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 розроблення програмного забезпечення
«Патерни проектування.»

Виконав:

Студент групи ІА-32

Костінський Макарій

Перевірив:

Мягкий Михайло Юрійович

Зміст

Тема: Патерни проектування.

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

Теоретичні відомості

Компонувальник (Composite): Структурний патерн, який дозволяє згрупувати об'єкти у деревоподібну структуру для представлення ієрархій "частина-ціле". Дозволяє клієнтам однаково працювати як з окремими об'єктами, так і з групами об'єктів .

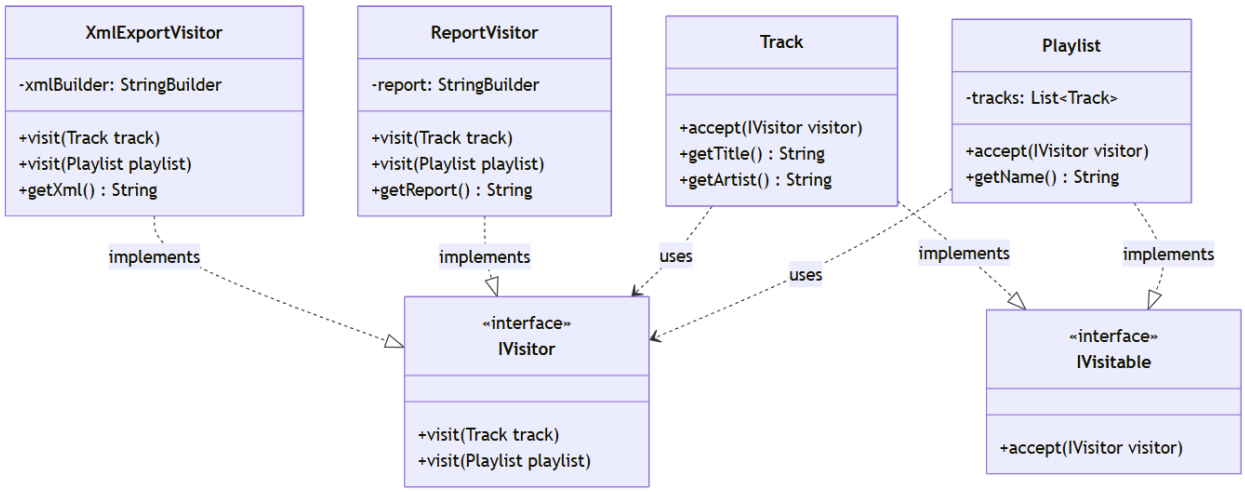
Легковаговик (Flyweight): Структурний патерн, який використовується для зменшення кількості об'єктів у програмі шляхом поділу спільних частин стану між множиною об'єктів. Це дозволяє економити оперативну пам'ять .

Інтерпретатор (Interpreter): Поведінковий патерн, що використовується для визначення граматики простої мови та інтерпретації речень цієї мови.

Відвідувач (Visitor): Поведінковий патерн, який дозволяє додавати нові операції над об'єктами, не змінюючи класи цих об'єктів. Він дозволяє відокремити алгоритм від структури об'єкта, яким він оперує.

Хід роботи

Діаграма класів реалізації патерну «Visitor»



Для реалізації функціоналу експорту даних (XML) та генерації звітів у системі "Online Radio Station" було обрано патерн Visitor. Це дозволило винести логіку обходу та обробки елементів медіатеки (треків та плейлистів) у окремі класи, не перевантажуючи моделі даних зайвими методами.

Фрагменти коду по реалізації шаблону

Інтерфейси IVisitor та IVisitable: Визначають контракт для відвідувачів та елементів, що відвідуються.

```
package ua.kpi.iasa.onlineradio.models.visitor;

import ua.kpi.iasa.onlineradio.models.Playlist;
import ua.kpi.iasa.onlineradio.models.Track;

public interface IVisitor {
    void visit(Track track);
    void visit(Playlist playlist);
}
```

```
package ua.kpi.iasa.onlineradio.models.visitor;
public interface IVisitable {
    void accept(IVisitor visitor);
}
```

Реалізація у класах даних (Track): Класи лише "приймають" відвідувача, передаючи йому посилання на себе.

```
@Override
public void accept(IVisitor visitor) {
    visitor.visit(this);
}
```

Конкретний відвідувач (XmlExportVisitor): Реалізує логіку експорту в XML, яка раніше могла б знаходитися всередині класів даних.

```
package ua.kpi.iasa.onlineradio.models.visitor;

import ua.kpi.iasa.onlineradio.models.Playlist;
import ua.kpi.iasa.onlineradio.models.Track;

public class XmlExportVisitor implements IVisitor {
    private StringBuilder xmlBuilder = new StringBuilder();

    public String getXml() {
        return xmlBuilder.toString();
    }

    @Override
    public void visit(Playlist playlist) {
        xmlBuilder.append("<playlist
id=\"").append(playlist.getId()).append("\">\n")
                .append("
<name>").append(playlist.getName()).append("</name>\n")
                .append("  <tracks>\n");

    @Override
    public void visit(Track track) {
        xmlBuilder.append("    <track
id=\"").append(track.getId()).append("\">\n")
                .append("
<title>").append(track.getTitle()).append("</title>\n")
                .append("
<artist>").append(track.getArtist()).append("</artist>\n")
                .append("
<duration>").append(track.getDuration().getSeconds()).append("s</duration>\n")
                .append("    </track>\n");
    }

    public void closePlaylist() {
        xmlBuilder.append("  </tracks>\n")
                .append("</playlist>");
    }
}
```

Висновок:

У ході виконання лабораторної роботи було вивчено принципи роботи патернів проєктування: «Composite», «Flyweight», «Interpreter», «Visitor».

Для системи "Online Radio Station" було імплементовано патерн Visitor. Це дало змогу реалізувати функціонал експорту даних у XML та створення текстових звітів, не змінюючи код існуючих класів Track та Playlist. Такий підхід забезпечив дотримання принципу відкритості/закритості (Open/Closed Principle), дозволяючи легко додавати нові формати експорту (наприклад, JSON або CSV) шляхом створення нових класів-відвідувачів, не порушуючи стабільність основної бізнес-логіки.