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

Технології розроблення програмного забезпечення

Лабораторна робота №4
«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY». »

Виконала:
Студентка групи ІА-22
Микитенко Ірина

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: шаблони «singleton», «iterator», «proxy», «state», «strategy»..

Мета: ознайомитися з шаблонами проектування «Singleton», «Iterator», «Proxy», » та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи

..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.

1. Короткі теоритичні відомості

Шаблони проектування – це формалізовані рішення типових завдань, які часто виникають при розробці інформаційних систем. Вони містять опис завдання, вдале рішення та рекомендації щодо його використання в різних ситуаціях. Кожен шаблон має загальноприйняту назву, що дозволяє легко ідентифікувати та повторно використовувати його.

Важливим етапом роботи з шаблонами є правильне моделювання предметної області, що допомагає формалізувати задачу і вибрати відповідний шаблон. Використання шаблонів проектування надає розробнику низку переваг: вони роблять систему структурованішою, простішою у вивченні та розширенні, підвищують її стійкість до змін і полегшують інтеграцію з іншими системами.

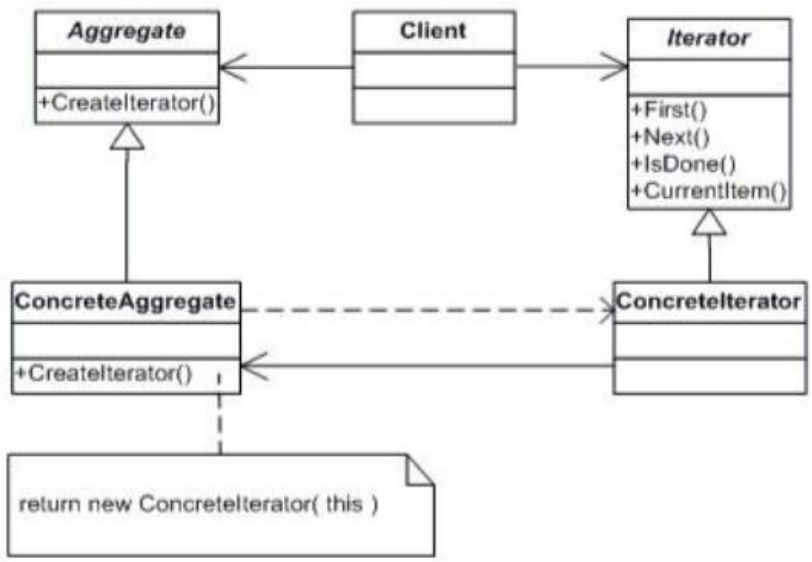
Таким чином, шаблони є перевіреними рішеннями, які використовуються в різних компаніях і проектах для створення архітектурних рішень у відповідних обставинах.

Навіщо використовувати шаблони ?

Шаблони проектування зменшують час і трудовитрати на створення архітектури, надають системі гнучкість та полегшують її підтримку. Вони полегшують спільне розуміння архітектури в команді та є економічно виправданими, але їх варто використовувати обдумано, а не в кожному проекті.

Шаблон «Iterator»

Структура:



Шаблонний ітератор містить:

▪ First() – установка покажчика перебору на перший елемент колекції;

▪ Next() – установка покажчика перебору на наступний елемент колекції;

▪ IsDone – булевське поле, яке встановлюється як true коли покажчик перебору досяг кінця

колекції;

▪ CurrentItem – поточний об'єкт колекції.

Призначення: Шаблон «Iterator» забезпечує доступ до елементів колекції без розкриття її внутрішньої структури. Він розділяє відповідальність: колекція зберігає дані, а ітератор відповідає за їх обхід. Ітератор дозволяє використовувати різні алгоритми проходження, як-от обхід у зворотному порядку або по парних/непарних елементах.

Проблема: різні колекції можуть мати складні структури (дерева, графи), і для них потрібні різні алгоритми обходу. Додавання таких алгоритмів безпосередньо до колекції ускладнює її код.

Рішення: винести логіку обходу в окремий клас ітератора, який відстежує стан і поточну позицію в колекції. Це дозволяє створювати нові способи обходу без зміни коду колекції.

Переваги:

- Дозволяє використовувати різні алгоритми обходу.

- Спрощує код колекцій.

Недоліки:

- Може бути зайвим, якщо достатньо простого циклу.

2. Реалізація не менше 3-х класів відповідно до обраної теми.

