8/06/2020

Java III

AT3 Project

Sean Boaden / 30010353

Contents

[Advanced Music Player 0](#_Toc43737151)

[Overview 0](#_Toc43737152)

[Demonstration / Marking Criteria 0](#_Toc43737153)

[The Player 0](#_Toc43737154)

[Marking Criteria 1](#_Toc43737155)

[Debugging 0](#_Toc43737156)

[Sorting 1](#_Toc43737157)

[Searching 3](#_Toc43737158)

[OpenCSV – writing to CSV files 0](#_Toc43737159)

[Sockets 1](#_Toc43737160)

[Unit Testing 3](#_Toc43737161)

[Server Module 3](#_Toc43737162)

[Client Module 4](#_Toc43737163)

# Advanced Music Player

## Overview

* To store music in a binary tree, I used the Java implementation of TreeSet<>
  + which is a red-black tree so it is self-balancing. I used the default constructor, meaning the elements are sorted in natural order.
* Merge sort, quick sort options in Settings window. Radio buttons will switch between the two.
  + User has option to sort by song name, duration, ascending and descending.
* IPC Mechanism to utilise is Sockets. A Client and Server module are included. The Server module must be running to allow the Client to connect and use the Player.
* Hashing techniques are performed on the Server side for client validation.
* User details are stored on the server side, in a LinkedList<Account> object. Serialized when the Server is closed, and deserialized when opened.
  + A default admin account is always present, use login: ***admin* – *admin*** [*user* – *pass*]
  + No plain-text passwords are stored. Passwords are stored as hash/salt combos.
* OpenCSV is used as a third-party library. To export playlist/audio information to a CSV file, click “Export to CSV” button in the Settings window.
* Help files included – help.txt file is opened when the user clicks “Help” button in Settings.

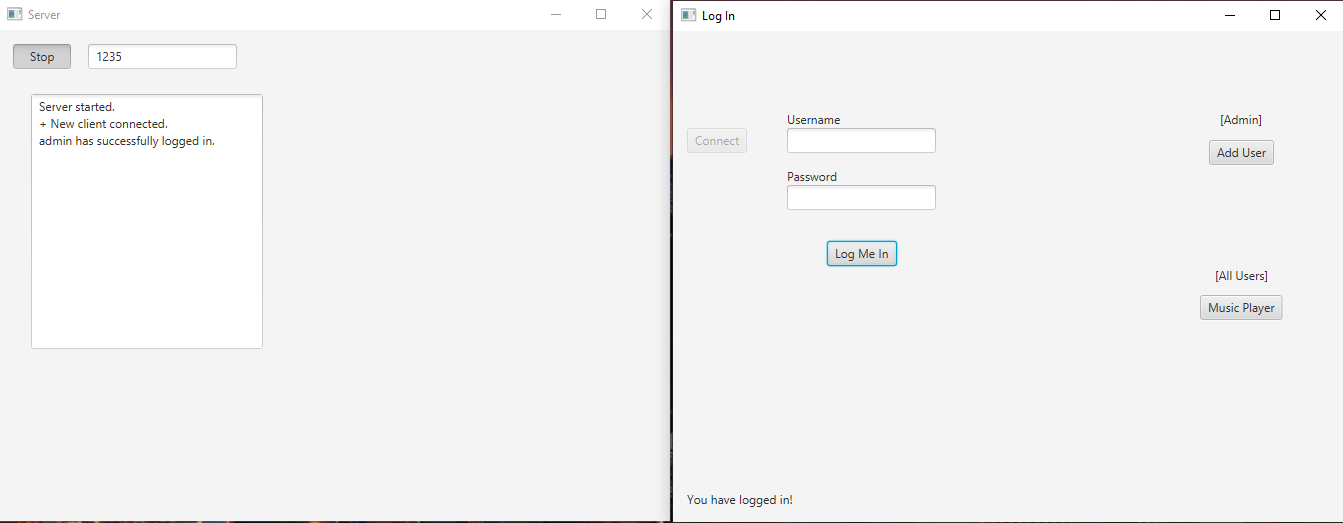
Additional Features:

* A Queue data structure is used to queue songs.
* A Stack is used to access song history via the previous button
* Use a similar concept to memento design pattern for serializing the application state.
* User data is saved on client-side, including CSV, object files (for the music player library state).
* Server will save account data on shutdown. A List<Account> object is serialised. Passwords are saved as hash/salt.

