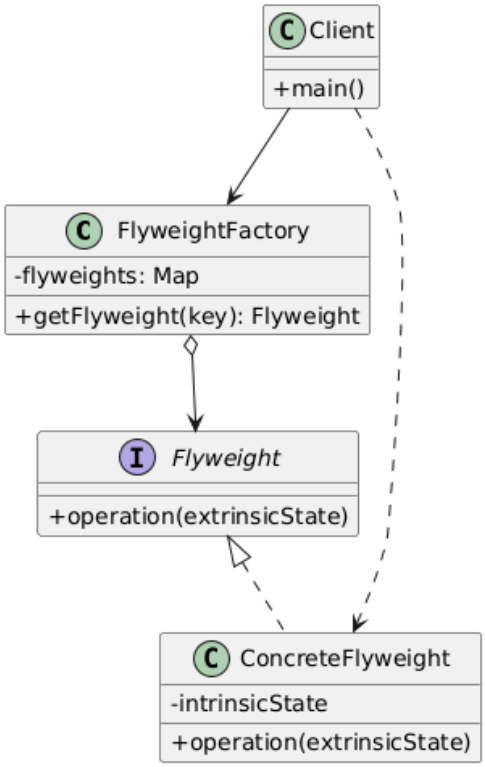
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

- **Component (Компонент):** Спільний інтерфейс (або абстрактний клас) для всіх елементів дерева (і простих, і складних). Оголошує методи бізнес-логіки та методи управління дочірніми елементами.
- **Leaf (Лист):** Простий елемент дерева, який не має дочірніх елементів. Реалізує основну поведінку.
- **Composite (Контейнер/Композит):** Складний елемент, що містить дочірні елементи (які можуть бути як Листами, так і іншими Композитами). Реалізує методи управління дітьми (add, remove).
- **Взаємодія:** Метод operation() у Композита зазвичай перебирає всіх своїх нащадків і викликає operation() у кожного з них, сумуючи або об'єднуючи результати. Клієнт працює з усіма елементами через інтерфейс Component.

4. Яке призначення шаблону «Легковаговик»?

Шаблон «Легковаговик» (або «Пристосуванець») призначений для ефективної підтримки великої кількості дрібних об'єктів. Він дозволяє заощаджувати оперативну пам'ять, розділяючи спільний стан об'єктів (внутрішній стан) і зберігаючи його в одному екземплярі, замість того щоб зберігати однакові дані в кожному об'єкті.

5. Нарисуйте структуру шаблону «Легковаговик».



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

- **Flyweight (Легковаговик):** Інтерфейс, через який легковаговики можуть отримувати зовнішній стан і діяти відповідно до нього.
- **ConcreteFlyweight:** Реалізує інтерфейс і зберігає внутрішній стан (intrinsic state), який є спільним і незмінним.
- **FlyweightFactory:** Створює та управляє пулом легковаговиків. Якщо клієнт просить легковаговика, якого ще немає, фабрика створює його; якщо він вже є - повертає існуючий.
- **Client:** Зберігає зовнішній стан (extrinsic state) - унікальні дані для кожного контексту (наприклад, координати об'єкта) і передає їх легковаговику під час виклику методу.

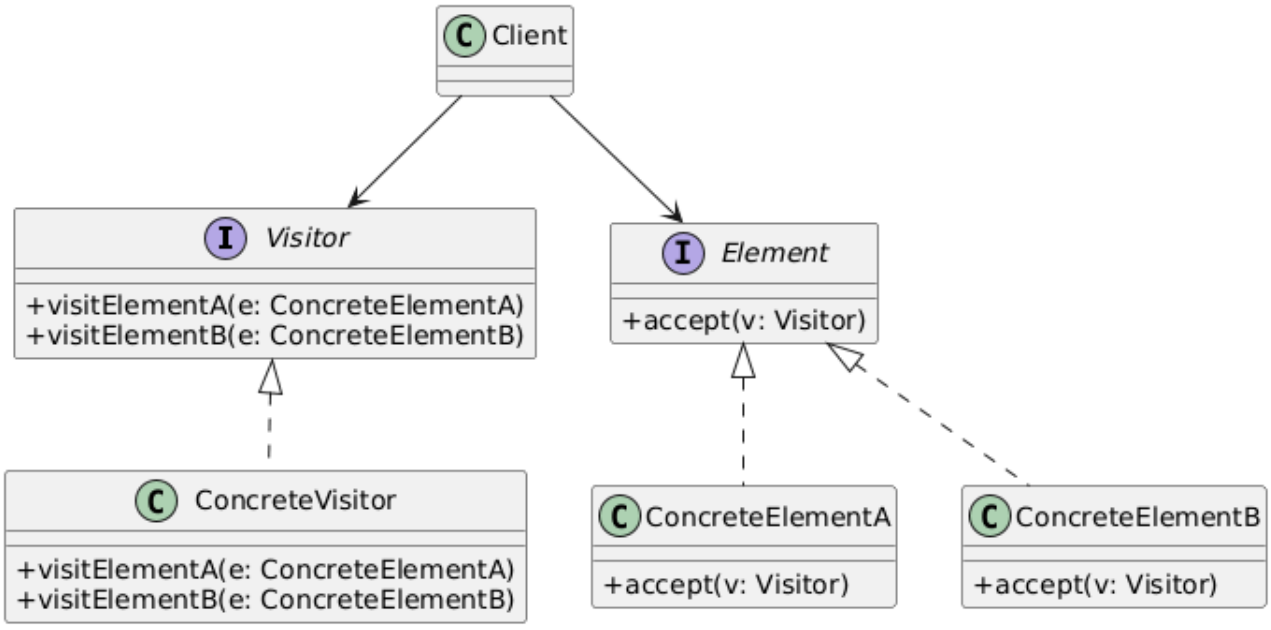
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Інтерпретатор» (Interpreter) використовується для визначення граматики простої мови та інтерпретації речень цієї мови. Він будує представлення речень у вигляді абстрактного синтаксичного дерева (AST), де кожен вузол є виразом (Expression), який реалізує метод interpret(). Застосовується для SQL-подібних запитів, математичних виразів, регулярних виразів тощо.

8. Яке призначення шаблону «Відвідувач»?

Шаблон «Відвідувач» (Visitor) дозволяє додавати нові операції до ієрархії класів без зміни коду цих класів. Він відокремлює алгоритм від структури об'єкта, дозволяючи виконувати різні дії над об'єктами в залежності від їх конкретного типу, використовуючи механізм подвійної диспетчеризації.

9. Нарисуйте структуру шаблону «Відвідувач».



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

- **Visitor (Відвідувач):** Інтерфейс, що оголошує методи visit для кожного класу конкретного елемента.
- **ConcreteVisitor:** Реалізує специфічну поведінку (алгоритм) для кожного типу елемента.
- **Element (Елемент):** Інтерфейс структури об'єктів, що оголошує метод accept(Visitor).
- **ConcreteElement:** Реалізує метод accept, в якому викликає відповідний метод відвідувача (наприклад, visitor.visitElementA(this)).
- **ObjectStructure (Структура об'єктів):** Колекція або складний об'єкт, що містить Елементи та дозволяє відвідувачу обійти їх.

Взаємодія: Клієнт створює Відвідувача і передає його Елементам. Елемент "приймає" Відвідувача і викликає у нього метод, що відповідає своєму класу (this). Так Відвідувач дізнається тип Елемента і виконує потрібну дію.