Full Stack Application Development
Software Workshop 2

Assignment 3:
Database Client-Server

Set by
Ana Stroescu

Submission Deadline:
Monday 16th May 2022 4pm BST

Use Canvas for the definitive deadlines

This assignment is worth
25 % (**twenty-five** percent)
of the overall course mark

Overview

Table of contents

1 Introduction

This assignment will assess your JDBC and Java Networks skills. You are required to implement a Multi Client–Server application using the TCP protocol, where the user continuously inputs the name of an artist and the server replies with the number of albums for that particular artist, until the user types 'stop'. The SQL query is a simple inner join executed on the Music database.

Descriptions of the overall requirements are given in §4 Coding tasks and in the skeleton.

2 Marking scheme

Question	Description	Marks
1	`Database.java`	50
2	`Client.java`	15
3	`Server.java`	15
4	`ClientHandler.java`	20
	Total	100

> **!** We are going to perform several tests on your code and most of your classes will be tested in isolation. Therefore, it is essential to leave the methods and the classes signatures in the skeleton unchanged. The text messages that should be printed to the console are also included in the skeleton code, please do not modify them, as your console output should match the sample output examples given in §5 Sample Outputs.

5

10

3 Setting up the database

For this assignment you will use the 'Music' database that you have cre-
ated in Week 10. Please use the instructions and the tutorial video on Canvas
for creating the database and for inserting data into the tables. The two files   15
you need can be downloaded from Canvas and are called `music_create.sql`
and `music_data.sql`. Import `music_create.sql` first and then import `mu-`
`sic_data.sql`.

The command to import these files is as follows. Remember, you need to
state the path on your own computer where you put the files. You might also   20
need to put the path in quote marks.

```
\i /your_path/music_create.sql
\i /your_path/music_data.sql
```

Once you have imported the tables, you can list them by performing the
following command in PostgreSQL: 25

```
\dt
```

Note that this will list all of the tables you may have created in this database.
Run SQL commands to view the contents of each table by doing 'SELECT *
FROM *tablename*;' (replacing *tablename* with the name of the table you want to
look at.)   30

4 Coding tasks

Only move to this stage when you have set up the database, as above. For
this assignment you will create a Java Client-Server application **using TCP** that
performs a query on the database and returns the result to the client.

The following sections describe the classes that you can find in the assign-   35
ment pack. You must add the code to create the functionality required for each
class.

4.1 `Server.java`

The `Server.java` implements a basic multi-threaded server that continu-
ously listens for connections and creates a new thread for each client connec-   40
tion. Note that the client-server communication is continuous and messages are
exchanged until the user types 'stop'.

Functionality:

- Creates the server socket;

- Creates a `Database` object;

- Checks the database connectivity by calling the `establishDBConnection()` method from `Database.java`, which returns the connection status (boolean);

- Prints the appropriate text messages to the console (given in the skeleton), in the right order;

- Continuously listens for client requests and accepts multiple client connections;

- Assigns a `clientId` for each client that connects to the server. The `clientId` starts from 1 and must not be reassigned once used;

- Creates a `ClientHandler` instance and starts a new thread for each client.

4.2 `Client.java`

The `Client.java` is a basic client program that **continuously** reads the name of the artist from the console and prints the response from the server, until the user types 'stop'.

Functionality:

- Creates the client socket;

- Creates I/O streams to read data from the server and console and send data to the server;

- Reads text from the console continuously, until the user types 'stop';

- Sends messages (artist names) to the server;

- Receives responses from the server (number of albums);

- Prints the appropriate text messages to the console (given in the skeleton), in the right order;

- Cleans up the environment, closes the socket and the I/O streams.

4.3 `ClientHandler.java`

The `ClientHandler` class provides the client-server interaction in the run() method. It implements `Runnable` and deals with the multiple client communication.

Functionality:

- Implements `Runnable`;

- Has a constructor;

- Implements the functionality of the threads in the `run()` method;

- Creates I/O streams to read/write data from/to the client;

- Continuously receives messages from the clients and replies back, until   80
  the user types 'stop';

- Requests the number of titles from the database by calling the `get-`
  `Titles()` method in the `Database.java` class and stores the result;

- Prints the appropriate text messages to the console (given in the skeleton),
  in the right order;   85

- Cleans up the environment, closes the socket and the I/O streams.