## Demonstration / Marking Criteria

Server must be running and started to allow client to log in. Default admin login is:

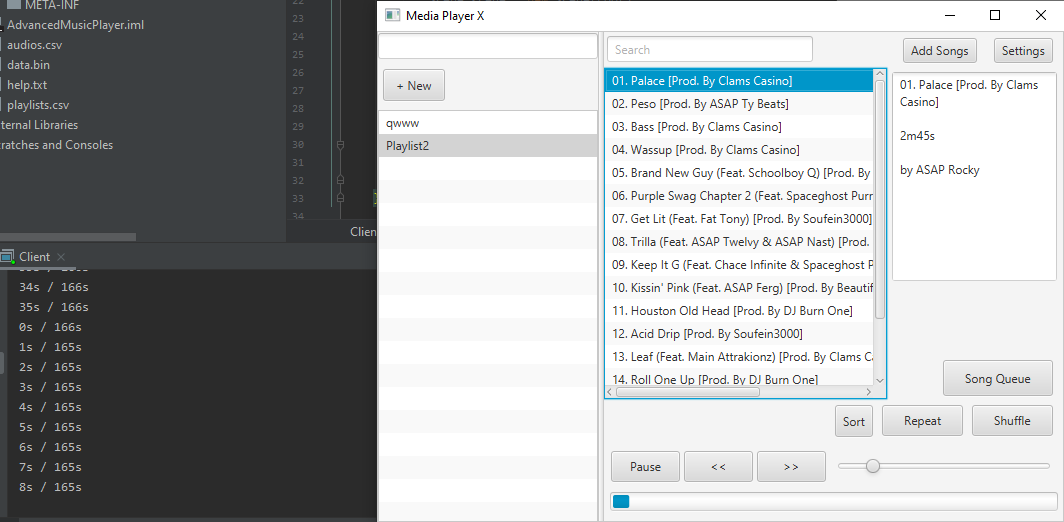
User: **admin**, Pass: **admin**



Add User button does nothing, but is only seen by Admin accounts. They would be able to add users to the server user base.

### The Player

A playlist is added by typing a name at the top-left input, and clicking “+ New” button

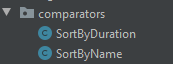
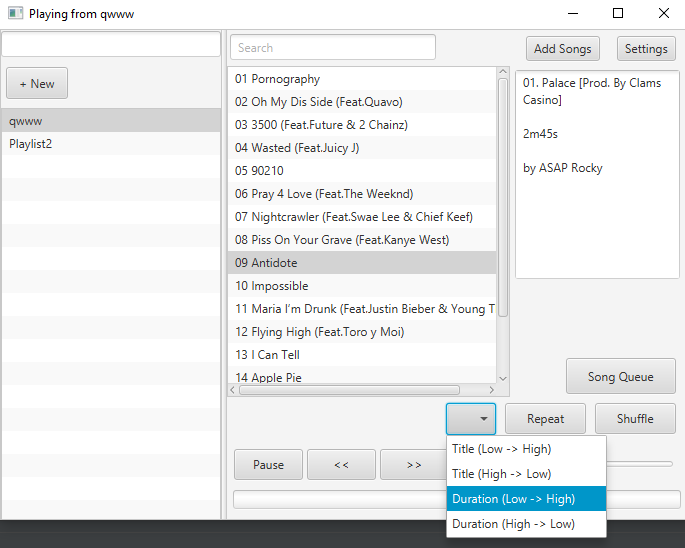