Структура проекту з реалізованими класами зображена на рисунку 1

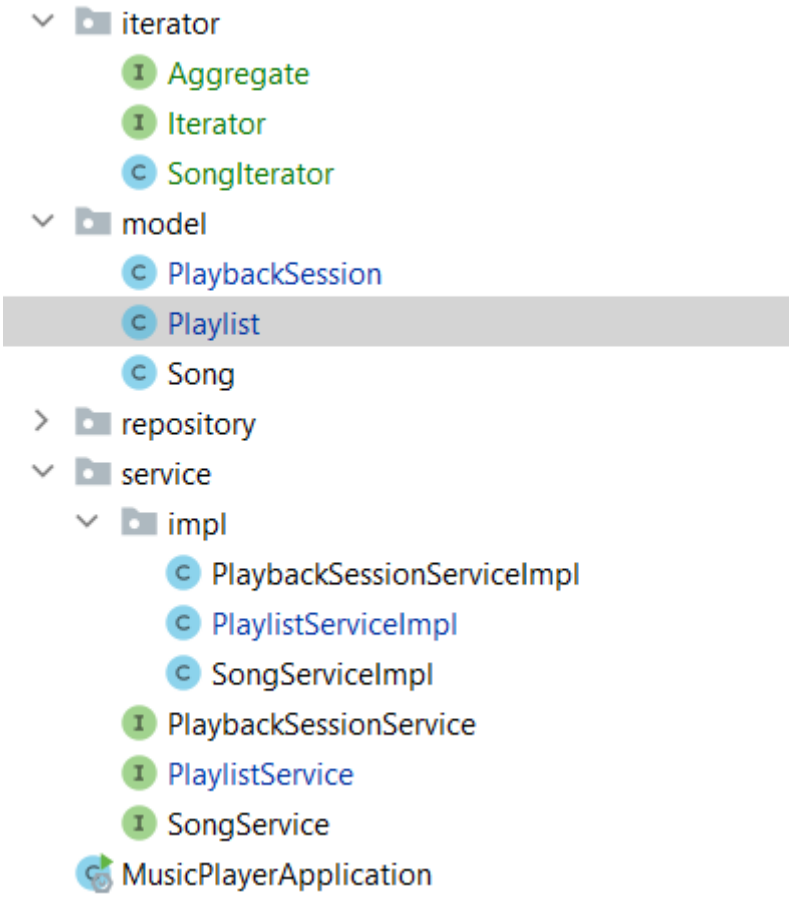


Рисунок 1 – структура проекту

У ході лабораторної роботи була реалізована система музичного плеєра, де основним завданням було використання патерну Ітератор для роботи з колекцією пісень у плейлисті. В результаті виконано такі кроки:

1 **Клас Playlist (Агрегатор):** Цей клас відповідає за зберігання колекції пісень у плейлисті. Окрім того, він надає метод для створення ітератора, який дозволяє пройтися по кожній пісні плейлиста. Основні операції цього класу включають додавання та видалення пісень.

2 **Клас SongIterator (Ітератор):** Реалізує логіку обходу елементів (пісень) у плейлисті. Він дозволяє отримати доступ до кожної пісні по черзі, перевіряти, чи є наступна пісня, і повертати її. Це дозволяє зручно маніпулювати плейлистом, не розкриваючи його внутрішню структуру.

3 **Клас PlaylistService (Клієнт):** Сервіс, який виступає клієнтом ітератора. Він отримує плейлисти з бази даних, додає або видаляє пісні, а також використовує SongIterator для доступу до пісень у плейлисті. PlaylistService виступає "посередником" між збереженими даними та логікою роботи з ними.

4. Клас PlaylistService виконує роль клієнта у патерні Ітератор. Він забезпечує взаємодію між контролером, репозиторіями, плейлистами та піснями. Основне завдання цього сервісу – виконувати бізнес-логіку роботи з плейлистами та піснями, використовуючи ітератор для доступу до колекції пісень.

Основні функції класу PlaylistService:

1. **Отримання плейлистів:** Сервіс отримує плейлист із бази даних за допомогою репозиторію (playlistRepository). Після цього він надає доступ до пісень у цьому плейлисті.
2. **Додавання та видалення пісень:** Сервіс дозволяє додавати пісні до плейлиста або видаляти їх, зберігаючи ці зміни через репозиторій. Це забезпечує можливість модифікації колекції пісень у плейлисті.
3. **Робота з ітератором:** Однією з ключових функцій є використання патерну Ітератор. Сервіс отримує ітератор через метод createIterator() у класі Playlist. Після цього за допомогою методу hasNext() він перевіряє наявність пісень, а методом next() проходить по кожній з них, забезпечуючи доступ до пісень у плейлисті без необхідності маніпулювати самою колекцією напряму.
4. **Інтеграція з контролером:** Сервіс PlaylistService надає методи, які можуть бути викликані контролером для виконання операцій над плейлистами: отримання пісень, додавання нових, видалення тощо. Ця взаємодія відбувається через стандартні CRUD-операції.

