Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №8
Технологія розробки програмного забезпечення
«Патерни проектування»

Виконала
студентка групи ІА–32:
Ткачук М. С.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

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.

Аудіо редактор (singleton, adapter, observer, mediator, composite, client-server)

Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Теоретичні відомості

Шаблон Composite

Шаблон Composite використовується тоді, коли об'єкти потрібно подати у вигляді деревоподібної структури за принципом «частина цілого». Він дозволяє однаково працювати як із поодинокими елементами, так і з цілими групами елементів, які можуть містити вкладені об'єкти. Це спрощує обробку ієрархій, оскільки клієнтський код не розрізняє, чи має справу з одиночним елементом, чи з композитом. Прикладом може бути форма з графічними елементами, де розтягування застосовується рекурсивно до всіх дочірніх компонентів. Composite зручно використовувати для представлення структур у вигляді дерева або при роботі з рекурсивними операціями. Переваги включають гнучкість, уніфіковану роботу з об'єктами та простоту розширення ієрархії. Недоліки полягають у необхідності створення спільного інтерфейсу та складності первинного впровадження.

Шаблон Flyweight

Шаблон Flyweight призначений для зменшення кількості однотипних об'єктів у пам'яті шляхом винесення повторюваних даних у спільні елементи. Він розділяє стан об'єкта на внутрішній та зовнішній. Внутрішній стан є незмінним і належить поділюваному об'єкту, а зовнішній зберігається в контексті використання. Це дозволяє мати один фізичний об'єкт і багато посилань на нього, що суттєво економить пам'ять. Flyweight доречно використовувати тоді, коли в системі існує велика кількість однакових елементів, наприклад, відображення великого тексту або множини графічних примітивів. Перевагами є економія пам’яті і можливість оптимізації великих структур, а недоліками є додаткові витрати часу на пошук або формування зовнішнього стану та збільшення кількості службових класів.

Шаблон Interpreter

Шаблон Interpreter застосовується для опису граматики та реалізації інтерпретатора для певної мови або формату. Граматика подається у вигляді абстрактного синтаксичного дерева, що складається з термінальних і нетермінальних виразів. Кожен вираз має власний метод інтерпретації, а нетермінальні вирази додатково інтерпретують свої дочірні елементи. Interpreter добре підходить для невеликих мов або специфічних форматів, де правила граматики часто змінюються або розширюються. Він забезпечує простоту додавання нових конструкцій, але при великій кількості правил кількість класів зростає, що ускладнює підтримку. Шаблон зручний для задач пошуку, побудови виразів, створення фільтрів та скриптових mini-мов.

Шаблон Visitor

Шаблон Visitor дозволяє визначати нові операції над об'єктами, не змінюючи їхніх класів. Він розділяє дані та поведінку: структура об'єктів залишається незмінною, а логіка обробки поміщається у відвідувачів. Це корисно тоді, коли існує багато різних операцій, які потрібно застосовувати до об'єктів з однієї ієрархії. Visitor добре підходить для розрахунків, генерації звітів, аналізу об’єктів або трансформацій. Основною перевагою є можливість легко додавати нові операції, не змінюючи структуру об’єктів. Недолік полягає в тому, що додавання нового типу елемента потребує зміни у всіх відвідувачах, що збільшує витрати на підтримку.

