Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота №7

із дисципліни «*ТРПЗ*»

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Виконав:
Студент групи ІА-24
Бакалець А.І.

Перевірив:
Мягкий М.Ю.

Київ-2024

Мета: Навчитися використовувати шаблони mediator, facade, bridge, template method.

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:

..1 Музичний програвач (iterator, command, memento, facade, visitor, client-server)

Музичний програвач становить собою програму для програвання музичних файлів або відтворення потокової музики з можливістю створення, запам'ятовування і редагування списків програвання, перемішування/повторення (shuffle/repeat), розпізнавання різних аудіо-форматів, еквалайзер.

Github: https://github.com/Clasher3000/TRPZ_labs

Теоретичні відомості

Посередник (Mediator)

Шаблон "Посередник" забезпечує централізоване управління взаємодією між об'єктами, зменшуючи кількість прямих залежностей між ними. Використовується, коли потрібно уникнути складності, що виникає через безпосередні зв'язки між об'єктами в системі. У Java цей шаблон зазвичай реалізується шляхом створення посередника, який координує обмін даними або подіями між об'єктами, що реєструються у ньому. Це спрощує модифікацію і підтримку компонентів, зменшуючи зв'язаність між ними.

Фасад (Facade)

Шаблон "Фасад" надає єдиний інтерфейс до групи взаємопов'язаних класів або підсистем, спрощуючи їх використання. Використовується для зменшення складності системи, приховуючи її деталі реалізації та надаючи зручний API. У Java цей шаблон зазвичай реалізується через клас, який містить методи для основних операцій, що викликають функціонал внутрішніх класів або підсистем. Це дозволяє зменшити залежності між клієнтським кодом і складними підсистемами, спрощуючи розробку і підтримку.

Міст (Bridge)

Шаблон "Міст" розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно одна від одної. Використовується для уникнення жорсткого зв'язку між

абстракцією та її конкретними реалізаціями. У Java цей шаблон реалізується через створення інтерфейсу або абстрактного класу для абстракції, а також окремих класів для реалізацій, які пов'язуються через композицію. Це забезпечує більшу гнучкість у розширенні функціональності системи.

Шаблонний метод (Template Method)

Шаблон "Шаблонний метод" визначає загальну структуру алгоритму, делегуючи реалізацію деяких його кроків підкласам. Використовується для забезпечення гнучкості та уникнення дублювання коду, коли алгоритм має спільні етапи для різних реалізацій. У Java цей шаблон реалізується через абстрактний клас із визначеним методом-шаблоном, який викликає конкретні реалізації абстрактних методів, що задаються підкласами. Це дозволяє стандартизувати алгоритм, зберігаючи варіативність його частин.

Хід роботи

У цій лабораторній роботі я реалізував шаблон **Фасад (Facade),** оскільки моя програма потребує спрощення доступу до складної системи, яка складається з кількох взаємопов’язаних підсистем. Цей шаблон дозволяє надати єдиний інтерфейс для управління різними компонентами системи, приховуючи деталі їх реалізації.

Шаблон Facade допоміг зменшити складність використання моєї програми, забезпечивши зручний API для виконання основних дій. Реалізація цього шаблону дозволила інкапсулювати логіку взаємодії з сервісами треків, плейлистів і механізмом відтворення музики. Це спростило код, зробивши його більш зрозумілим і підтримуваним.

Завдяки реалізації шаблону Facade моя програма надала користувачам прості методи для виконання основних операцій, таких як відтворення треку, запуск плейлиста, пауза, відновлення та зупинка музики. Користувачеві не потрібно знати, як саме обробляються файли, працюють сервіси або обробляються винятки — усе це виконується за фасадом.

Шаблон Facade дозволив значно зменшити зв'язаність між компонентами програми та клієнтським кодом, що спростило її розширення і підтримку. Наприклад, якщо в майбутньому потрібно змінити логіку роботи з треками чи плеєром, це можна зробити, не змінюючи клієнтський код, оскільки всі зміни будуть приховані за фасадом.

У моїй програмі реалізація шаблону включає клас **MusicPlayer**, який виконує роль фасаду. Він забезпечує єдиний доступ до функціоналу, пов’язаного з треками, плейлистами та їх відтворенням, надаючи прості й інтуїтивно зрозумілі методи для управління музикою. Це підвищило зручність роботи з програмою та спростило її інтеграцію в інші системи.

