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

із дисципліни «**Технології розроблення програмного забезпечення**»

“ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»”

Варіант №5. Аудіоредактор

Виконав
студент групи ІА–24
Криворучек В. С.

Перевірив
викладач
Мягкий М. Ю.

Київ 2024

Зміст

Мета: Вивчення основних принципів застосування шаблонів «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD» при створенні проєкту за індивідуальним варіантом

Тема (Варіант №5)

5 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).

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Теоретичні відомості

Для розробки масштабних систем із великою кількістю користувачів і пов'язаних програм часто використовують шаблони роботи з базами даних. Вони забезпечують ефективний доступ до даних, зменшуючи складність і підвищуючи продуктивність при зростанні обсягів інформації.

Active Record. Цей шаблон передбачає, що кожен об'єкт відповідає за дані та поведінку, пов'язані з одним рядком у базі даних. Усі запити до БД і логіка обробки даних розміщуються безпосередньо у класі сутності. Active Record підходить для простих систем, таких як Ruby on Rails, але з ростом складності логіку часто переносять у спеціалізовані об'єкти.

Table Data Gateway. Цей шаблон виділяє окремий клас для взаємодії з базою даних для кожного типу даних. У цьому класі містяться всі запити до БД, включаючи операції збереження та видалення. Завдяки цьому забезпечується більша гнучкість, легкість тестування і розділення даних від логіки доступу до них. Щоб уникнути дублювання коду, загальні методи часто переносять у базовий клас. Цей підхід також відомий як "репозиторій".

Data Mapper. Шаблон Data Mapper відповідає за перетворення об'єктів програми у формат бази даних або мережі. Він містить інформацію про зв’язок між властивостями об'єкта і стовпцями таблиці, забезпечуючи коректне відображення даних. Mapper дозволяє розв’язувати невідповідності між типами даних об'єкта та БД, полегшуючи інтеграцію складних систем.

Шаблони роботи з об'єктами

Composite створює деревоподібну структуру для представлення ієрархій типу "частина-ціле". Це дозволяє однаково обробляти окремі об'єкти та складені структури. Наприклад, у формі можуть бути поля для введення, написи чи інші вкладені елементи. Усі вони уніфіковано обробляються, наприклад, при зміні розміру форми. Composite ефективний для роботи з деревами, наприклад, замовленнями, що включають продукти й коробки. Рішення полягає у

використанні єдиного інтерфейсу з методом для обчислення вартості всього дерева.

Flyweight. Шаблон Flyweight допомагає зменшити кількість об'єктів у додатку шляхом спільного використання загальних даних між різними частинами програми. Наприклад, у тексті кожна буква може здаватися окремим об'єктом, але фактично це спільний об'єкт із посиланнями. Flyweight розділяє дані на внутрішній стан (загальний для об'єктів) і зовнішній (унікальний для кожного контексту). Це особливо корисно у додатках з великою кількістю однотипних об'єктів, таких як графічні редактори чи ігри.

Interpreter використовується для роботи з мовами та граматиками. Він забезпечує створення синтаксичного дерева, де кожен термінальний і нетермінальний символ інтерпретується у своєму контексті. Батьківські елементи дерева викликають інтерпретацію дочірніх, забезпечуючи виконання складних операцій. Interpreter підходить для невеликих мов і простих контекстів, але при частих змінах граматики може бути складним.

Visitor дозволяє виконувати операції над об'єктами без зміни їхньої структури. Нові операції додаються шляхом створення окремих класів-відвідувачів. Наприклад, для експорту графу геоданих у формат XML можна створити окремий клас-відвідувач, що виконує відповідну операцію. Однак додавання нових елементів у систему вимагає змін у всіх наявних відвідувачах, що може ускладнювати підтримку.

Використання шаблонів роботи з базами даних і об'єктами дозволяє ефективно організувати складні системи, забезпечуючи їхню гнучкість, продуктивність і легкість у підтримці. Це особливо важливо для корпоративних додатків із великим навантаженням і високими вимогами до масштабованості.

Шаблон Composite

Структура та обгрунтування вибору.

Ознайомившись з теоретичними матеріалами, було прийнято рішення реалізувати шаблон Composite (Компонувальник). Компонувальник — це структурний патерн проєктування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт.