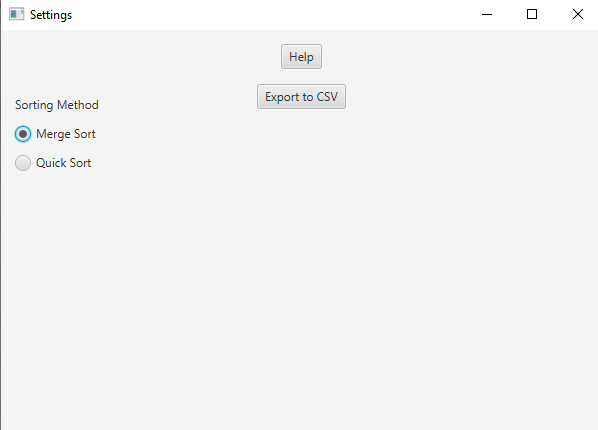
The Binary Search method requires the exact song name (as displayed in the playlist) to be typed. Binary search relies on the comparison being accurate to work.

A better searching method might check for the closest string, using contains().



Sorting can be performed using merge sort or quick sort. Switch between the two options in Settings. Sorting the list by Title or Duration is possible using the comparators.





### Marking Criteria

**Dynamic Data Structures**: Queue, Stack, Binary Tree (playlist TreeSet<Audio>)

**Hashing Techniques**: When the client application sends a login request, the server application will perform a hashing test to match the credentials entered with the existing hash/salt stored in the account list.

**Sorting Algorithm**: Merge and Quick sort

**Searching Algorithm**: Binary search

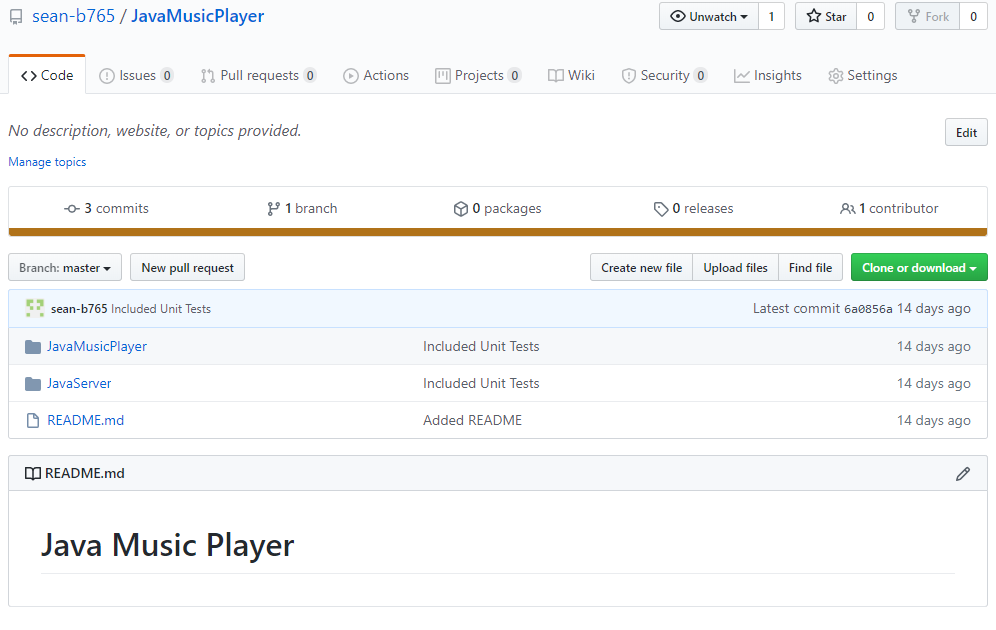
**3rd Party Library**: OpenCSV

**Help Files**: if help.txt is not found, the application will create the help file and fill it with help on using the player.