Структура шаблону Facade:

```java
public class MusicPlayer {
    2 usages
    private TrackService trackService;
    3 usages
    private PlaylistService playlistService;
    7 usages
    private AdvancedPlayer player;
    5 usages
    private Thread playThread;
    21 usages
    private PrintWriter out;
    10 usages
    private Track track;
    14 usages
    private Playlist playlist;
    2 usages
    private int pausedFrame = 0;
    4 usages
    private boolean isPaused = false;
    11 usages
    private TrackIterator trackIterator;
    3 usages
    private final Caretaker caretaker = new Caretaker();

    public MusicPlayer(PrintWriter out) {
        this.trackService = new TrackServiceImpl();
        this.playlistService = new PlayListServiceImpl();
        this.out = out;
    }

    6 usages  Clasher3000 *
    public void playSong(String fileName) {
        stopSong();

        playThread = new Thread(() -> {
            try {
                track = trackService.findByTitle(fileName);
                if (track == null) {
                    out.println("Track not found.");
                    return;
                }
                FileInputStream fis = new FileInputStream(track.getPath());
                BufferedInputStream bis = new BufferedInputStream(fis);

                player = new AdvancedPlayer(bis);
                out.println("Playing: " + fileName);
                player.play();

            } catch (NoResultException e) {
                out.println("Incorrect track title");
```

```
        } catch (JavaLayerException | IOException e) {
            out.println("Error playing song: " + e.getMessage());
        }
    });

    playThread.start();
}
```

```
1 usage   Clasher3000
public void playPlaylist(String playlistName) {
    try {
        playlist = playlistService.findByName(playlistName);
        if (playlist != null && playlist.getTracks() != null && !playlist.getTracks().isEmpty()) {
            trackIterator = new TrackIteratorImpl(playlist.getTracks());

            playNextTrackInPlaylist();
            out.println("Playlist: " + playlistName + " is playing");
        } else {
            out.println("Playlist is empty or does not exist.");
        }
    } catch (NoResultException e) {
        out.println("Incorrect playlist name");
    }
}
```

Рис.1-3 – Клас-фасад MusicPlayer

Клас **MusicPlayer** у нашій програмі виконує роль посередника між клієнтом і складною системою взаємодії з об'єктами, такими як Track, Playlist, та сервіси для роботи з ними. Його завдання — забезпечити простий і зрозумілий інтерфейс для користувача, приховуючи всі технічні деталі.

Цей клас інкапсулює роботу кількох сервісів, включаючи TrackService і PlaylistService, а також функціонал відтворення музики через плеєр. Він пропонує зручні методи для виконання ключових операцій, таких як відтворення треку, запуск плейлиста, зупинка, пауза та відновлення.

Клас дозволяє клієнтському коду взаємодіяти з програмою на високому рівні, не вникаючи в подробиці роботи плеєра чи взаємодії з базою даних. Наприклад, клієнт може викликати метод playSong для відтворення треку, не турбуючись про те, як здійснюється пошук треку, обробка помилок чи ініціалізація плеєра.

Завдяки цьому класу досягається зменшення зв'язаності між клієнтським кодом і підсистемами програми, що спрощує їх розширення і підтримку. У майбутньому можна змінити логіку сервісів або додати нові функції без змін клієнтського коду. Таким чином, клас **MusicPlayer** є ключовим елементом, який робить програму більш зручною для використання та підтримки.

