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 № 4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: “ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»”

Варіант №5

Виконала:
студентка групи ІА-23
Архип’юк Катерина

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Завдання

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.

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. Короткі теоретичні відомості

Шаблони проєктування — це формалізовані рішення завдань у розробці, які описують, як ефективно вирішити проблему і де краще застосувати це рішення. Вони базуються на багаторічному досвіді розробників і спрощують роботу з системами. Наприклад, застосування шаблонів робить систему більш зрозумілою, стійкою до змін і легкою для подальшої інтеграції та підтримки. Водночас це є уніфікованою мовою для спілкування між розробниками. Шаблони корисні, але їх слід застосовувати обдумано, оскільки надмірне використання може погіршити дизайн.

Шаблон Singleton забезпечує створення тільки одного екземпляра класу і надає глобальну точку доступу до нього. Його використовують, коли потрібен єдиний об'єкт для управління спільними ресурсами, наприклад, налаштуваннями програми.

Шаблон Iterator дозволяє перебирати елементи колекції без розкриття її внутрішньої реалізації. Він виділяє логіку обходу в окремий клас, що робить код колекції простішим. Наприклад, для складних структур, як дерева, ітератор забезпечує послідовний обхід, навіть якщо спосіб обходу змінюється.

Шаблон Proxy створює об'єкт-замінник, який виконує додаткову логіку до або після звернення до реального об'єкта. Це корисно для відкладеної ініціалізації або доступу до ресурсоємних об'єктів. Наприклад, платіжна картка є проксі для готівки: забезпечує ту саму функцію, але зручніше у використанні.

Шаблон State змінює поведінку об'єкта залежно від його стану. Наприклад, банківська картка може нараховувати різні відсотки залежно від свого типу (Classic, Platinum). State допомагає уникнути великої кількості умов у коді, розділяючи стани в окремі класи.

Strategy (Стратегія) — це патерн проєктування, який дозволяє визначити сімейство схожих алгоритмів, інкапсулювати кожен з них у власний клас і зробити їх взаємозамінними. Стратегія дозволяє легко додавати нові алгоритми або змінювати існуючі, не змінюючи клієнтський код, що використовує ці алгоритми.

2. Шаблон Singleton

2.1 Структура та обгрунтування вибору

Шаблон Singleton використовується тут, щоб гарантувати, що кожен конвертор буде мати тільки один екземпляр протягом усього часу роботи програми. Це означає, що не будуть створюватись нові копії конвертера щоразу, коли потрібно конвертувати аудіофайл, а буде використовуватись той самий об'єкт. Це також допомагає уникнути помилок, пов'язаних з наявністю кількох копій одного й того ж об'єкта, і забезпечує централізоване управління доступом до нього.

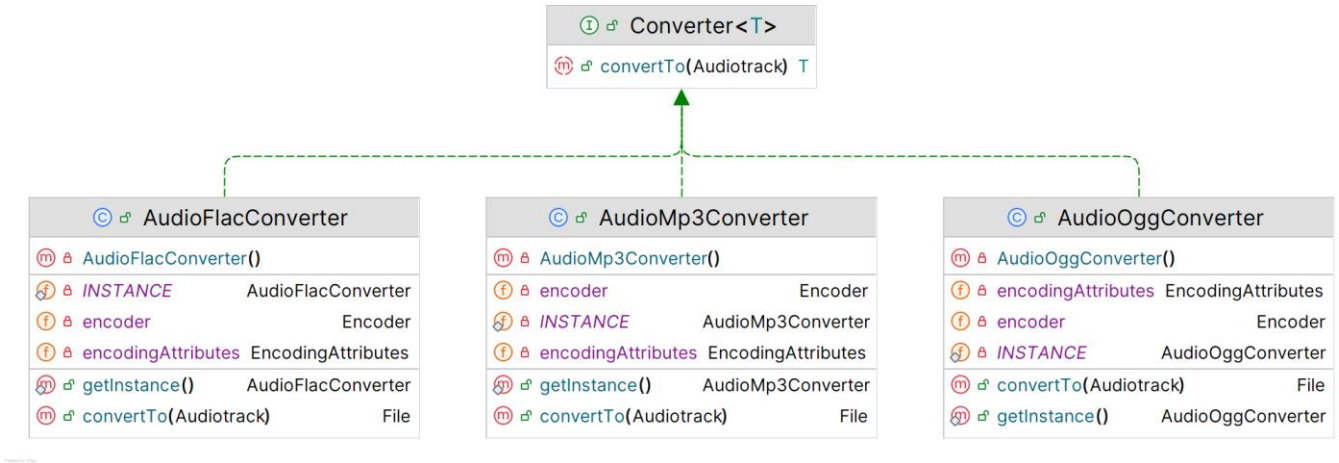

Рисунок 1. Структура реалізації шаблону Singleton для трьох класів-конвертерів

Converter<T> — це інтерфейс, що оголошує метод convertTo(Audiotrack audiotrack), який є спільним для всіх конвертерів.

