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 №7

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

“ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»”

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

Принципи проєктування

Don't Repeat Yourself (DRY). Принцип DRY спрямований на уникнення дублювання коду. Якщо однаковий фрагмент коду повторюється в кількох місцях, це ускладнює підтримку програми, адже кожна зміна вимагатиме оновлення в усіх місцях. Щоб цього уникнути, необхідно використовувати функції, класи чи інтерфейси для реалізації загального функціоналу, підвищуючи гнучкість і зручність підтримки.

Keep It Simple, Stupid! (KISS). KISS закликає створювати прості, зрозумілі системи. Поділ програми на малі й логічно ізольовані компоненти робить її більш надійною, легкою для розуміння та модифікації. Надмірна складність ускладнює розробку та тестування, тому простота стає ключем до ефективності.

You Only Load It Once (YOLO). Принцип YOLO передбачає одноразову ініціалізацію даних під час запуску програми. Це знижує навантаження на систему, адже повторна обробка або завантаження даних можуть значно вплинути на продуктивність. Ефективне використання цього принципу зменшує час виконання операцій.

Принцип Парето. Цей принцип нагадує, що 80% результатів забезпечують лише 20% зусиль. У контексті програмування варто зосередитися на оптимізації найважливіших частин системи, які приносять найбільшу користь, замість рівномірного розподілу ресурсів на малозначущі задачі.

You Ain’t Gonna Need It (YAGNI). YAGNI рекомендує уникати додавання непотрібної функціональності. Система повинна реалізовувати лише ті можливості, які дійсно потрібні зараз, без передбачення майбутніх сценаріїв, які можуть ніколи не виникнути. Це дозволяє зменшити складність, підвищити продуктивність і скоротити час розробки.

Шаблони проєктування

Mediator (Посередник). Шаблон «Посередник» централізує взаємодію між об'єктами, щоб зменшити їхню залежність одне від одного. Усі комунікації здійснюються через окремий об'єкт-посередник, що спрощує структуру системи, знижує зв'язність компонентів і полегшує внесення змін.

Facade (Фасад). «Фасад» спрощує взаємодію із системою, пропонуючи єдиний інтерфейс для роботи з її складними підсистемами. Він приховує всі деталі реалізації, надаючи користувачам зрозумілий і зручний спосіб доступу до функціоналу системи.

Bridge (Міст). Шаблон «Міст» дозволяє розділити абстракцію та її реалізацію, що робить систему більш гнучкою. Це особливо корисно, коли потрібно незалежно розширювати як абстракцію, так і реалізацію, зменшуючи кількість класів та складність ієрархії.

Template Method (Шаблонний метод). Шаблонний метод визначає загальну структуру алгоритму у базовому класі, залишаючи підкласам можливість реалізовувати окремі його кроки. Такий підхід зберігає основну логіку незмінною, водночас забезпечуючи гнучкість у деталях реалізації.

Шаблон Mediator

Структура та обгрунтування вибору.

Ознайомившись з теоретичними матеріалами, було прийнято рішення реалізувати шаблон Mediator. Застосунок складається з багатьох елементів графічного інтерфейсу, робити між ними залежності всередині один одного – нераціонально. Mediator дає змогу зменшити зв’язність елементів між собою, отже значно облегшить розробку. Основна ідея шаблону — централізувати управління взаємодією між компонентами через один об'єкт-посередник. Загальну структуру графічно описано на Рисунку 1.

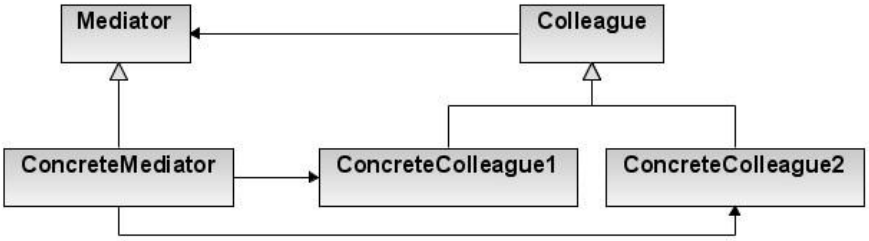

Рисунок 1. Загальна структура шаблону Mediator

Компоненти — це різнорідні об’єкти, що містять бізнес-логіку програми. Посередник визначає інтерфейс для обміну інформацією з компонентами. Конкретний посередник містить код взаємодії кількох компонентів між собою. Компоненти не повинні спілкуватися один з одним безпосередньо. Якщо в компоненті відбувається важлива подія, він повинен повідомити свого посередника, а той сам вирішить, чи стосується подія інших компонентів, і чи треба їх сповістити.

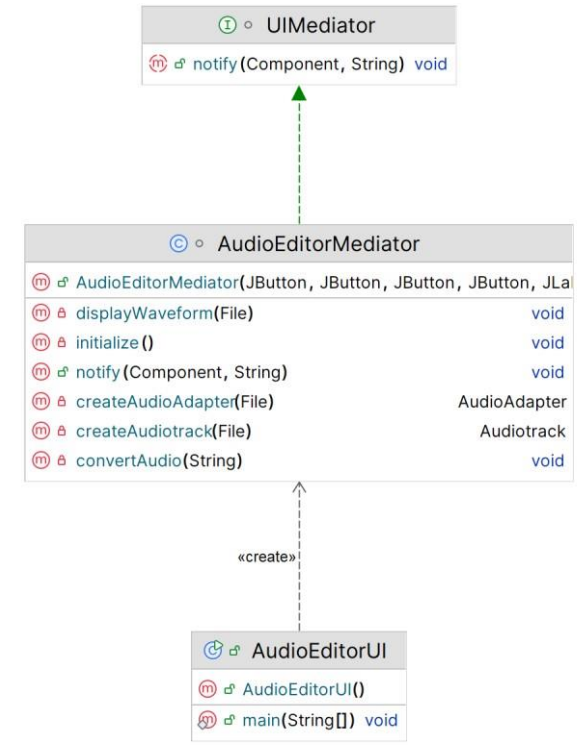

Рисунок 2. Структура реалізованого шаблону Mediator

Інтерфейс UIMediator визначає метод notify(Component sender, String event) для зв'язку між компонентами.