4.4 `Database.java`

The `Database.java` class deals with the JDBC part of the assignment and has
2 methods: `getTitles()` and `establishDBConnection()`.

Functionality: 90

- It must prevent SQL injections;

- `getTitles()` method executes a query and returns an integer, the num-
  ber of albums for the artist. This method is called in the `ClientHand-`
  `ler.java` class;

- `establishDBConnection()` method establishes a connection to the   95
  database and returns the status of the connection (boolean). The status
  must be *true* if the database connection is established within 5 seconds.
  This method is called in the `Server.java` class to check the database
  connectivity.

4.5 `Credentials.java`   100

The `Credentials.java` contains the credentials for the database (username,
password, url) and for the server connection (host address, port number). You
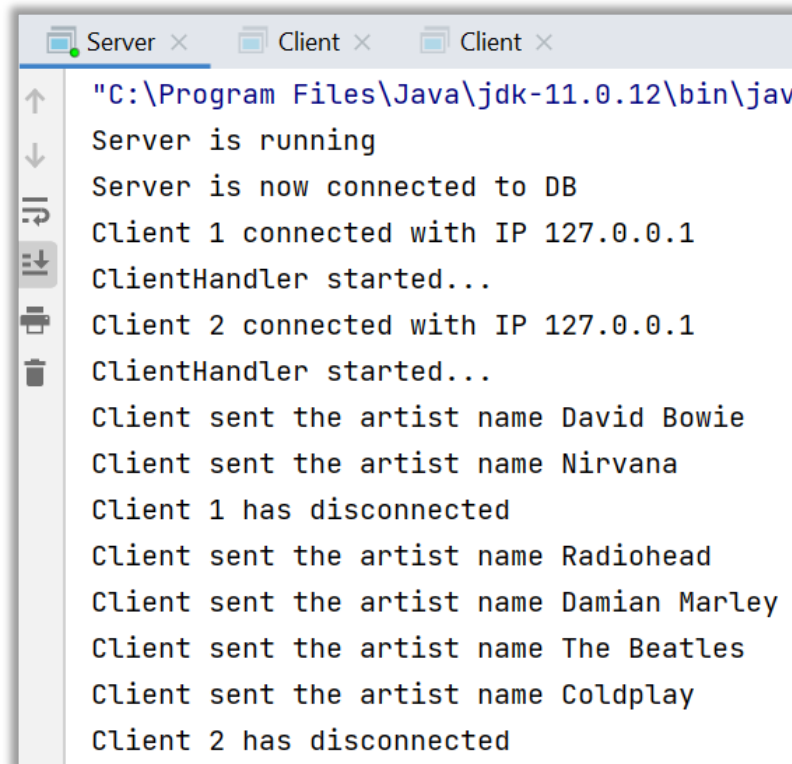must replace the credentials with your own.

Submitting this class is optional, as we are going to use our own credentials
for testing your code. Therefore, you can delete this class from the `src` folder of   105
your `.zip` file before submitting your assignment.

5   Sample Outputs

Note that the text messages printed to the console are included in the skeleton code, as we are not assessing your abilities to write strings in Java. Your outputs should match the sample outputs images below, as long as you do not modify the output strings in the skeleton.

Figure 1 shows a sample output for the Server.java class. The database connectivity is successful, two clients connect to the server and send messages.

Figure 1 Server output.

Figure 2 shows a sample output for the `Client.java` class. The client successfully connects to the server and sends artist names, until the user types 'stop'. The server replies with the number of album titles for each artist.

Figure 2 Client 1 output.