Кожен з конвертерів містить:

INSTANCE — статичний екземпляр класу, який гарантує, що буде створено лише один об'єкт цього класу. Це є основною сутністю шаблону Singleton.

encoder та encodingAttributes — ресурси для кодування аудіофайлів, які використовуються в конвертаціях. Вони зберігаються на рівні класу, а не екземпляра, щоб уникнути повторної ініціалізації.

getInstance() — метод, який перевіряє, чи вже існує екземпляр цього класу, і якщо ні, створює його.

Усі конвертери мають свій метод convertTo(Audiotrack audiotrack), який відповідає за конвертацію аудіофайлів у відповідний формат (FLAC, MP3, OGG).

2.2 Реалізація функціоналу у коді з використанням шаблону

Interface Converter

```
package org.example.converter;

import org.example.audiotrack.Audiotrack;

public interface Converter<T> {
    T convertTo(Audiotrack audiotrack);
}
```

Class AudioFlacConverter

```
package org.example.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.example.audiotrack.Audiotrack;

import java.io.File;

public class AudioFlacConverter implements Converter<File> {
    private static AudioFlacConverter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioFlacConverter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("flac");
        encoder = new Encoder();
    }

    public static AudioFlacConverter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioFlacConverter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        File convertedFlac = new File(audiotrack.getFileLink().getParent() + "\\" +
audiotrack.getFileLink().getName()
                + " (converted to flac).flac");
        try {
            encoder.encode(audiotrack.getFileLink(), convertedFlac,
encodingAttributes);
        } catch (EncoderException e) {
            throw new RuntimeException(e);
        }
```

```
        return convertedFlac;
    }
}
```

Class AudioMp3Converter

```
package org.example.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.example.audiotrack.Audiotrack;

import java.io.File;
import java.io.IOException;

public class AudioMp3Converter implements Converter<File> {
    private static AudioMp3Converter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioMp3Converter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("mp3");
        encoder = new Encoder();
    }

    public static AudioMp3Converter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioMp3Converter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        try {
            File convertedMp3 = new File(audiotrack.getFileLink().getParent() + "\\"
+ audiotrack.getFileLink().getName() + " (converted to mp3).mp3");
            convertedMp3.createNewFile();
            encoder.encode(audiotrack.getFileLink(), convertedMp3,
encodingAttributes);
            return convertedMp3;
        } catch (EncoderException | IOException e) {
            throw new RuntimeException(e);
        }

    }
}
```

Class AudioOggConverter

```
package org.example.converter;

import it.sauronsoftware.jave.Encoder;
import it.sauronsoftware.jave.EncoderException;
import it.sauronsoftware.jave.EncodingAttributes;
import org.example.audiotrack.Audiotrack;

import java.io.File;
```

```
public class AudioOggConverter implements Converter<File> {
    private static AudioOggConverter INSTANCE;
    private EncodingAttributes encodingAttributes;
    private Encoder encoder;

    private AudioOggConverter() {
        encodingAttributes = new EncodingAttributes();
        encodingAttributes.setFormat("ogg");
        encoder = new Encoder();
    }

    public static AudioOggConverter getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new AudioOggConverter();
        }
        return INSTANCE;
    }

    @Override
    public File convertTo(Audiotrack audiotrack) {
        encodingAttributes.setAudioAttributes(audiotrack.getAudioAttributes());

        File convertedOgg = new File(audiotrack.getFileLink().getParent() + "\\" +
audiotrack.getFileLink().getName() + " (converted to ogg).ogg");
        try {
            encoder.encode(audiotrack.getFileLink(), convertedOgg,
encodingAttributes);
        } catch (EncoderException e) {
            throw new RuntimeException(e);
        }

        return convertedOgg;
    }
}
```

Висновки: При виконанні цієї лабораторної роботи я закріпила навички застосування шаблонів проєктування. Singleton гарантує існування тільки одного екземпляра класу, що корисно для глобальних ресурсів. Iterator дозволяє послідовно перебирати елементи колекції, не розкриваючи її структуру. Proxy створює об'єкта-замінник для контролю доступу до реального об'єкта. State змінює поведінку об'єкта залежно від його стану, що корисно для керування процесами. Strategy дає можливість змінювати алгоритм виконання під час роботи програми, що корисно для адаптивних систем.

Посилання на репозиторій з кодом проєкту:

https://github.com/KatiaArkhyp/AudioEditor