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 № 5

з дисципліни «Технології розроблення програмного забезпечення»

Тема: “ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»”

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

Анти-шаблони проєктування (Anti-patterns) — це типові погані рішення, яких слід уникати. Вони з'являються при вирішенні проблем у розробці, коли використовується неефективний або недоцільний підхід. Термін з'явився як протилежність шаблонам проектування, описаним авторами "Банди чотирьох".

Анти-патерни в управлінні розробкою ПЗ:

- Дим і дзеркала: демонстрація незавершених функцій.
- Роздування ПЗ: збільшення вимог до ресурсів без виправданих змін.
- Функції для галочки: додавання непотрібних функцій для вигляду.

Анти-патерни в розробці ПЗ:

- Великий клубок бруду: складні і непідтримувані системи.
- Інверсія абстракції: приховування функціоналу без необхідності.
- Роздування інтерфейсу: занадто великі інтерфейси.

Анти-патерни в ООП:

- Божественний об'єкт: концентрація надмірної логіки в одному класі.
- Самотність (Singletonitis): надмірне використання патерну "Одинак".

Анти-патерни в програмуванні:

- Непотрібна складність: введення зайвої складності.
- Жорстке кодування: використання фіксованих значень в коді замість гнучких рішень.
- Спагеті-код: заплутаний і важкий для підтримки код.

Методологічні анти-патерни:

- Копіювання-вставка: копіювання коду замість створення спільних рішень.
- Передчасна оптимізація: оптимізація без достатньої інформації.

Шаблон Adapter – структурний шаблон, який дозволяє класам з несумісними інтерфейсами працювати разом. Адаптер «перетворює» інтерфейс одного класу в інтерфейс, очікуваний іншим класом. Це зручно, коли потрібно використовувати сторонній клас, але його інтерфейс не відповідає потребам вашої системи.

Приклад: Уявімо, що у вас є клас, який взаємодіє з API стороннього сервісу, і цей API має свій інтерфейс, відмінний від вашого. Для того, щоб інтегрувати цей API в свою систему без змін в існуючому коді, можна створити адаптер, який «перекладає» виклики до API в потрібний для вашого класу формат.

Шаблон Builder дозволяє створювати складні об'єкти покроково. Це особливо корисно, коли об'єкт має багато варіантів налаштувань, і їх не можна легко створити через один конструктор. Шаблон будується за допомогою окремих компонентів, що дозволяють створити різні варіанти об'єкта.

Цей шаблон передбачає наявність окремого класу, який відповідає за створення об'єкта, і методи для додавання різних частин або налаштувань. Наприклад, коли ви створюєте об'єкт «Автомобіль», можна додавати до нього різні компоненти, такі як двигун, трансмісія, колеса тощо.

Шаблон Command – поведінковий шаблон, який дозволяє інкапсулювати запит як об'єкт, таким чином, дозволяючи параметризувати методи з різними запитами, чергувати чи реєструвати запити, а також підтримувати відміни операцій. Команда відокремлює викликаючий об'єкт від об'єкта, що виконує дію.

Замість того щоб викликати метод безпосередньо, ви створюєте команду як об'єкт і передаєте її виконавцю. Це дозволяє зберігати інформацію про команду (наприклад, для підтримки відміни чи виконання в черзі). Команду можна використовувати для реалізації таких функцій, як макроси або відкладені операції.

Шаблон Chain of Responsibility – поведінковий шаблон, що дозволяє обробляти запит по черзі кількома об'єктами, причому кожен об'єкт може або обробити запит, або передати його наступному в ланцюзі. Цей шаблон дозволяє уникнути жорсткого зв'язку між відправником запиту та отримувачем.

Ланцюг обов'язків часто використовується для обробки запитів, де один з елементів ланцюга може обробити запит, а якщо ні — передати його далі. Наприклад, у системах перевірки прав доступу чи обробки подій цей шаблон дозволяє гнучко вирішувати, хто має обробити певний запит.

Шаблон Prototype – шаблон створювальний шаблоном, який дозволяє створювати нові об'єкти шляхом копіювання вже існуючих (прототипів) замість створення нових об'єктів з нуля. Цей шаблон використовує механізм клонування об'єктів для створення нових екземплярів.

Замість того щоб створювати об'єкти за допомогою стандартного конструктора, ви створюєте копію іншого об'єкта. Це зручно, коли об'єкти мають складні налаштування або ресурси, і їх створення потребує багато часу. У такому випадку, замість того щоб ініціалізувати новий об'єкт, ви копіюєте вже існуючий. Шаблон прототипу корисний для створення великих об'єктів з однаковими або схожими налаштуваннями.

