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 №9

«Взаємодія компонентів системи»

Виконав:
студент групи ІА–32
Лось Ярослав

Київ 2025

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Зміст

Теоретичні відомості 3
Хід роботи 5
1. Загальний опис роботи 5
2. Опис класів програмної системи 6
2.1 Пакет Common (Middleware) 6
2.2 Пакет Server 7
2.3 Пакет Client 10
3. Опис результатів коду 13
4. Діаграма класів 15
Висновки 16
Контрольні запитання 16

Теоретичні відомості

Клієнт-серверна архітектура

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

У такому варіанті використання майже все навантаження лягає на сервер або групу серверів.

Перевагою таких моделей є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою.

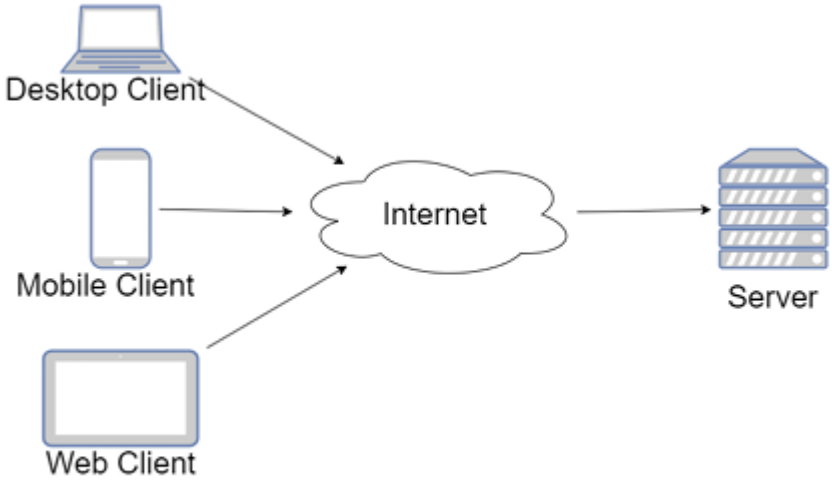

Рисунок 9.1. Клієнт-серверна архітектура

Товстий клієнт – антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випа-

дках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API. З одного боку більшу частину логіки вони відпрацьовують на клієнтський стороні, за рахунок чого зменшується серверне навантаження. Також оновлення простіше ніж для товстих клієнтів. Але такі застосунки не працюють, якщо сервер не доступний.

Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина, серверна частина.

Оскільки велика частина даних загальна (класи, використовувані системою), їх прийнято виносити в загальну частину (middleware) системи.

Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

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. Загальний опис роботи

Метою роботи є реалізація розподіленої системи музичного програвача. У локальній версії (монолітній) вся логіка і дані знаходяться в одній програмі. У клієнт-серверній архітектурі ми розділимо систему на три частини:

1. **Server (Сервер):** Зберігає музичну бібліотеку (базу даних пісень), обробляє запити від клієнтів та керує потоками підключень. Сервер є "товстим" з точки зору даних.
2. **Client (Клієнт):** Відповідає за інтерфейс користувача (консольне меню), відправку команд на сервер ("Отримати список", "Грати пісню") та відображення відповідей. Це реалізація "тонкого" клієнта (основна логіка пошуку та зберігання - на сервері).
3. **Middleware (Загальна частина):** Містить класи даних, які відомі обом сторонам. Це дозволяє передавати об'єкти через мережу.

Для зв'язку використаємо протокол TCP/IP через Java Sockets (ServerSocket та Socket). Обмін даними відбувається за допомогою серіалізації об'єктів (ObjectOutputStream).

2. Опис класів програмної системи

У розподіленій системі класи чітко поділяються на три групи: спільні дані (DTO), серверна логіка та клієнтський інтерфейс.

2.1 Пакет Common (Middleware)

Клас Song (DTO)

Опис:

Клас Song є фундаментальною одиницею даних, якою обмінюються клієнт і сервер. Оскільки ці дві програми працюють у різних віртуальних машинах Java (JVM), ми не можемо просто передати посилання на об'єкт. Об'єкт необхідно перетворити на потік байтів для передачі мережею.