Хід Роботи

Діаграма класів

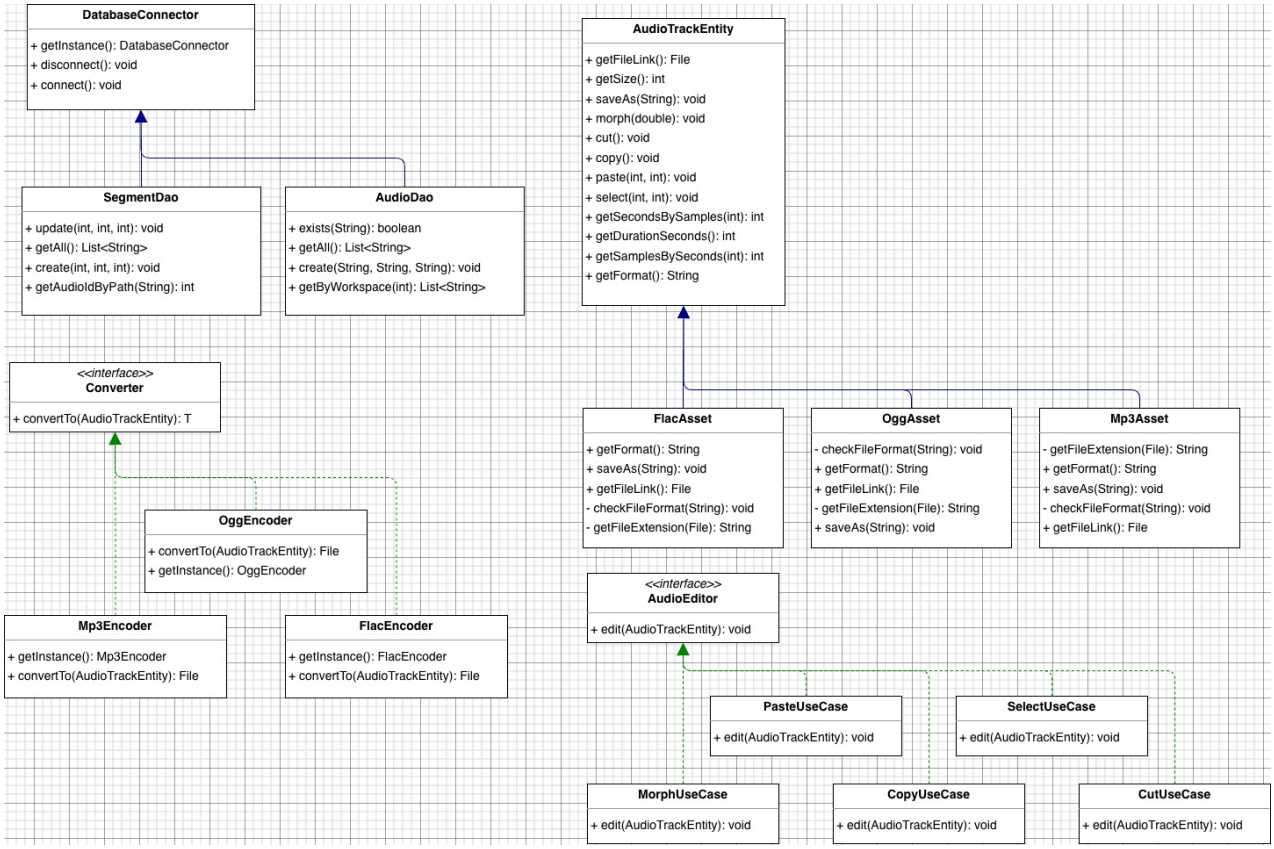


Рисунок 1 – Діаграма класів

На рисунку 1 накведено діаграму класів аудіоредактора, що відображає логічну структуру програмного забезпечення, основні класи, їхні атрибути, методи та взаємозв’язки між ними.

Центральним елементом системи є клас AudioTrackEntity, який описує сутність аудіотреку та містить методи для виконання основних операцій редагування cut(), copy(), paste(), morph(), а також методи для роботи з форматом файлу, отримання семплів і тривалості (getDurationSeconds(), getFormat(), getSampleBySeconds()).

Цей клас є базовим для роботи з різними типами звукових файлів і виступає ядром усієї системи.

Від AudioTrackEntity наслідуються класи Mp3Asset, OggAsset та FlacAsset, що реалізують підтримку різних аудіоформатів. Кожен із цих класів має власні методи для збереження (saveAs()), перевірки формату (checkFileFormat()), а також отримання розширення файлу (getFileExtension()), що дозволяє системі працювати з популярними форматами MP3, OGG та FLAC.

Окремий клас AudioEditor реалізує високорівневу логіку редагування файлів. Він містить метод edit(), який викликає відповідні операції залежно від сценарію редагування: копіювання, вставлення, вирізання або деформації звуку. Для реалізації цих сценаріїв використовуються окремі класи CutUseCase, CopyUseCase, SelectUseCase і MorphUseCase, кожен із яких виконує власну операцію редагування через метод edit(AudioTrackEntity).

