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 №7

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

Варіант 1

Виконав
студент групи ІА – 13:
Вознюк Максим

Перевірив:
Мягкий М. Ю

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для

досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Варіант:

..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.

Хід роботи

Паттерн Фасад(Facade)

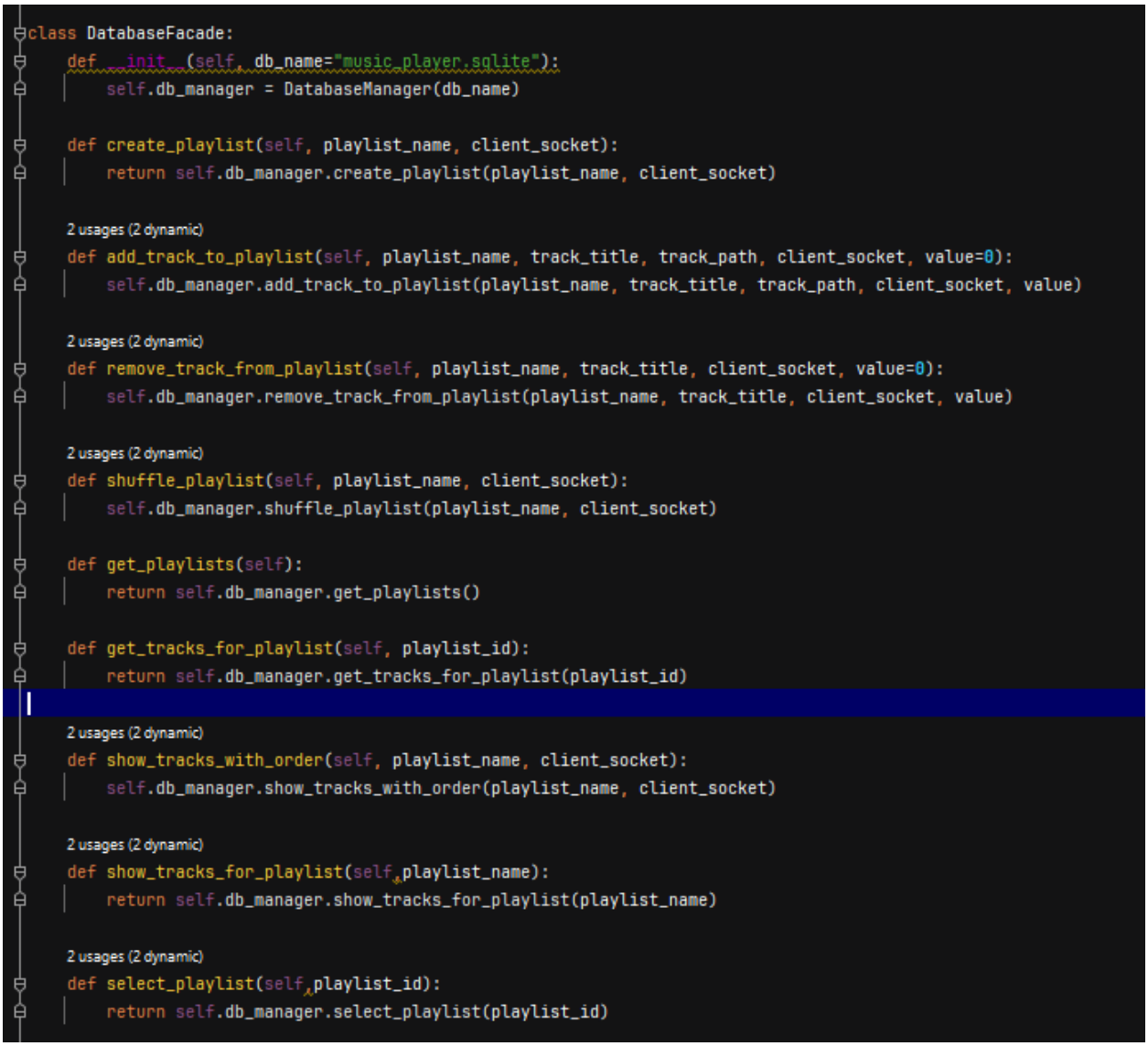
Фасад - це структурний патерн проєктування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

DatabaseFacade виступає як Facade, що приховує деталі роботи з базою даних за єдиною точкою входу.

Такий підхід має кілька переваг:

1) Спрощення інтерфейсу: Клієнтський код взаємодіє лише з DatabaseFacade, не звертаючись безпосередньо до DatabaseManager. Це дозволяє спростити інтерфейс та зробити його більш зрозумілим для користувачів класу.

2) Ізоляція змін в підсистемі: Якщо буде прийняте рішення змінити внутрішню реалізацію бази даних (наприклад, перейти з SQLite на іншу базу даних),не потрібно буде змінювати клієнтський код. Просто доведеться змінити код в DatabaseManager без впливу на зовнішній інтерфейс.
3) Зменшення залежностей: Клієнтський код залежить лише від DatabaseFacade, і не потрібно знати деталей роботи бази даних.



```
class DatabaseFacade:
    def __init__(self, db_name="music_player.sqlite"):
        self.db_manager = DatabaseManager(db_name)

    def create_playlist(self, playlist_name, client_socket):
        return self.db_manager.create_playlist(playlist_name, client_socket)

    2 usages (2 dynamic)
    def add_track_to_playlist(self, playlist_name, track_title, track_path, client_socket, value=0):
        self.db_manager.add_track_to_playlist(playlist_name, track_title, track_path, client_socket, value)

    2 usages (2 dynamic)
    def remove_track_from_playlist(self, playlist_name, track_title, client_socket, value=0):
        self.db_manager.remove_track_from_playlist(playlist_name, track_title, client_socket, value)

    2 usages (2 dynamic)
    def shuffle_playlist(self, playlist_name, client_socket):
        self.db_manager.shuffle_playlist(playlist_name, client_socket)

    def get_playlists(self):
        return self.db_manager.get_playlists()

    def get_tracks_for_playlist(self, playlist_id):
        return self.db_manager.get_tracks_for_playlist(playlist_id)

    2 usages (2 dynamic)
    def show_tracks_with_order(self, playlist_name, client_socket):
        self.db_manager.show_tracks_with_order(playlist_name, client_socket)

    2 usages (2 dynamic)
    def show_tracks_for_playlist(self,playlist_name):
        return self.db_manager.show_tracks_for_playlist(playlist_name)

    2 usages (2 dynamic)
    def select_playlist(self,playlist_id):
        return self.db_manager.select_playlist(playlist_id)
```