Ключові характеристики:

- Інтерфейс Serializable: Це маркерний інтерфейс, який дозволяє механізму Java Serialization перетворити об'єкт у бінарний формат. Без цього при спробі відправки виникне виключення NotSerializableException.
- Поле serialVersionUID: Унікальний ідентифікатор версії класу. Він гарантує, що клієнт і сервер використовують сумісні версії класу Song (якщо на сервері додати нове поле, а на клієнті ні - версії не співпадуть і десеріалізація не вдасться).

```java
public class Song implements Serializable {
    no usages
    private static final long serialVersionUID = 1L;
    3 usages
    private String title;
    2 usages
    private String artist;

    3 usages
    public Song(String title, String artist) {
        this.title = title;
        this.artist = artist;
    }

    no usages
    public String getTitle() {
        return title;
    }

    @Override
    public String toString() {
        return title + " - " + artist;
    }
}
```

Рисунок 1 – Код класу Song

2.2 Пакет Server

Клас MusicServer

Опис:

Цей клас є точкою входу для серверної частини. Його головна задача - слухати мережевий порт і приймати вхідні з'єднання, але не обробляти їх самостійно.

Ключові характеристики

- ServerSocket: Створює сокет, прив'язаний до конкретного порту (5000). Це двері, через які заходять клієнти.
- Нескінченний цикл (while(true)): Сервер повинен працювати постійно, очікуючи нових підключень.

- Блокуючий виклик accept(): Програма зупиняється на цьому рядку і спить, поки не постукає клієнт. Як тільки це стається, метод повертає об'єкт Socket для зв'язку саме з цим клієнтом.
- Багатопотоковість: Для кожного підключеного клієнта створюється новий об'єкт ClientHandler і запускається в окремому потоці. Якщо цього не зробити, сервер зможе обслуговувати тільки одного користувача одночасно, а інші чекатимуть у черзі.

```
public class MusicServer {
    2 usages
    private static final int PORT = 5000;
    4 usages
    private static List<Song> musicLibrary = new ArrayList<>();

    static {
        musicLibrary.add(new Song( title: "Shape of You", artist: "Ed Sheeran"));
        musicLibrary.add(new Song( title: "Blinding Lights", artist: "The Weeknd"));
        musicLibrary.add(new Song( title: "Bohemian Rhapsody", artist: "Queen"));
    }

    public static void main(String[] args) {
        System.out.println("Сервер запущено на порту " + PORT);
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Новий клієнт підключився: " + clientSocket.getInetAddress());
                new Thread(new ClientHandler(clientSocket, musicLibrary)).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Рисунок 2 – Код класу MusicServer

Клас ClientHandler

Опис:

Цей клас виконує всю роботу з обслуговування конкретного клієнта. Він реалізує інтерфейс Runnable, що дозволяє йому працювати паралельно з іншими обробниками.

Логіка роботи:

1. Ініціалізація потоків: Отримує InputStream та OutputStream від сокета для читання та запису даних.
2. Цикл обробки: Входить у цикл, очікуючи команди від клієнта.
3. Десеріалізація команд: Читає об'єкт (у нашому випадку рядок String) з потоку.
4. Маршрутизація (Switch/If): Аналізує текст команди:
    a. Якщо "LIST" - серіалізує весь список пісень (ArrayList<Song>) і відправляє його клієнту.
    b. Якщо "PLAY:..." - витягує назву пісні та імітує початок трансляції.
    c. Якщо "EXIT" - розриває з'єднання та завершує потік.

Код класу ClientHandler

```
public class ClientHandler implements Runnable {
    private Socket socket;
    private List<Song> library;

    public ClientHandler(Socket socket, List<Song> library) {
        this.socket = socket;
        this.library = library;
    }

    @Override
    public void run() {
        try (
                ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream in = new
ObjectInputStream(socket.getInputStream())
        ) {
            while (true) {
                String command = (String) in.readObject();

                if ("LIST".equalsIgnoreCase(command)) {
                    out.writeObject(library);
                    out.flush();
                }
```