**Debugging**: has been documented in [Debugging section](#_Debugging)

**Unit Tests**: [Unit Testing section](#_Unit_Testing)

[**Source Control**](https://github.com/sean-b765/JavaMusicPlayer)

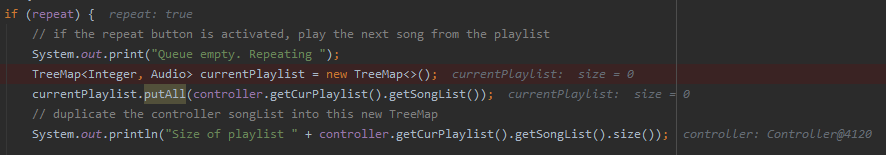


## Debugging

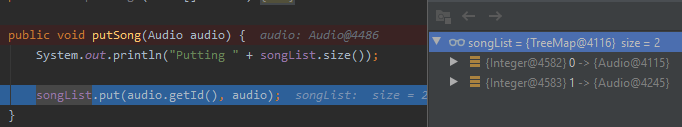
Required VM Options:

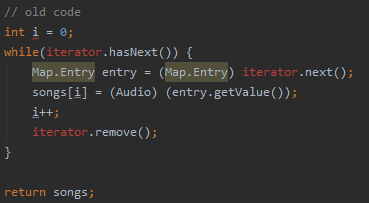
--module-path ${PATH\_TO\_FX} --add-modules=javafx.controls,javafx.fxml,javafx.media

When selecting the repeat toggle button, I encountered an issue where the songList TreeMap wasn’t correctly putting and saving songs.



The curPlaylist variable in the controller was expectedly returning the selected playlist, but the songList field was not working for some reason.



The issue was not when storing songs (seen above) so I did a quick code review to see if I could find the error by eye. This was not productive so I used CTRL+F to find mentions to songList, or getSongList() in my classes.

The error was in my getSongs() method in Playlist.java, where I used an iterator to return Audio object array. After iterating past a song, I removed the iterator, and this would remove the Audio object itself as well.

To fix this, I iterated through the Map Entries using a for loop.

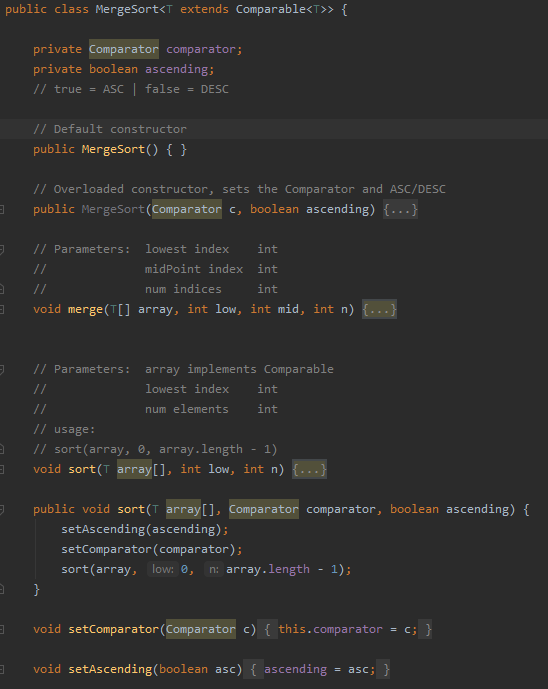
### Sorting

When attempting to sort a TreeMap, it is a bit difficult, as a TreeMap will already be sorted by the [natural ordering](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html) of its keys, or by a [Comparator](https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html) provided at map creation time *(Java API)*.

I realised at this stage that the TreeMap is more of a hashmap than a binary tree. So I changed to a TreeSet.

If I hadn’t used a Playlist class containing a *Tree* songList field, this would have been a lot more tedious to change over, but my putSong(), remove() and contains() methods were all localised into the one class. This is similar to the façade design pattern, where an interface would be used instead to encapsulate methods.

#### Merge Sort

I used generics to allow the merge sort algorithm to work with any comparable object. This was not necessary, but was a good thing to learn.