Figure 3 shows a sample output for a second client. Follow the instructions in Week 11 - Part 2 lecture on how to run multiple instances of the `Client.java` class in IntelliJ. Note that if the artist name is not found in the database, the server will return '0 records found'.
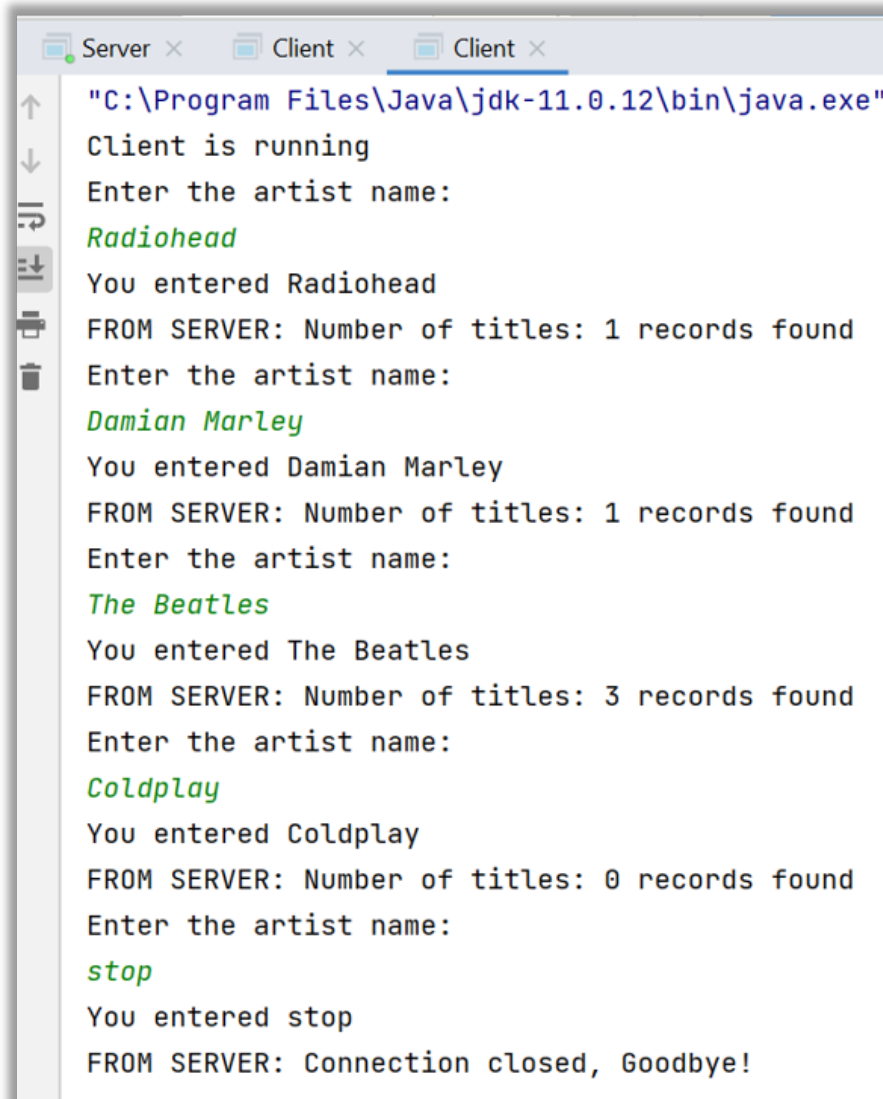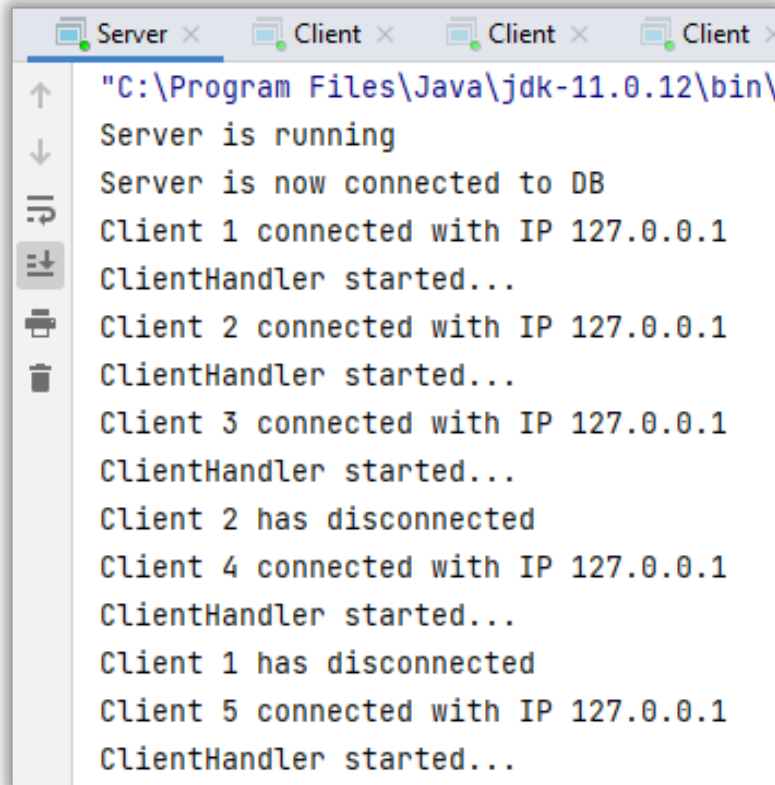
```
Server ×        Client ×        Client ×

"C:\Program Files\Java\jdk-11.0.12\bin\java.exe"
Client is running
Enter the artist name:
Radiohead
You entered Radiohead
FROM SERVER: Number of titles: 1 records found
Enter the artist name:
Damian Marley
You entered Damian Marley
FROM SERVER: Number of titles: 1 records found
Enter the artist name:
The Beatles
You entered The Beatles
FROM SERVER: Number of titles: 3 records found
Enter the artist name:
Coldplay
You entered Coldplay
FROM SERVER: Number of titles: 0 records found
Enter the artist name:
stop
You entered stop
FROM SERVER: Connection closed, Goodbye!
```