```
                else if (command.startsWith("PLAY:")) {
                    String songTitle = command.split(":")[1];
                    out.writeObject("STREAMING_DATA_FOR: " + songTitle);
                    out.flush();
                }
                else if ("EXIT".equalsIgnoreCase(command)) {
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Клієнт відключився.");
        }
    }
}
```

2.3 Пакет Client

Клас MusicClient

Опис:

Цей клас приховує від решти програми складність роботи з мережею. Для основного коду клієнта (ClientApp) він виглядає як звичайний локальний сервіс.

Ключові характеристики:

- Інкапсуляція сокетів: Зберігає стан з'єднання (Socket, ObjectOutputStream, ObjectInputStream).
- Синхронні запити: Методи getSongList() та playSong() працюють синхронно. Вони відправляють запит, блокуються до отримання відповіді від сервера, і лише потім повертають результат.
- Обробка типів: Метод getSongList() знає, що сервер поверне саме List<Song>, і виконує приведення типів (Casting) після десеріалізації.

Код класу MusicClient

```
public class MusicClient {
    private Socket socket;
    private ObjectOutputStream out;
    private ObjectInputStream in;

    public void connect(String host, int port) throws IOException {
        socket = new Socket(host, port);
        out = new ObjectOutputStream(socket.getOutputStream());
        in = new ObjectInputStream(socket.getInputStream());
    }

    public List<Song> getSongList() throws IOException, ClassNotFoundException {
        out.writeObject("LIST");
        out.flush();
        return (List<Song>) in.readObject();
    }

    public String playSong(String title) throws IOException,
ClassNotFoundException {
        out.writeObject("PLAY:" + title);
        out.flush();
        return (String) in.readObject();
    }

    public void disconnect() throws IOException {
        out.writeObject("EXIT");
        socket.close();
    }
}
```

Клас ClientApp

Опис:

Консольний інтерфейс користувача. Він відповідає за взаємодію з людиною.

- Обробка вводу: Використовує Scanner для зчитування вибору користувача.
- Виклик бізнес-логіки: Делегує виконання мережевих запитів об'єкту MusicClient.

Код класу ClientApp

```
public class ClientApp {
    public static void main(String[] args) {
        MusicClient client = new MusicClient();
        try {
            System.out.println("Підключення до сервера");
            client.connect("localhost", 5000);
            System.out.println("Підключено!");

            Scanner scanner = new Scanner(System.in);
            while (true) {
                System.out.println("\n- MENU -");
                System.out.println("1. Список пісень");
                System.out.println("2. Грати пісню");
                System.out.println("3. Вихід");
                System.out.print("Вибір: ");

                String choice = scanner.nextLine();

                if ("1".equals(choice)) {
                    List<Song> songs = client.getSongList();
                    System.out.println("Пісні на сервері:");
                    for (Song s : songs) System.out.println(" - " + s);
                }
                else if ("2".equals(choice)) {
                    System.out.print("Введіть назву: ");
                    String title = scanner.nextLine();
                    String response = client.playSong(title);
                    System.out.println("Сервер: " + response);
                }
                else if ("3".equals(choice)) {
                    client.disconnect();
                    break;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Опис результатів коду

Спочатку запустимо серверну частину програми (MusicServer). У консолі з'явилося повідомлення про те, що сервер успішно стартував і очікує підключень на порту 5000. Після цього запустимо клієнтський додаток (ClientApp). Сервер одразу зафіксував нове підключення, вивівши повідомлення про нового клієнта.