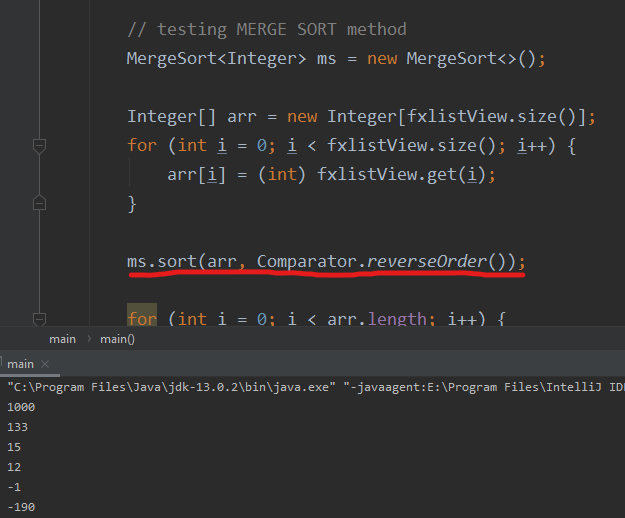
I also made the Comparator to be changed for the MergeSort, and a boolean which corresponds to ascending/descending.

The only public method sort() takes the Comparable array[], the comparator to be used, and the ascending boolean.

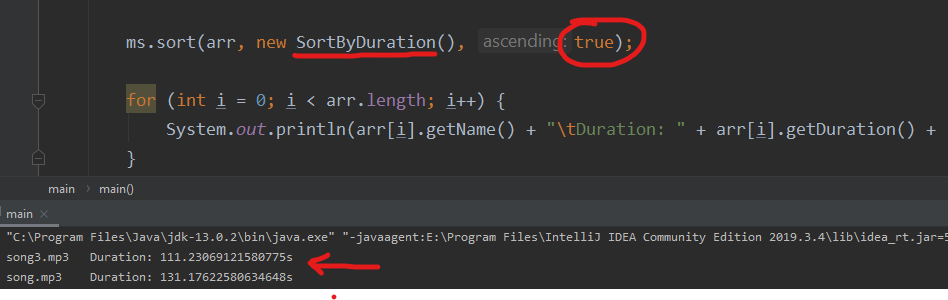
#### Quick Sort

This was easy enough to implement after already having done the Merge Sort algorithm. However, I still performed tests.



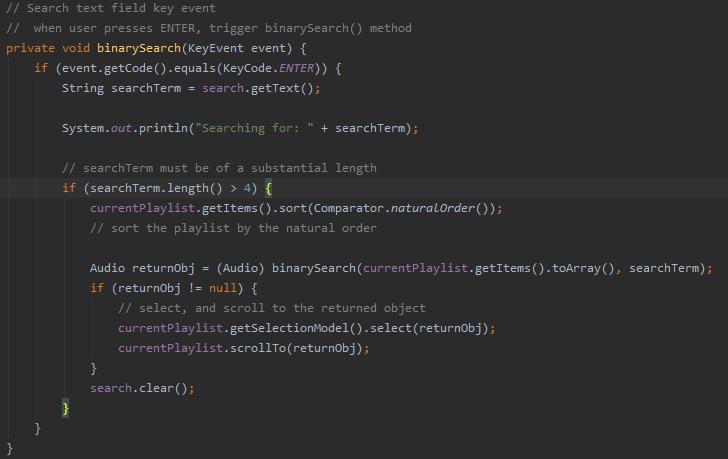
The TreeSet<> in Java is still **naturally sorted**. To allow a sorting function to actually sort the playlist, I performed my sort methods on the **javafx ListView items**. This would allow the user to view a sorted arrangement of the different songs, but the underlying binary tree would not be altered.

The custom Audio Comparators worked fine.



### Searching

My search method is triggered when pressing the Enter key on the Search text field. It will call a binarySearch() method which takes the text field String as an argument.