3. Реалізація шаблону за обраною темою

Для виконання цього пункту лабораторної роботи потрібно реалізувати один з розглянутих шаблонів проєктування. Оскільки ми працюємо над темою музичного плеєра з використанням **патерну Ітератор**, то реалізуємо цей шаблон для проходження по списку пісень у плейлисті.

Кроки реалізації патерну Ітератор:

1. Інтерфейс Iterator<T>

- Це загальний інтерфейс для всіх ітераторів, який визначає базові методи для роботи з колекцією. Даний інтерфейс зображено на рисунку 2

```java
package com.example.music_player.iterator;

public interface Iterator<T> {  6 usages  1 implementation  imykytenko
    boolean hasNext();  2 usages  1 implementation  imykytenko
    T next();  1 usage  1 implementation  imykytenko
}
```

Рисунок 2 – інтерфейс Iterator<T>

2. Інтерфейс Aggregate<T>

- Інтерфейс для колекції, яка може створювати ітератор. У нашому випадку це буде плейлист, який має метод для створення ітератора. Даний інтерфейс зображено на рисунку 3

```java
package com.example.music_player.iterator;

public interface Aggregate<T> {  2 usages  1 implementation  imykytenko
    Iterator<T> createIterator();  1 usage  1 implementation  imykytenko
}
```

Рисунок 3 - Інтерфейс Aggregate<T>

3. Клас SongIterator (Ітератор для пісень)

- Цей клас реалізує інтерфейс Iterator і відповідає за ітерацію по списку пісень у плейлисті. Даний клас зображено на рисунку 4

```java
import com.example.music_player.model.Song;
import java.util.List;
import java.util.NoSuchElementException;

public class SongIterator implements Iterator<Song>{  2 usages  imykytenko
    private final List<Song> songs;  3 usages
    private int position;  2 usages

    public SongIterator(List<Song> songs) { this.songs = songs; }

    @Override  2 usages  imykytenko
    public boolean hasNext() { return position < songs.size(); }

    @Override  1 usage  imykytenko
    public Song next() {
        if (!hasNext()) {
            throw new NoSuchElementException("No more songs in the playlist.");
        }
        return songs.get(position++);
    }
}
```

Рисунок 4 - Клас SongIterator

4. Клас Playlist (Агрегатор)

- Це основний клас, який зберігає список пісень і надає метод для створення ітератора. Даний метод в класі ми можемо побачити на рисунку 5

```
@Override  1 usage  imykytenko
public Iterator<Song> createIterator() {
    return new SongIterator(songs);
}
```

Рисунок 5 - Клас Playlist

5. Клас PlaylistService (Клієнт)

- Сервіс відповідає за взаємодію з плейлистом і використання ітератора для проходження по піснях. На рисунку 6 ми можемо побачити використання ітератора.

```
@Override  1 usage  imykytenko
public List<Song> getSongsInPlaylist(Long playlistId) {
    Playlist playlist = getPlaylistById(playlistId);

    if (playlist == null) {
        return null;
    }

    List<Song> songsInPlaylist = new ArrayList<>();
    Iterator<Song> songIterator = playlist.createIterator();

    while (songIterator.hasNext()) {
        songsInPlaylist.add(songIterator.next());
    }

    return songsInPlaylist;
}
```

Рисунок 6 - Клас PlaylistService

PlaylistService виступає клієнтом, який використовує ітератор для доступу до пісень у плейлисті та передає їх контролеру для подальшого відображення.