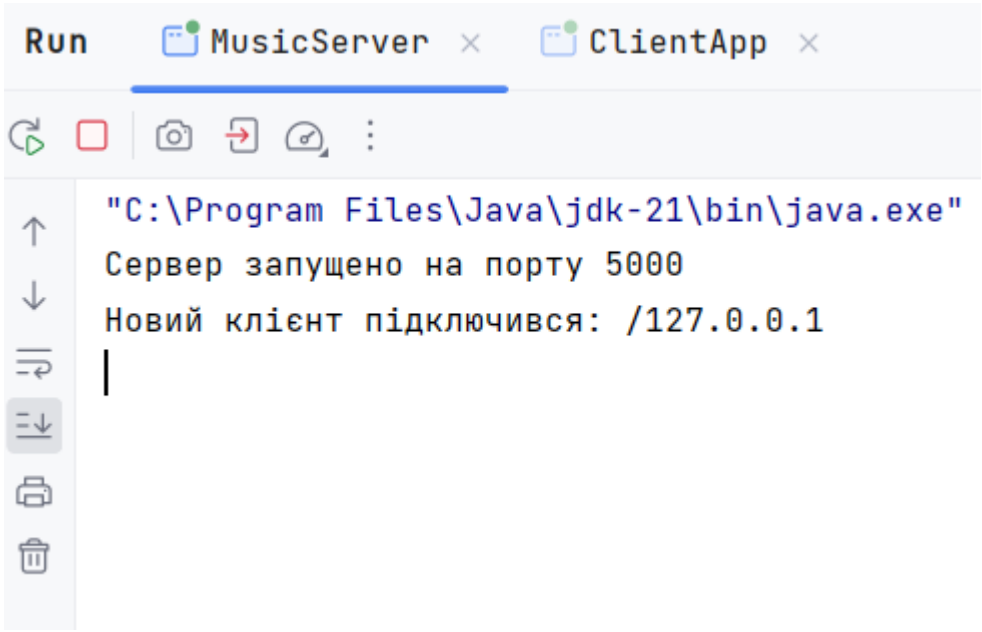

Рисунок 3 – Результат виконання програми у консолі сервера

Далі перевіримо основний функціонал через меню клієнта:

1. При виборі пункту **"1. Список пісень"** клієнт відправив запит на сервер. У відповідь сервер надіслав список треків, які зберігаються у його базі даних. Клієнт успішно прийняв цей список і вивів його на екран (назви пісень та виконавців). Це підтвердило, що передача складних об'єктів через мережу працює коректно.
2. При виборі пункту **"2. Грати пісню"** ми ввели назву треку "Shape of You". Клієнт сформував команду і відправив її серверу. Сервер обробив запит і повернув текстове підтвердження про початок трансляції, яке відобразилося у консолі клієнта.

3. При виборі пункту **"3. Вихід"** клієнт розірвав з'єднання, а сервер коректно закрив потік обслуговування цього користувача, продовжуючи чекати на нових.