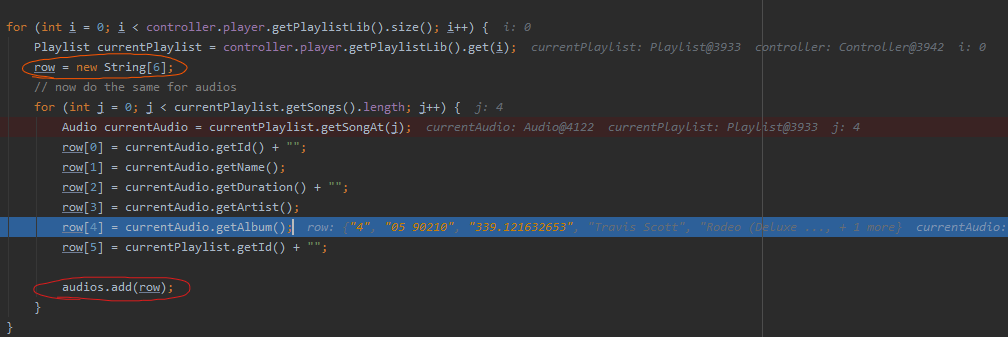


### OpenCSV – writing to CSV files

When exporting Audio objects as CSV records, I ran into an issue where the CSV files would consist of duplicate rows.

To fix this, I debugged the exportCSV method in the place where I was writing to the audio List<String[]> collection. The error was in the String[] row object, which when added to a List, will pass the object itself, rather than the values.

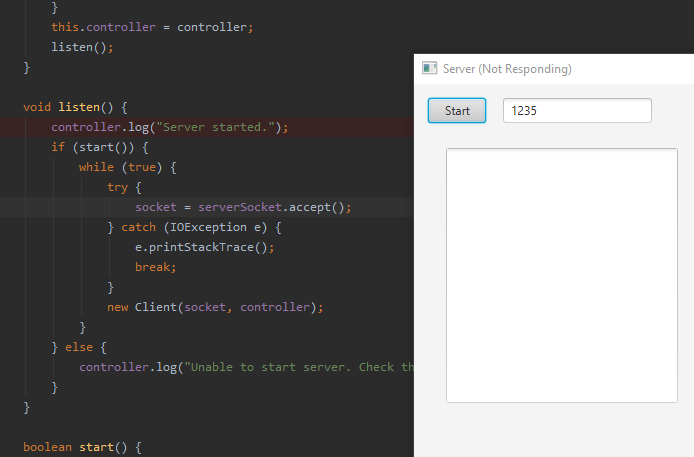
To fix this, I needed to reinstantiate the row String[] each iteration, so a new record would be saved each time.



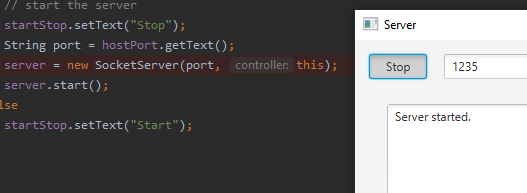
### Sockets

When testing the server program, it would freeze during the listen() method which should listen for new client connections.

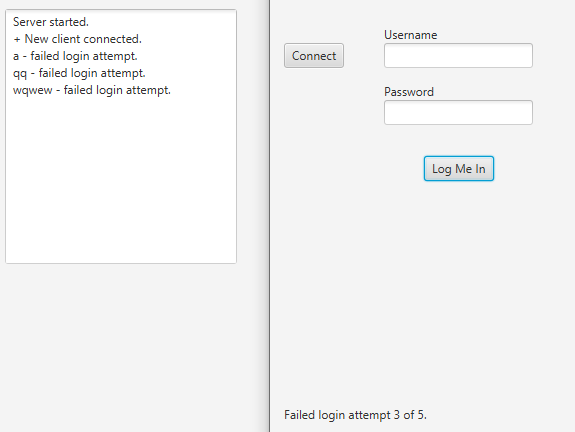
This is probably because, currently the server networking class is running on the same thread as the JavaFX UI. To fix this, I made the server object a thread, which would listen synchronously.



