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
**Технології розроблення програмного забезпечення**
«Патерни проектування.»

Виконав:
Студент групи ІА-32
Костінський Макарій

Перевірив:
Мягкий Михайло Юрійович

# Зміст



**Тема:** Патерни проектування.

**Мета:** Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

## Теоретичні відомості

Фасад (Facade): Структурний патерн, який передбачає створення єдиного способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Він надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

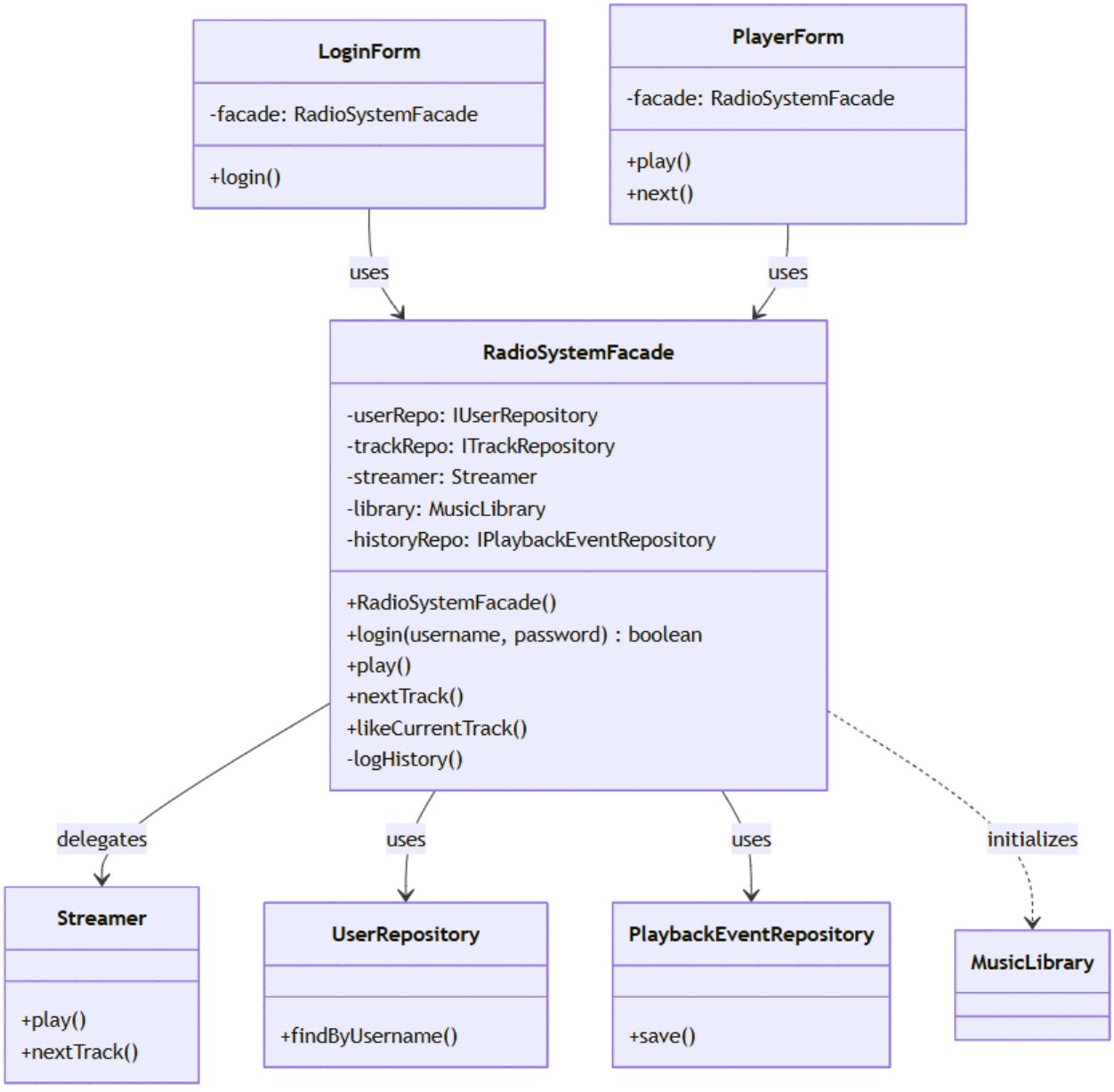
Посередник (Mediator): Поведінковий патерн, який використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Це дозволяє зменшити зв'язність між класами.

Міст (Bridge): Структурний патерн, що використовується для поділу інтерфейсу і його реалізації. Це дозволяє змінювати абстракцію та реалізацію незалежно одна від одної.

Шаблонний метод (Template Method): Поведінковий патерн, який дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам.

## Хід роботи

### Діаграма класів реалізації патерну «Facade»




Для спрощення взаємодії користувацького інтерфейсу (UI) з логікою системи "Online Radio Station" було реалізовано патерн Facade. Створено клас RadioSystemFacade, який інкапсулює роботу з репозиторіями (User, Track, Playlist, History), бібліотекою музики та стрімером.

Тепер UI-форми не створюють та не керують цими об'єктами напряму, а лише викликають високорівневі методи Фасаду.