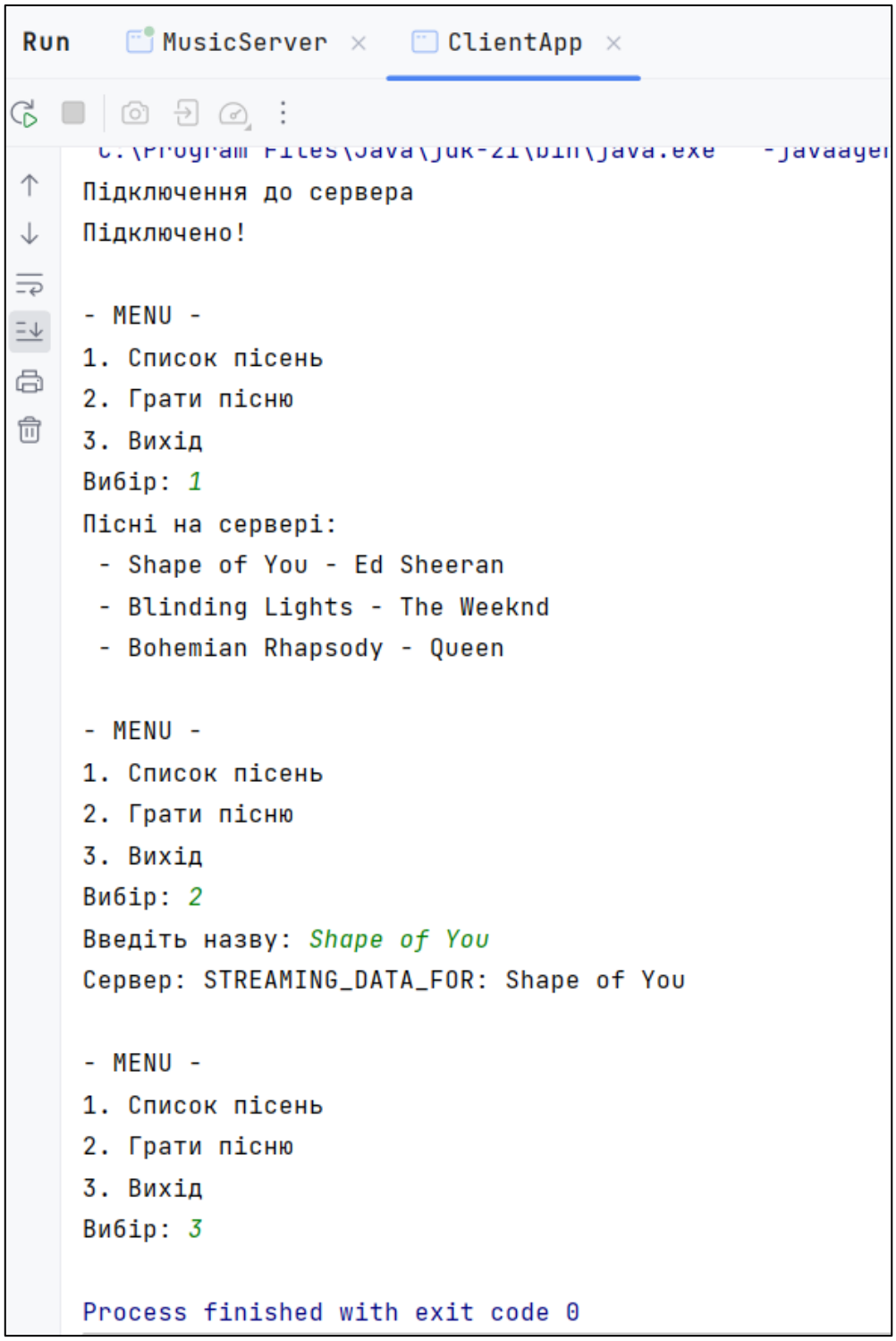

Рисунок 4 – Результат виконання програми у консолі клієнта

4. Діаграма класів

Для візуалізації архітектури розробленої системи та взаємозв'язків між її компонентами побудуємо UML-діаграму класів.