Figure 3 Client 2 output.

7

Figure 4 shows the output of the `Server.java` class for multiple client connections. Note that the `clientID` starts from 1 and must not be reassigned once used, i.e. if there are 3 clients connected to the server and the client with `clientID` 2 disconnects, the next client that connects will have `clientID` 4.

Figure 4 Server output for multiple client connections.

Figure 5 shows a sample output for the `Server.java` class, if the connection to the database failed (i.e the database password is wrong). The server will throw an exception and print 'DB connection fail, stopping.'

Figure 5 Server output when the database connectivity fails.

6 About the project

You are given a `.zip` file containing the IntelliJ project with the skeleton code that you have to complete: this a framework that lacks functionality. You must expand this skeleton to complete the assignment according to the instructions in §4 Coding tasks.

6.1 Obtaining the assignment project

Go to Canvas, then go to the FSAD/SW2 course, then to Assignment 3.[1] There is a link to a `.zip` file. Download and unpack this file to its own folder and move the unpacked directory (folder) somewhere sensible. The folder contains an IntelliJ project with the skeletons of the five classes described in §4 Coding tasks.

6.2 Loading the assignment project

Remember when you load this project into IntelliJ to open the directory itself rather than one of the files inside it. The `.java` files under the `src` folder contain a skeleton and method stubs for the required methods. Do not change the signatures of the classes and methods.

In your project you will also need to make available the external `jar` file, `postgresql-42.3.3.jar`, or the latest version. This file contains the drivers that you need to connect to a PostgreSQL database using Java. If you don't know how to do that, there are instructions on Canvas (week 10). In essence, you need to add the driver as an external jar file in your project. Do not try and 'import' this file. You also don't need to open it to view its contents.
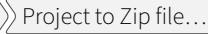
6.3 Allowed imports

The allowed imports are found in the skeleton code.

6.4 How to submit

1. `Build` ⟩ `Rebuild Project` and fix any compilation errors.

2. `File` ⟩ `Export` ⟩ `Project to Zip file…` and rename the resulting `.zip` file to
   `FSADAssign3_Givenname_Familyname_studentIDnumber.zip`
   If you officially have only one name then use `X` for the 'missing' name.

3. Check your `.zip` file contains everything it should under the `src` folder:
   `Client.java`, `Server.java`, `ClientHandler.java` and `Database.java`.

---

[1] Direct link: https://canvas.bham.ac.uk/courses/56081/assignments/343494

4. You may delete `Credentials.java` from the `src` folder of your `.zip` before uploading it on Canvas, as we are going to use our own credentials for testing your code.   160

5. Ensure you do not include a previous exported `.zip` file in the latest project export (a `.zip` inside a `.zip`) because this means you cannot guarantee the autotester will run the latest version of your code.

6. Upload the renamed `.zip` file to Canvas for the Java course Assignment 3.   165

7. Canvas will probably add some extra information to the end of the file-name you have uploaded: do not worry about this.

Failure to comply with the submission requirements will result in penalties.