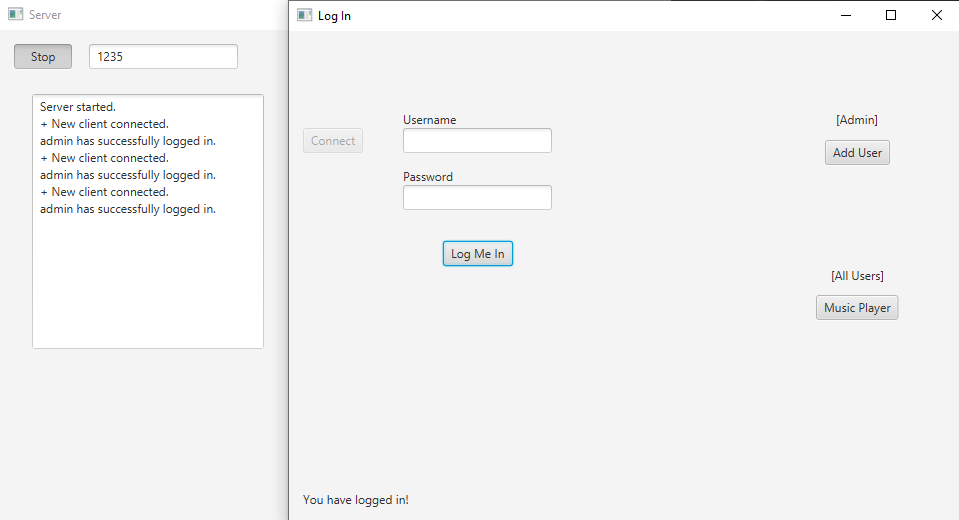
The thread running concurrently with the JavaFX UI:



When validating login attempts, the server will check first if the username exists in the registered accounts list. If it does not, there is no need for hash validation. The client side will receive a “incorrect login” message from the server, which will update the client UI.



If the login was successful, then the client side will display the buttons to access the music player. It will also display the admin options, if the Client is an admin.



You can see above that multiple clients can connect. There is currently no account limit i.e. multiple admins can be logged in. And there is no disconnect message when the Client exits the application.