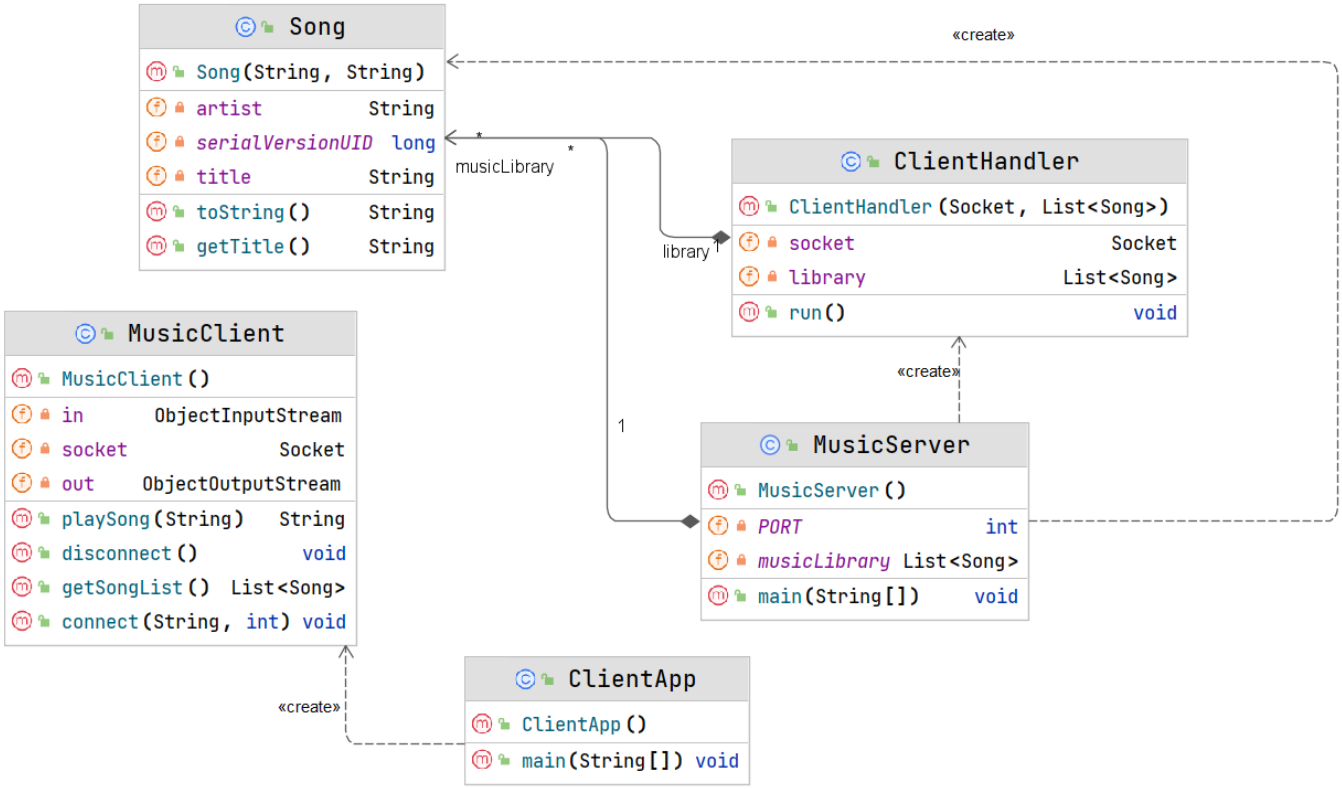

Рисунок 5 – Діаграма класів

Висновки

В ході виконання лабораторної роботи було розроблено розподілену програмну систему для музичного програвача, побудовану за принципами клієнт-серверної архітектури. Основною метою було розділити логіку зберігання даних та інтерфейс користувача на дві незалежні програми, що взаємодіють через мережу.

Для реалізації цього завдання було використано технологію сокетів (Java Sockets), яка дозволила встановити надійне TCP-з'єднання між компонентами системи. Серверна частина була реалізована як багатопотоковий додаток, здатний обслуговувати кілька клієнтів, тоді як клієнтська частина виконувала роль "тонкого клієнта", делегуючи основну роботу серверу. Також було успішно реалізовано передачу серіалізованих об'єктів, що дозволило обмінюватися даними про пісні між різними частинами системи. Виконана робота продемонструвала переваги розподілених систем, такі як централізоване управління даними та легкість масштабування.

Контрольні запитання

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура - це модель взаємодії в розподіленій системі, де завдання або навантаження розподіляються між постачальниками ресурсу або послуги, які називаються серверами, і замовниками послуг, які називаються клієнтами.

- Клієнт ініціює з'єднання і надсилає запит.
- Сервер пасивно очікує на з'єднання, обробляє запит і повертає відповідь. Взаємодія зазвичай відбувається через комп'ютерну мережу.

2. Розкажіть про сервіс-орієнтовану архітектуру.

SOA (Service-Oriented Architecture) - це архітектурний стиль проектування програмного забезпечення, який базується на використанні сервісів (служб). Сервіси - це автономні компоненти, які виконує певні бізнес-задачі та взаємодіють між собою через мережу за допомогою стандартизованих протоколів. Головна мета SOA - повторне використання компонентів, гнучкість бізнес-процесів та легка інтеграція різнорідних систем (наприклад, написаних різними мовами). Часто в SOA використовується ESB (Enterprise Service Bus) - шина даних для маршрутизації повідомлень.

3. Якими принципами керується SOA?

