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

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

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

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 «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY» 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

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.

Шаблон Singleton

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.

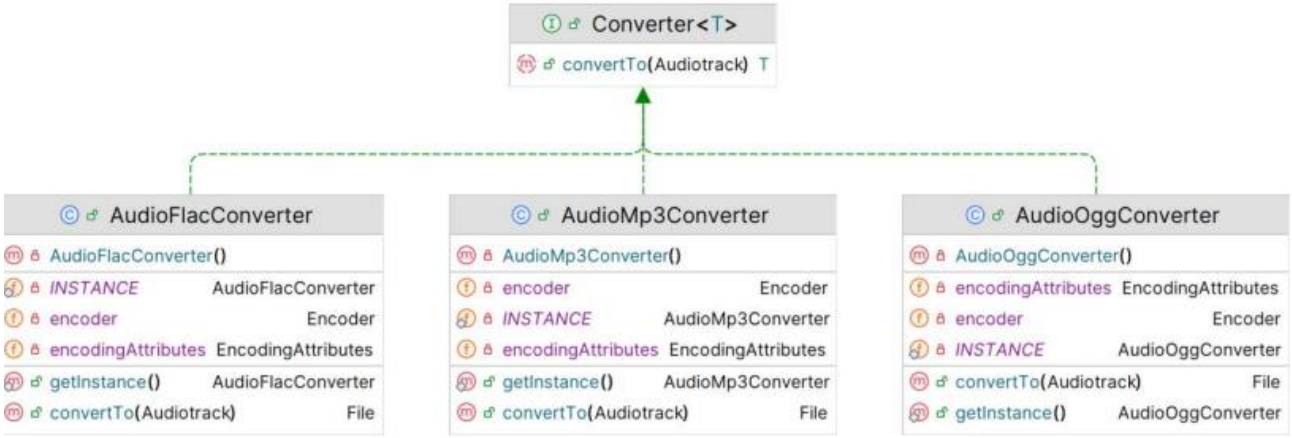

Рисунок 1. Структура реалізації шаблону Singleton для трьох класів-конвертерів

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).

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;
    }
```

Висновок

У ході виконання лабораторної роботи було розглянуто та реалізовано шаблон проектування Singleton для класів-конвертерів аудіофайлів. Досліджено особливості застосування цього шаблону, його переваги та недоліки.

Основні результати:

1. Реалізовано три класи-конвертери: AudioFlacConverter, AudioMp3Converter і AudioOggConverter, які забезпечують конвертацію аудіофайлів у відповідні формати.

2. Завдяки шаблону Singleton для кожного класу було створено лише один екземпляр, що гарантує економію ресурсів та централізоване управління екземплярами.
3. Упроваджено базовий узагальнений клас Converter<T>, який забезпечив уніфікацію інтерфейсу для реалізації різних типів конвертерів.

Переваги використання Singleton:

- Гарантія створення лише одного екземпляра класу.
- Спрощення доступу до об'єкта через метод getInstance().
- Централізоване управління та контроль за станом об'єкта.

Недоліки:

- Singleton ускладнює тестування через жорстке зв'язування з глобальним екземпляром.
- У багатопотокових додатках реалізація шаблону потребує додаткового захисту для уникнення проблем синхронізації.

Шаблон Singleton є ефективним інструментом для реалізації класів, які мають бути представлені в системі єдиним екземпляром. У випадку з класами-конвертерами Singleton забезпечує централізовану обробку аудіофайлів та ефективне використання системних ресурсів. Отримані знання та реалізація шаблону можуть бути застосовані для розробки програмного забезпечення з аналогічними вимогами до структури і функціональності.