2. Шаблон Adapter

2.1 Структура та обгрунтування вибору

Шаблон Adapter застосований тут для забезпечення сумісності між класами ієрархії Audiotrack та іншими компонентами програми або зовнішніми бібліотеками. Основна мета адаптера полягає в тому, щоб приховати специфічну реалізацію аудіотреків (Mp3, Flac, Ogg) і надати уніфікований інтерфейс для доступу до потрібних даних, таких як атрибути аудіо або файл аудіотреку.

Клас AudioAdapter виконує роль "перекладача", який дозволяє іншим компонентам працювати з об’єктами Audiotrack, не знаючи їхньої конкретної реалізації. Наприклад, метод adaptAttributes() повертає атрибути аудіо у форматі, очікуваному зовнішньою бібліотекою, а метод adaptFile() повертає об'єкт файлу.

Описану структуру показано на Рисунку 1.

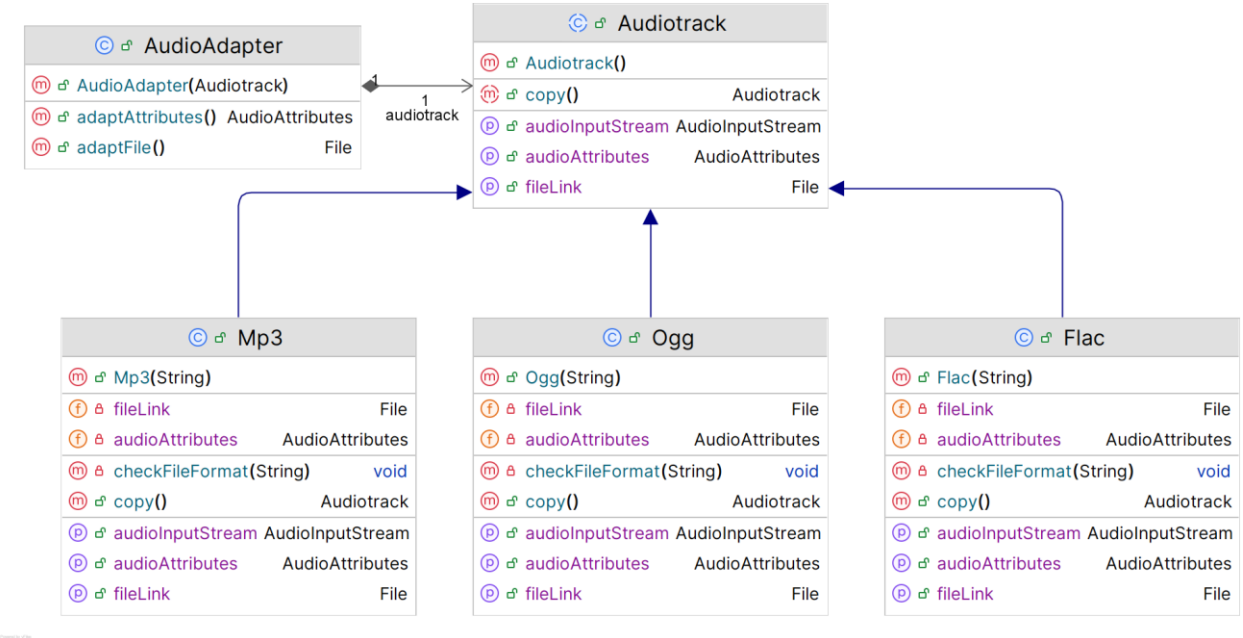

Рисунок 1. Структура класів реалізованого шаблону Adapter

Audiotrack — абстрактний клас, який визначає загальний інтерфейс для аудіоформатів, з методами для отримання атрибутів аудіо, файлів і копії треку. Класи Mp3, Ogg, і Flac наслідують його та реалізують ці методи для конкретних форматів.

AudioAdapter — адаптер, який містить об'єкт Audiotrack і надає зручний інтерфейс для отримання атрибутів і файлів з конкретних реалізацій Audiotrack, таких як Mp3, Ogg, або Flac. Він перетворює різні формати в єдиний, стандартизований інтерфейс, що дозволяє працювати з ними однотипно.