Конкретний Посередник AudioEditorMediator реалізує інтерфейс UIMediator та містить логіку координації між компонентами інтерфейсу.

Головний клас AudioEditorUI створює графічний інтерфейс і ініціалізує AudioEditorMediator.

У цьому випадку компонентами інтерфейсу є: JButton (loadFileButton, convertToMp3Button, тощо), JLabel (fileLabel), JPanel (waveformPanel),Logger.

Реалізація функціоналу у коді з використанням шаблону

Interface UIMediator

```
package org.example.swing;

import java.awt.*;

interface UIMediator {
    void notify(Component sender, String event);
}
```

Class AudioEditorUI

```
package org.example.swing;

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

import javax.swing.*;
import java.awt.*;

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);
        frame.add(loadFileButton);

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

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

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

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

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

        Logger logger = new Logger();

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

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

Class AudioEditorMediator

```
class AudioEditorMediator implements UIMediator {
    private JButton loadFileButton;
    private JButton convertToMp3Button;
    private JButton convertToOggButton;
    private JButton convertToFlacButton;
    private JLabel fileLabel;
    private JPanel waveformPanel;
    private Logger logger;
    private File selectedFile;

    public AudioEditorMediator(JButton loadFileButton, JButton
convertToMp3Button, JButton convertToOggButton, JButton convertToFlacButton,
                               JLabel fileLabel, JPanel waveformPanel, Logger
logger) {
        this.loadFileButton = loadFileButton;
        this.convertToMp3Button = convertToMp3Button;
        this.convertToOggButton = convertToOggButton;
        this.convertToFlacButton = convertToFlacButton;
        this.fileLabel = fileLabel;
        this.waveformPanel = waveformPanel;
        this.logger = logger;

        initialize();
    }
    private void initialize() {
        loadFileButton.addActionListener(e -> notify(loadFileButton,
"loadFile"));
        convertToMp3Button.addActionListener(e -> notify(convertToMp3Button,
"convertToMp3"));
        convertToOggButton.addActionListener(e -> notify(convertToOggButton,
"convertToOgg"));
        convertToFlacButton.addActionListener(e -> notify(convertToFlacButton,
"convertToFlac"));
    }

    @Override
    public void notify(Component sender, String event) {
```

```
        switch (event) {
            case "loadFile":
                JFileChooser fileChooser = new JFileChooser();
                int returnValue = fileChooser.showOpenDialog(null);
                if (returnValue == JFileChooser.APPROVE_OPTION) {
                    selectedFile = fileChooser.getSelectedFile();
                    fileLabel.setText("Selected: " + selectedFile.getName());
                    displayWaveform(selectedFile);
                    logger.fileOpen();
                }
                break;

            case "convertToMp3":
                convertAudio("mp3");
                break;

            case "convertToOgg":
                convertAudio("ogg");
                break;

            case "convertToFlac":
                convertAudio("flac");
                break;

            default:
                throw new IllegalArgumentException("Unknown event: " + event);
        }
    }

    private AudioAdapter createAudioAdapter(File file) {
        Audiotrack audiotrack = createAudiotrack(file);
        return new AudioAdapter(audiotrack);
    }

    private Audiotrack createAudiotrack(File file) {
        String fileName = file.getName().toLowerCase();
        if (fileName.endsWith(".mp3")) {
            return new Mp3(file.getAbsolutePath());
        } else if (fileName.endsWith(".ogg")) {
            return new Ogg(file.getAbsolutePath());
        } else if (fileName.endsWith(".flac")) {
            return new Flac(file.getAbsolutePath());
        } else {
            throw new IllegalArgumentException("Unsupported file format: " +
fileName);
        }
    }
}
```

Висновок

У результаті виконання лабораторної роботи я вивчив основні принципи проєктування та шаблони, які забезпечують ефективну розробку програмного забезпечення. Принцип Don't Repeat Yourself (DRY) дозволяє уникати дублювання коду, що спрощує підтримку програми та зменшує ризик помилок.

Завдяки Keep It Simple, Stupid! (KISS) система стає зрозумілою, надійною та зручною для модифікації завдяки спрощенню її компонентів. You Only Load It Once (YOLO) допомагає оптимізувати продуктивність шляхом уникнення повторного завантаження даних, що знижує витрати ресурсів. Принцип Парето демонструє, що 80% результату можна досягти завдяки 20% зусиль, що дозволяє фокусуватися на найбільш значущих аспектах розробки. Принцип You Ain’t Gonna Need It (YAGNI) допомагає уникнути надмірної складності, розробляючи лише необхідну функціональність.

Шаблони проєктування також відіграють ключову роль у створенні гнучких і масштабованих систем. Шаблон Mediator забезпечує централізацію взаємодії між об'єктами, знижуючи їхню залежність один від одного, що спрощує підтримку системи. Facade приховує складність внутрішньої реалізації підсистем, пропонуючи єдиний і зрозумілий інтерфейс для роботи з ними. Шаблон Bridge дозволяє незалежно розвивати абстракцію та її реалізацію, забезпечуючи більшу гнучкість у розробці. Template Method допомагає стандартизувати структуру алгоритмів, залишаючи можливість для кастомізації окремих кроків підкласами.