Діаграма класів для паттерну Iterator

Діаграма класів, які реалізовують паттерн Iterator зображена на рисунку 7

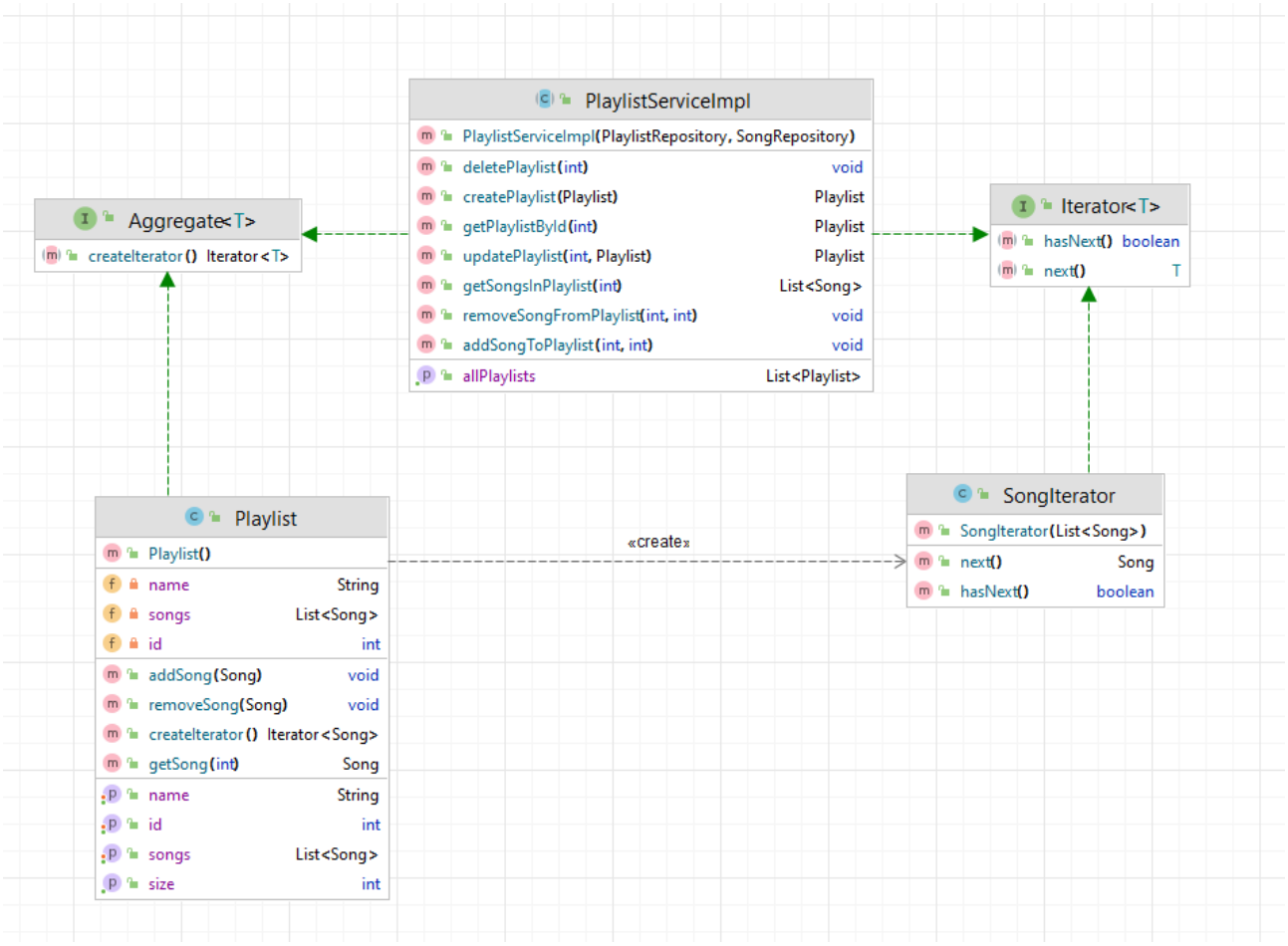


Рисунок 7- діаграма класів

Проблема, яку допоміг вирішити шаблон Iterator:

До використання цього патерну, логіка доступу до елементів колекції була б пов'язана з деталями її реалізації, і це могло б призвести до складнощів у підтримці та розширюванні коду. Кожного разу, коли потрібно було пройти по колекції пісень, доводилось би реалізовувати власну логіку ітерації, що призвело б до дублювання коду та погіршення його якості.

Використовуючи патерн Iterator, вдалося:

- Інкапсулювати логіку проходження по елементах колекції (піснях) у окремий клас ітератора, зробивши код чистішим і зрозумілішим.
- Забезпечити єдиний спосіб доступу до елементів плейлиста, незалежно від того, як вони зберігаються.
- Легко додавати нові способи ітерації (наприклад, у випадку зміни структури даних або додавання нових функціональностей).

Переваги застосування патерну Iterator:

1. **Інкапсуляція логіки ітерації:** Логіка проходження по колекції винесена в окремий клас, що робить код більш організованим.

2. **Полегшена підтримка і розширення:** Нові способи доступу до елементів можна легко додавати, змінюючи лише клас ітератора, не зачіпаючи інші частини коду.
3. **Зручне використання:** Завдяки патерну, можна легко реалізувати різні способи доступу до елементів колекції, наприклад, ітерацію у зворотному порядку, що підвищує гнучкість системи.

Висновок: У даній лабораторній роботі я реалізувала патерн проєктування Iterator для музичного плеєра, що дозволило створити ефективний механізм доступу до пісень у плейлисті без необхідності розкриття внутрішньої структури колекції. Завдяки цьому патерну, логіка ітерації була винесена в окремий клас, що значно спростило підтримку та розширення коду. Переваги застосування Iterator включають інкапсуляцію логіки проходження по елементах, гнучкість у використанні та зручність додавання нових можливостей. Реалізація цього патерну зробила систему більш масштабованою та організованою, покращивши її підтримуваність у майбутньому.