Для кодування звукових доріжок у різні формати використовується абстрактний клас Converter<T>, який містить універсальний метод convertTo(AudioTrackEntity). Від нього наслідуються конкретні реалізації Mp3Encoder, OggEncoder та FlacEncoder, які забезпечують перетворення аудіо у відповідний формат.

Компонент DatabaseConnector відповідає за взаємодію з локальною базою даних SQLite. Він реалізує підключення, відключення та отримання екземпляра підключення (getInstance(), connect(), disconnect()). З ним працюють два DAO-класи AudioDao і SegmentDao, які реалізують доступ до даних аудіофайлів і сегментів.
AudioDao забезпечує методи create(), getAll(), exists() і getByWorkspace(), а SegmentDao: операції create(), update() та getAudioIdByPath(). Це дає змогу системі зберігати інформацію про проєкти, сегменти та аудіофайли у структурованому вигляді.

Діаграма класів демонструє, як система аудіоредактора розділена на логічні шари:

- рівень даних (DAO-класи, DatabaseConnector),
- рівень бізнес-логіки (AudioEditor, UseCase-класи),
- рівень моделей (AudioTrackEntity і похідні класи для форматів),
- рівень перетворень (Converter і Encoder-класи).

Патерн composite

Патерн Composite дозволяє створювати ієрархічні структури з об’єктів, у яких окремі елементи і групи елементів обробляються однаково. Усі компоненти реалізують спільний інтерфейс UiNode з методом operate, тому клієнтський код може взаємодіяти з будь-яким елементом незалежно від його складності.

UiLeaf представляє простий елемент інтерфейсу, який виконує дію без дочірніх компонентів. UiGroup є складеним елементом, що містить інші вузли і дозволяє додавати та видаляти їх за допомогою методів add та remove. Виклик operate для UiGroup запускає виконання цієї операції у всіх вкладених елементах, що дозволяє будувати інтерфейс у вигляді дерева.

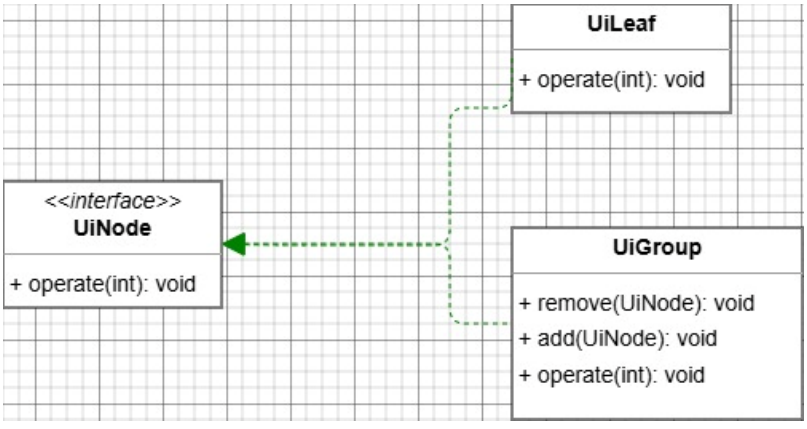


Рисунок 2 – Схема патерну composite

Фрагменти коду:

CompositeDemo.java

```
package app.soundlab.ui;

public class CompositeDemo {
    public static void main(String[] args) {
        UiGroup root = new UiGroup("Studio Dashboard");

        UiGroup toolbar = new UiGroup("Toolbar");
        toolbar.add(new UiLeaf("Open Button"));
        toolbar.add(new UiLeaf("Save Button"));
        toolbar.add(new UiLeaf("Export Button"));

        UiGroup trackPanel = new UiGroup("Track Panel");
        UiGroup trackOne = new UiGroup("Track #1");
        trackOne.add(new UiLeaf("Waveform View"));
        trackOne.add(new UiLeaf("Volume Slider"));
        UiGroup trackTwo = new UiGroup("Track #2");
        trackTwo.add(new UiLeaf("Waveform View"));
        trackTwo.add(new UiLeaf("Volume Slider"));
        UiGroup fxControls = new UiGroup("FX Controls");
        fxControls.add(new UiLeaf("Reverb Toggle"));
        fxControls.add(new UiLeaf("Delay Toggle"));
        trackPanel.add(trackOne);
        trackPanel.add(trackTwo);
        trackPanel.add(fxControls);

        UiGroup statusPanel = new UiGroup("Status Panel");
```

```
        statusPanel.add(new UiLeaf("CPU Meter"));
        statusPanel.add(new UiLeaf("Disk Meter"));
        statusPanel.add(new UiLeaf("Project Clock"));

        root.add(toolbar);
        root.add(trackPanel);
        root.add(statusPanel);

        root.operate();
    }
}
```