**Фрагменти коду по реалізації шаблону**

Клас RadioSystemFacade: Надає прості методи для UI та приховує складну логіку (наприклад, логування історії при відтворенні).

```
package ua.kpi.iasa.onlineradio.facade;

import ua.kpi.iasa.onlineradio.data.*;
import ua.kpi.iasa.onlineradio.models.*;
import ua.kpi.iasa.onlineradio.repositories.*;

import java.util.Optional;

public class RadioSystemFacade {
    // Підсистеми, які ми приховуємо від UI
    private final IUserRepository userRepo;
    private final ITrackRepository trackRepo;
    private final IPlaylistRepository playlistRepo;
    private final IPlaybackEventRepository historyRepo;

    private final MusicLibrary library;
    private final Streamer streamer;
    private User currentUser;

    public RadioSystemFacade() {
        // Ініціалізація всіх підсистем (приховуємо складність створення)
        this.trackRepo = new TrackRepository();
        this.userRepo = new UserRepository();
        this.playlistRepo = new PlaylistRepository();
        this.historyRepo = new PlaybackEventRepository();

        this.library = new MusicLibrary(trackRepo);
        this.streamer = new Streamer();

        setupInitialData();
    }

    // --- Методи для Автентифікації ---
    public boolean login(String username, String password) {
        Optional<User> userOpt = userRepo.findByUsername(username);
        if (userOpt.isPresent() &&
userOpt.get().getPasswordHash().equals(password)) {
            this.currentUser = userOpt.get();
            System.out.println("Facade: Користувач " + currentUser.getUsername()
+ " увійшов у систему.");
            return true;
        }
        return false;
    }

    public User getCurrentUser() {
        return currentUser;
    }

    // --- Методи для Керування Відтворенням ---
    public void setPlaylist(int playlistId) {

playlistRepo.findById(playlistId).ifPresent(streamer::setActivePlaylist);
    }

    public void changePlaybackMode(IterationMode mode) {
        // Отримуємо поточний плейлист зі стрімера (або через репозиторій)
        // Для прикладу припустимо, що ми знаємо ID активного плейлиста або
```

```
додали геттер в Streamer
        // Тут ми просто оновимо перший плейлист для демонстрації
        playlistRepo.findById(1).ifPresent(p -> {
            p.setMode(mode);
            // Перезавантажуємо плейлист у стрімер, щоб застосувати новий
ітератор
            streamer.setActivePlaylist(p);
        });
    }

    public void play() {
        streamer.play();
        logHistory();
    }

    public void nextTrack() {
        streamer.nextTrack();
        logHistory();
    }

    public void likeCurrentTrack() {
        Track current = streamer.getCurrentTrack();
        if (current != null && currentUser != null) {
            current.addLike(currentUser);
            System.out.println("Facade: Лайк додано для '" + current.getTitle()
+ "'");
        }
    }

    public Track getCurrentTrack() {
        return streamer.getCurrentTrack();
    }

    // --- Внутрішні методи ---

    private void logHistory() {
        Track current = streamer.getCurrentTrack();
        if (current != null && currentUser != null) {
            PlaybackEvent event = new PlaybackEvent(currentUser.getId(),
current.getId());
            historyRepo.save(event);
        }
    }

    private void setupInitialData() {
        userRepo.save(new Administrator(0, "admin", "admin"));
        userRepo.save(new User(0, "listener", "1234"));

        Track t1 = new Track(0, "Bohemian Rhapsody", "Queen",
"music/queen.mp3");
        Track t2 = new Track(0, "Smells Like Teen Spirit", "Nirvana",
"music/nirvana.mp4");
        Track t3 = new Track(0, "Shape of You", "Ed Sheeran", "music/ed.mp3");
        Track t4 = new Track(0, "Believer", "Imagine Dragons",
"music/believer.mp3");

        trackRepo.save(t1);
        trackRepo.save(t2);
        trackRepo.save(t3);
        trackRepo.save(t4);

        Playlist p1 = new Playlist(0, "Best Rock");
        p1.addTrack(t1);
        p1.addTrack(t2);
        p1.addTrack(t3);
        p1.addTrack(t4);
```

```
        playlistRepo.save(p1);
    }
}
```

Клас Main (Клієнт): Демонструє, наскільки спростилася ініціалізація системи.

```
package ua.kpi.iasa.onlineradio;

import ua.kpi.iasa.onlineradio.facade.RadioSystemFacade;
import ua.kpi.iasa.onlineradio.ui.LoginForm;

import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {}

        System.out.println("--- Запуск Online Radio System (with Facade) ---");

        // Створюємо фасад. Він сам ініціалізує БД, бібліотеку і стрімер.
        RadioSystemFacade radioFacade = new RadioSystemFacade();

        // Встановлюємо активний плейлист за замовчуванням через фасад
        radioFacade.setPlaylist(1);

        SwingUtilities.invokeLater(() -> {
            // Передаємо лише фасад
            new LoginForm(radioFacade).setVisible(true);
        });
    }
}
```

**Висновок:**

У ході виконання лабораторної роботи було досліджено та практично застосовано структурний патерн проєктування Фасад для оптимізації архітектури програмної системи "Online Radio Station". Реалізація класу RadioSystemFacade дозволила інкапсулювати складну логіку ініціалізації та взаємодії між підсистемами репозиторіїв користувачів, треків, плейлистів та модулем відтворення музики, що значно спростило клієнтський код у класах інтерфейсу користувача. Завдяки впровадженню цього патерну було досягнуто зниження зв'язності між компонентами системи, забезпечено централізоване керування бізнес-логікою, зокрема процесами автентифікації та логування історії прослуховування, а також підвищено загальну надійність та зручність супроводу програмного коду за рахунок приховування деталей реалізації від зовнішнього середовища.