2.2 Реалізація функціоналу у коді з використанням шаблону

abstract class Audiotrack

```
public abstract class Audiotrack {
    public abstract AudioAttributes getAudioAttributes();
    public abstract File getFileLink();
    public abstract AudioInputStream getAudioInputStream();
    public abstract Audiotrack copy();
}
```

class AudioAdapter

```
package org.example.audiotrack;

import it.sauronsoftware.jave.AudioAttributes;
```

```
import java.io.File;

public class AudioAdapter{
    private final Audiotrack audiotrack;

    public AudioAdapter(Audiotrack audiotrack) {
        this.audiotrack = audiotrack;
    }

    public AudioAttributes adaptAttributes() {
        return audiotrack.getAudioAttributes();
    }

    public File adaptFile() {
        return audiotrack.getFileLink();
    }
}
```

class Flac

```
package org.example.audiotrack;

import it.sauronsoftware.jave.AudioAttributes;

import javax.sound.sampled.AudioInputStream;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;


public class Flac extends Audiotrack {
    private AudioAttributes audioAttributes;
    private File fileLink;

    public Flac(String filePath) {
        checkFileFormat(filePath);
        File audioFile = new File(filePath);

        fileLink = audioFile;
        audioAttributes = new AudioAttributes();
        audioAttributes.setCodec("flac");
        audioAttributes.setBitRate(128000);
        audioAttributes.setChannels(2);
        audioAttributes.setSamplingRate(44100);
    }


    private void checkFileFormat(String path) {
        try {
            if (!Files.probeContentType(Path.of(path)).equals("audio/x-flac")) {
                throw new IllegalArgumentException("Wrong audio format");
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public AudioAttributes getAudioAttributes() {
        return audioAttributes;
    }
```

```
    public File getFileLink() {
        return fileLink;
    }

    @Override
    public AudioInputStream getAudioInputStream() {
        return null;
    }

    @Override
    public Audiotrack copy() {
        return null;
    }
}
```

class Mp3

```
package org.example.audiotrack;

import it.sauronsoftware.jave.AudioAttributes;

import javax.sound.sampled.AudioInputStream;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class Mp3 extends Audiotrack {
    private AudioAttributes audioAttributes;
    private File fileLink;

    public Mp3(String filePath) {
        checkFileFormat(filePath);
        File audioFile = new File(filePath);
        fileLink = audioFile;
        audioAttributes = new AudioAttributes();
        audioAttributes.setCodec("libmp3lame");
        audioAttributes.setBitRate(128000);
        audioAttributes.setChannels(2);
        audioAttributes.setSamplingRate(44100);
    }


    private void checkFileFormat(String path) {
        try {
            if (!Files.probeContentType(Path.of(path)).equals("audio/mpeg")) {
                throw new IllegalArgumentException("Wrong audio format");
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public AudioAttributes getAudioAttributes() {
        return audioAttributes;
    }

    public File getFileLink() {
        return fileLink;
    }
```

```
    @Override
    public AudioInputStream getAudioInputStream() {
        return null;
    }

    @Override
    public Audiotrack copy() {
        return null;
    }
}
```

class Ogg

```
package org.example.audiotrack;

import it.sauronsoftware.jave.AudioAttributes;

import javax.sound.sampled.AudioInputStream;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

public class Ogg extends Audiotrack {
    private AudioAttributes audioAttributes;
    private File fileLink;

    public Ogg(String filePath) {
        checkFileFormat(filePath);
        File audioFile = new File(filePath);
        fileLink = audioFile;
        audioAttributes = new AudioAttributes();
        audioAttributes.setCodec("vorbis");
        audioAttributes.setBitRate(128000);
        audioAttributes.setChannels(2);
        audioAttributes.setSamplingRate(44100);
    }


    private void checkFileFormat(String path) {
        try {
            if (!Files.probeContentType(Path.of(path)).equals("audio/ogg")) {
                throw new IllegalArgumentException("Wrong audio format");
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }


    public AudioAttributes getAudioAttributes() {
        return audioAttributes;
    }


    public File getFileLink() {
        return fileLink;
    }

    @Override
    public AudioInputStream getAudioInputStream() {
        return null;
```

```
    }

    @Override
    public Audiotrack copy() {
        return null;
    }
}
```

Висновки: При виконанні цієї лабораторної роботи я закріпила навички застосування шаблонів проєктування. Шаблон Adapter використовується для забезпечення сумісності класів з різними інтерфейсами. Builder допомагає створювати складні об'єкти покроково, надаючи гнучкість у налаштуваннях. Command дозволяє інкапсулювати запити як об'єкти для більшої гнучкості у виконанні операцій. Chain of Responsibility допомагає обробляти запити через послідовність об'єктів, де кожен має можливість обробити або передати запит далі. Prototype дозволяє створювати нові об'єкти через клонування, що заощаджує ресурси при створенні схожих об'єктів.

Посилання на репозиторій з кодом проєкту:
https://github.com/KatiaArkhyp/AudioEditor