UiGroup.java

```
package app.soundlab.ui;

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

public class UiGroup implements UiNode {
    private final String name;
    private final List<UiNode> children = new ArrayList<>();

    public UiGroup(String name) {
        this.name = name;
    }

    public void add(UiNode node) {
        children.add(node);
    }
```

```
    public void remove(UiNode node) {
        children.remove(node);
    }

    @Override
    public void operate(int depth) {
        System.out.println("  ".repeat(depth) + "+ " + name);
        for (UiNode child : children) {
            child.operate(depth + 1);
        }
    }
}
```

UiLeaf.java

```
package app.soundlab.ui;

public class UiLeaf implements UiNode {
    private final String name;

    public UiLeaf(String name) {
        this.name = name;
    }

    @Override
    public void operate(int depth) {
        System.out.println("  ".repeat(depth) + "- " + name);
    }
}
```

UiNode.java

```
package app.soundlab.ui;

public interface UiNode {
    void operate(int depth);

    default void operate() {
        operate(0);
    }
}
```

Вихідний код:
https://github.com/mandarinchik21/AudioEditor/tree/main/lab8

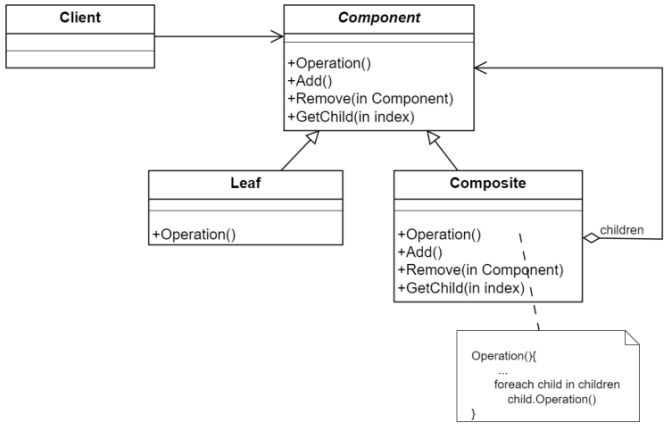
Висновок: У ході виконання роботи були розглянуті патерни Composite, Flyweight, Interpreter та Visitor, а також проаналізовано їх практичне застосування у розробці аудіоредактора. Використання шаблону Composite дозволило оптимізувати структуру системи, спростити роботу з ієрархічними елементами інтерфейсу, зменшити кількість об’єктів у пам’яті, описати складні операції через формальні граматики та відокремити логіку обробки від структури даних. Реалізація одного з патернів у проєкті показала, що вони підвищують гнучкість, масштабованість та зручність підтримки програмної системи, забезпечуючи більш чисту та передбачувану архітектуру.

Контрольні запитання

1. Яке призначення шаблону «Композит»?

Композит використовується для подання об’єктів у вигляді деревоподібної структури «частина цілого» та дозволяє однаково обробляти як окремі елементи, так і групи елементів.

2. Нарисуйте структуру шаблону «Композит».