Основні принципи SOA:

1. Стандартизований контракт сервісу: Сервіси взаємодіють через чітко визначені інтерфейси (WSDL, Swagger/OpenAPI).
2. Слабка зв'язність (Loose Coupling): Сервіси мінімально залежать один від одного.
3. Абстракція: Внутрішня логіка сервісу прихована від клієнта.
4. Повторне використання: Сервіси проектуються так, щоб їх можна було використовувати в різних бізнес-процесах.
5. Автономність: Сервіс самостійно керує своїм середовищем виконання.
6. Відсутність стану (Statelessness): Сервіс не повинен зберігати інформацію про стан між запитами (бажано).
7. Можливість виявлення (Discoverability): Сервіси мають бути зареєстровані в реєстрі, щоб їх можна було знайти.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють шляхом обміну повідомленнями.

- Пряма взаємодія: Один сервіс викликає інший через HTTP (REST) або SOAP.

- Через посередника (ESB): Сервіс надсилає повідомлення на шину (Enterprise Service Bus), яка трансформує дані та перенаправляє їх потрібному отримувачу.
- Взаємодія може бути синхронною (запит-відповідь) або асинхронною (через черги повідомлень).

5. Як розробники взнають про існуючі сервіси і як робити до них запити?

Для цього існує механізм Service Discovery (Виявлення сервісів) та Service Registry (Реєстр сервісів).

1. Як дізнатися: Розробник звертається до Реєстру сервісів (каталогу), де описані всі доступні сервіси та їхні адреси.
2. Як робити запит: Розробник вивчає контракт сервісу (наприклад, файл WSDL для SOAP або документацію Swagger/OpenAPI для REST), де описані доступні методи, структура запиту (вхідні дані) та структура відповіді.

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги:

- Централізація: Всі дані зберігаються в одному місці (на сервері), що полегшує контроль доступу, захист та створення резервних копій.
- Простота обслуговування: Оновлення логіки на сервері автоматично стає доступним для всіх клієнтів.
- Масштабованість сервера: Сервер можна посилити потужнішим "залізом".

Недоліки:

- Вразливість (Single Point of Failure): Якщо сервер виходить з ладу, робота зупиняється у всіх клієнтів.
- Залежність від мережі: Без інтернету робота неможлива.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

В P2P кожен вузол є одночасно і клієнтом, і сервером.

Переваги:

- Відмовостійкість: Немає єдиного центру, вихід з ладу одного вузла не зупиняє роботу мережі.
- Масштабованість: Кожен новий користувач додає в мережу свої ресурси (диск, процесор).
- Економічність: Не потрібні дорогі виділені сервери.

Недоліки:

- Децентралізація: Важко керувати мережею, оновлювати ПЗ та контролювати дані.
- Безпека: Важче захистити дані, оскільки вони розкидані по багатьох комп'ютерах.
- Нестабільність: Ресурси можуть зникати, якщо користувачі вимикають свої комп'ютери.

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура — це підхід до розробки, при якому додаток будується як набір невеликих, незалежних сервісів.

- Кожен мікросервіс відповідає за одну конкретну бізнес-функцію (Single Responsibility).
- Він має власну базу даних.
- Він може бути написаний на іншій мові програмування.
- Він розгортається незалежно від інших (можна оновити один сервіс, не перезапускаючи всю систему).

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

Використовуються легковагі протоколи:

1. Синхронні (запит-відповідь):
    - HTTP/REST: Найпоширеніший варіант (JSON).
    - gRPC: Високопродуктивний протокол від Google (на основі Protobuf).
2. Асинхронні (подієво-орієнтовані):

- Протоколи черг повідомлень, такі як AMQP (RabbitMQ) або протоколи брокерів потоків, як у Apache Kafka.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

У контексті монолітного додатку це називається "Шар сервісів" (Service Layer). Це просто спосіб організації коду (класів) всередині однієї програми. Справжня SOA (і мікросервіси) передбачає фізичне розділення компонентів. Тобто сервіси - це окремі процеси, які можуть працювати на різних серверах і спілкуються через мережу, а не просто викликами методів всередині однієї JVM чи .NET Runtime.