Буде доречно застосувати цей шаблон до desktop застосунку, де елементи графічного інтерфейсу користувача можуть бути «вкладені» один в одного. Наприклад, головне вікно за визначенням має всередині себе всі інші елементи застосунку. Також можна навести приклад кнопки меню, яка при натисканні відкриває список підпунктів. Загальну структуру графічно описано на Рисунку 1.

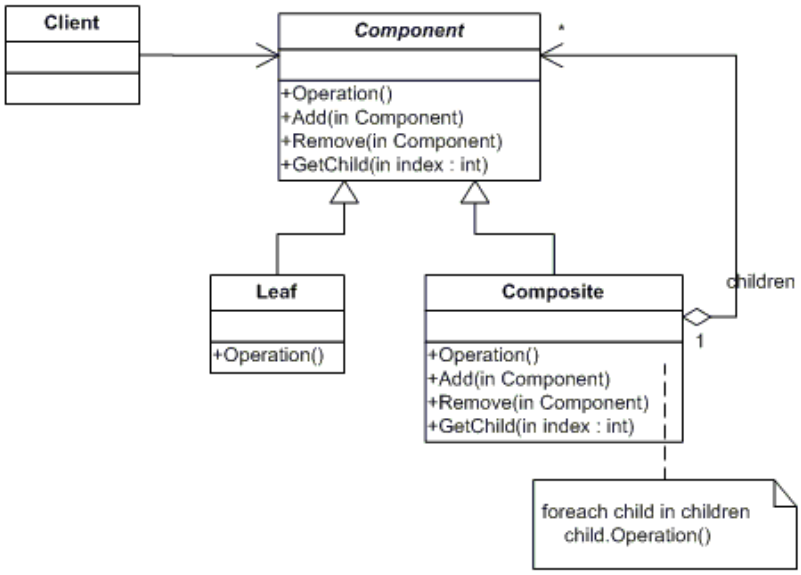

Рисунок 1. Загальна структура шаблону Composite

Компонент описує загальний інтерфейс для простих і складових компонентів дерева. Лист — це простий компонент дерева, який не має відгалужень. Класи листя міститимуть більшу частину корисного коду, тому що їм нікому передавати його виконання. Контейнер (або композит) — це складовий компонент дерева. Він містить набір дочірніх компонентів, але нічого не знає про їхні типи. Клієнт працює з деревом через загальний інтерфейс компонентів.

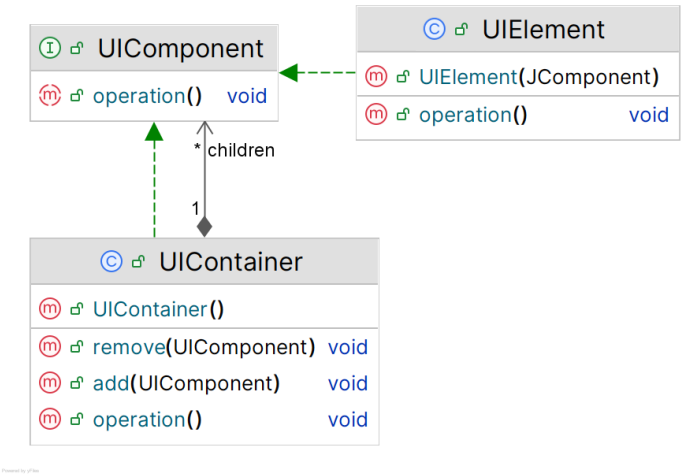

Рисунок 2. Структура реалізованого шаблону Composite

На Рисунку 2 зображено діаграму класів для шаблону Composite. UIComponent (інтерфейс) описує загальний контракт для всіх компонентів інтерфейсу користувача. Метод operation() виконує дію, специфічну для компонента.

UIContainer (клас Composite) реалізує контейнер для вкладених компонентів. Методи add() і remove() додають або видаляють дочірні компоненти. Поле children це список компонентів, які можуть бути контейнерами або елементами. Виклик operation() для кожного дочірнього компонента.

UIElement (клас Leaf) реалізує окремий елемент інтерфейсу користувача. Поле component зберігає Swing-компонент. Метод operation() виконує дію над компонентом, викликаючи його repaint() для оновлення UI.