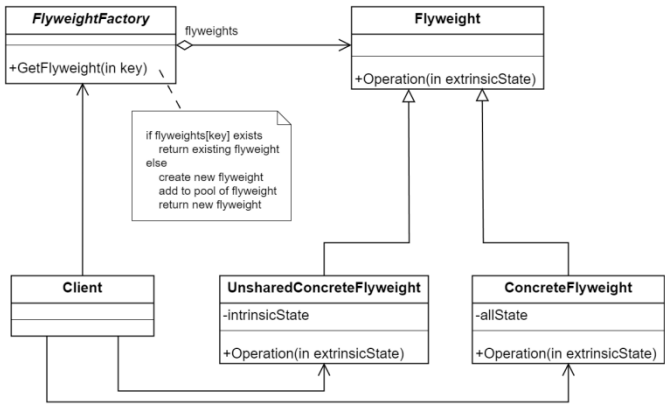
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

- Component спільний інтерфейс для всіх елементів дерева
- Leaf кінцевий елемент без дочірніх об’єктів
- Composite вузол, що містить колекцію Component та делегує їм виклики
- Client працює з об’єктами через інтерфейс Component, не розрізняючи Leaf і Composite

Composite зберігає список дочірніх Component і при виклику операцій обходить та викликає їх для кожного елемента.

4. Яке призначення шаблону «Легковаговик» (Flyweight)?
   Легковаговик зменшує кількість об’єктів у системі, розділяючи спільний внутрішній стан між багатьма використаннями об’єкта та виносячи змінний зовнішній стан у контекст.

5. Нарисуйте структуру шаблону «Легковаговик».



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

- Flyweight інтерфейс спільного об’єкта, приймає зовнішній стан як параметр
- ConcreteFlyweight реалізує внутрішній стан, який розділяється між клієнтами
- UnsharedConcreteFlyweight окремі об’єкти, які не розділяються (необов’язково)
- FlyweightFactory створює і кешує ConcreteFlyweight, повертаючи вже існуючі екземпляри

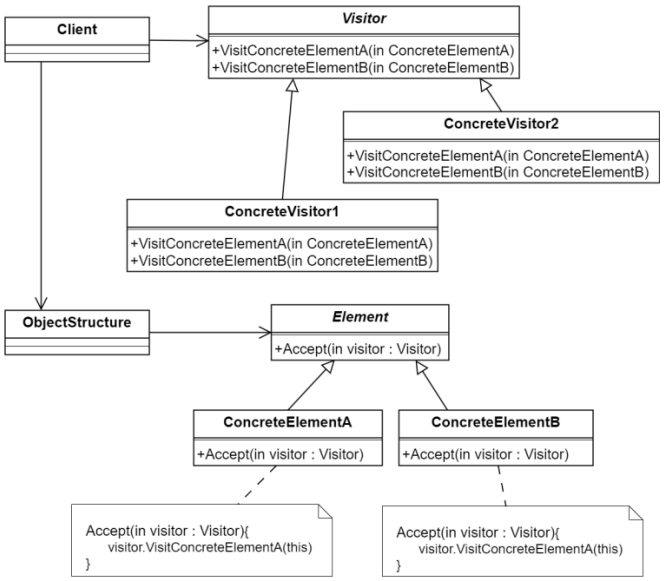
- Client запитує об’єкти у фабрики та передає зовнішній стан при виклику методів

Клієнт не створює об’єкти напряму, а звертається до FlyweightFactory, яка або віддає вже наявний екземпляр, або створює новий.

7. Яке призначення шаблону «Інтерпретатор» (Interpreter)?
   Інтерпретатор описує граматику простої мови у вигляді класів та надає механізм обчислення виразів цієї мови через абстрактне синтаксичне дерево. Використовується для побудови невеликих скриптових мов, фільтрів, правил пошуку.

8. Яке призначення шаблону «Відвідувач» (Visitor)?
   Відвідувач дозволяє додавати нові операції над елементами складної структури, не змінюючи класи самих елементів. Логіка обробки виноситься у окремі об’єкти Visitor, а структура даних залишається сталою.

9. Нарисуйте структуру шаблону «Відвідувач».



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

- Visitor інтерфейс відвідувача з методами visitX для кожного типу елемента
- ConcreteVisitor конкретні відвідувачі з реалізацією логіки для кожного типу
- Element інтерфейс елемента з методом accept(Visitor)
- ConcreteElement конкретні елементи структури, які викликають visitor.visitConcreteElementX(this) у методі accept
- Client створює елементи та відвідувачів, ініціює обхід

Клієнт передає Visitor елементам. Кожен ConcreteElement викликає відповідний метод у Visitor, тим самим передаючи себе на обробку.