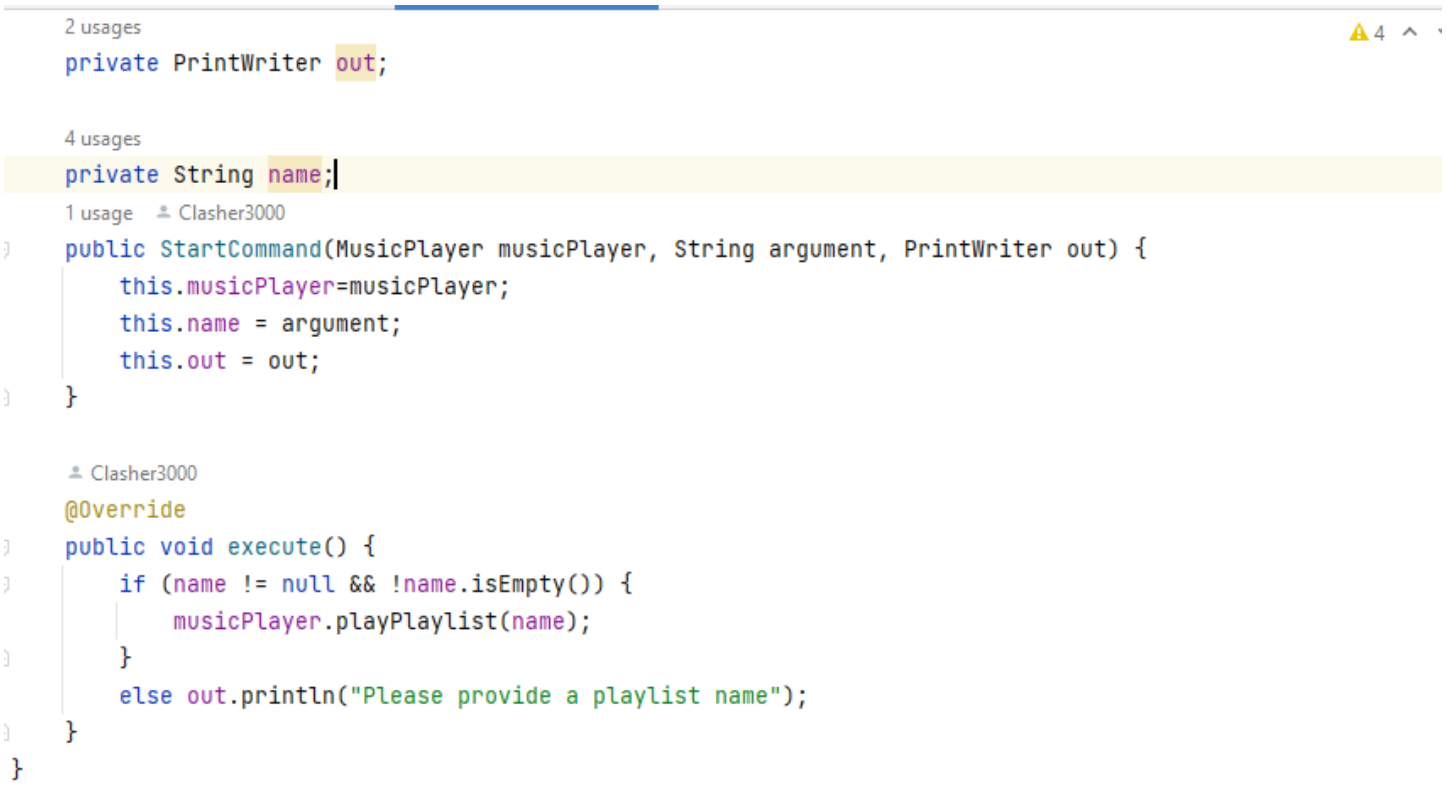

Рис.4 – Приклад роботи з нашим класом-фасадом MusicPlayer

Клас **MusicPlayer** у ролі фасада спрощує взаємодію клієнта з системою, яка керує відтворенням музики та плейлистами. У наведеному прикладі клас StartCommand використовує об'єкт **MusicPlayer** для відтворення плейлиста.

Цей приклад демонструє, як фасад **MusicPlayer** приховує складність роботи з підсистемами (наприклад, пошуком, запуском плейлистів, ітератором), дозволяючи клієнтському коду працювати лише з простими командами.

Висновок: В процесі виконання лабораторної роботи я ознайомився з такими патернами проєктування, як Facade, Mediator, Bridge та Template Method. Особливу увагу я приділив шаблону Facade, який дозволив спростити взаємодію клієнта із системою керування музичним плеєром. Завдяки фасаду вдалося приховати складність роботи з підсистемами, такими як плейлисти, треки та їх відтворення, надаючи простий і зрозумілий інтерфейс для клієнтського коду. Це забезпечило зручність у використанні, спрощення структури коду і його масштабування.

Застосування зазначених патернів підвищило модульність програми, забезпечило розділення відповідальностей і спростило тестування. Використання цих шаблонів зробило код більш читабельним, підтримуваним і гнучким для майбутніх змін або розширень функціоналу.