Реалізація функціоналу у коді з використанням шаблону

Interface UIComponent

```
package org.example.composite;

public interface UIComponent {
    void operation();
}
```

Class UIContainer

```
package org.example.composite;

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

public class UIContainer implements UIComponent {
```

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

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

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

    @Override
    public void operation() {
        for (UIComponent child : children) {
            child.operation();
        }
    }
}
```

Class UIElement

```
package org.example.composite;

import javax.swing.*;

public class UIElement implements UIComponent {
    private JComponent component;

    public UIElement(JComponent component) {
        this.component = component;
    }

    @Override
    public void operation() {
        component.repaint();
    }
}
```

Class AudioEditorUI (використання шаблону)

```
package org.example.swing;

import org.example.database.DatabaseInitializer;
import org.example.logs.Logger;
import org.example.composite.*;

public class AudioEditorUI {
    public static void main(String[] args) {
        DatabaseInitializer.initializeDatabase();

        JFrame frame = new JFrame("Audio Editor");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(null);

        JButton loadFileButton = new JButton("Load Audio File");
        loadFileButton.setBounds(50, 50, 180, 30);

        JLabel fileLabel = new JLabel("No file selected");
        fileLabel.setBounds(250, 50, 300, 30);

        JButton convertToMp3Button = new JButton("Convert to MP3");
        convertToMp3Button.setBounds(50, 100, 180, 30);
```

```
        JButton convertToOggButton = new JButton("Convert to OGG");
        convertToOggButton.setBounds(50, 150, 180, 30);

        JButton convertToFlacButton = new JButton("Convert to FLAC");
        convertToFlacButton.setBounds(50, 200, 180, 30);

        JPanel waveformPanel = new JPanel();
        waveformPanel.setBounds(50, 250, 500, 100);
        waveformPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        Logger logger = new Logger();

        UIContainer mainContainer = new UIContainer();

        UIContainer buttonPanel = new UIContainer();
        buttonPanel.add(new UIElement(loadFileButton));
        buttonPanel.add(new UIElement(convertToMp3Button));
        buttonPanel.add(new UIElement(convertToOggButton));
        buttonPanel.add(new UIElement(convertToFlacButton));

        mainContainer.add(buttonPanel);
        mainContainer.add(new UIElement(fileLabel));
        mainContainer.add(new UIElement(waveformPanel));

        frame.add(loadFileButton);
        frame.add(fileLabel);
        frame.add(convertToMp3Button);
        frame.add(convertToOggButton);
        frame.add(convertToFlacButton);
        frame.add(waveformPanel);

        AudioEditorMediator mediator = new AudioEditorMediator(
                loadFileButton, convertToMp3Button, convertToOggButton,
convertToFlacButton,
                fileLabel, waveformPanel, logger);

        mainContainer.operation();

        frame.setVisible(true);
    }
}
```

Висновок

У ході даної лабораторної роботи я розглянув та дослідив теоретичні аспекти шаблонів проектування для роботи з базами даних і об'єктами, зокрема Active Record, Table Data Gateway, Data Mapper, Composite, Flyweight, Interpreter, і Visitor. Було вивчено їхні особливості, переваги та недоліки, а також приклади застосування в реальних програмних системах.

Особливу увагу було приділено реалізації шаблону Composite у контексті створення проекту аудіоредактора. Шаблон Composite показав себе як потужний інструмент для роботи з ієрархічними структурами, що забезпечує:

1. Гнучкість структури: дозволяє організувати компоненти програми у вигляді деревоподібної ієрархії.
2. Єдиний інтерфейс: спрощує роботу з окремими елементами та їх контейнерами, дозволяючи виконувати групові операції.
3. Рекурсивність обробки: забезпечує ефективне застосування операцій до всієї структури або її частин.

У результаті реалізації було створено систему, в якій звукові доріжки представлені як контейнери, що можуть містити як прості сегменти, так і вкладені структури. Це спростило виконання операцій над усією доріжкою чи її частинами, таких як копіювання, видалення або переміщення.

Шаблон Composite продемонстрував свою актуальність для створення масштабованих, зрозумілих і ефективних програмних систем. Він дозволив спростити реалізацію складних ієрархій об'єктів, що особливо важливо у таких проєктах, як аудіоредактори, де потрібна робота з численними взаємопов'язаними елементами.