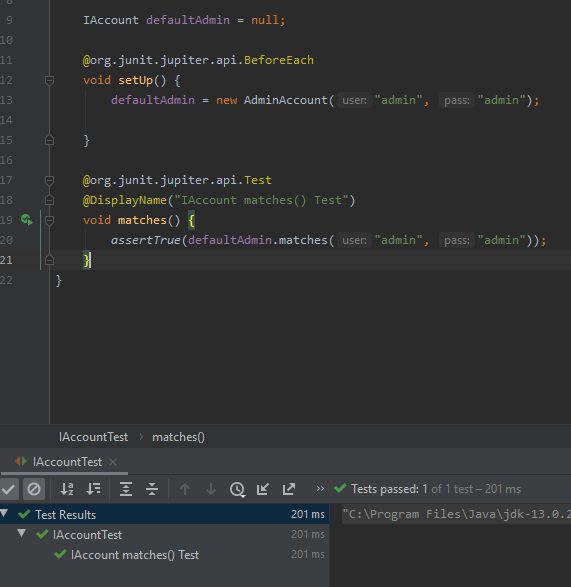
If the client

## Unit Testing

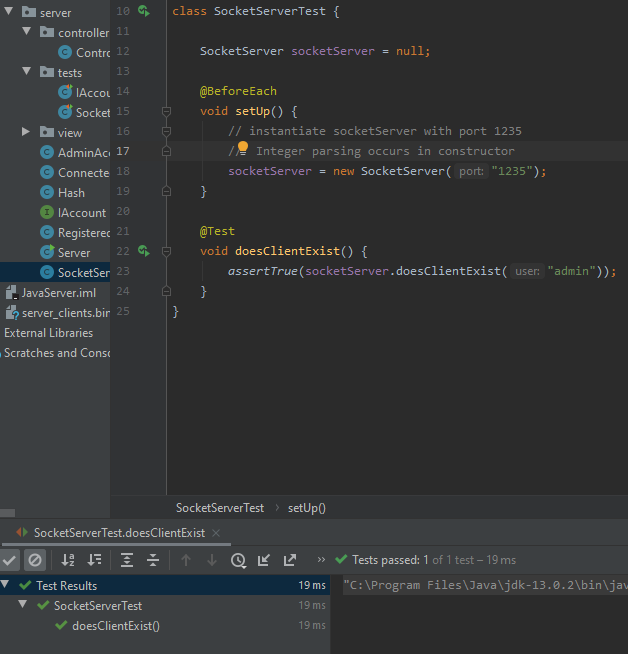
### Server Module

IAccount matches() function. Will call the static Hash.matches(char[], byte[], byte[])

It is vital to test any authentication functions as they are the main security to my application.

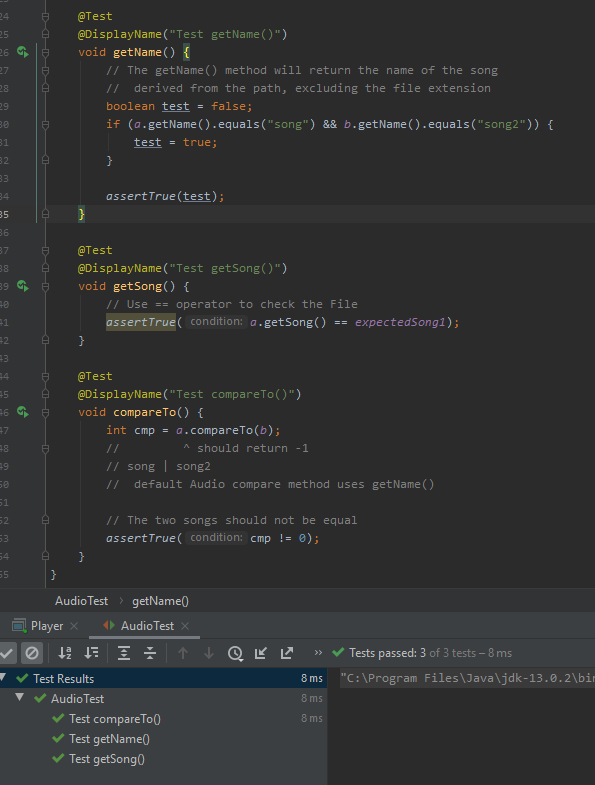


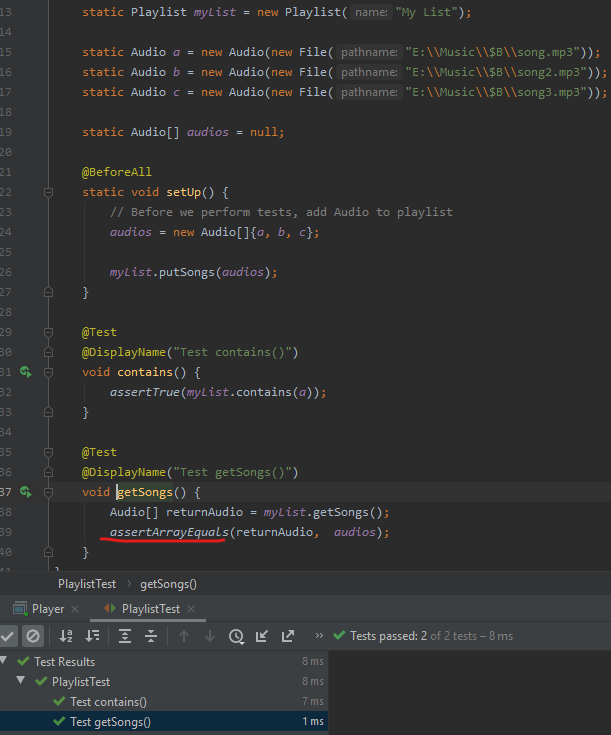
When the socket server object is constructed, the deserialization method is called. This method will also check for the default admin record, and will create one if there isn’t one already.



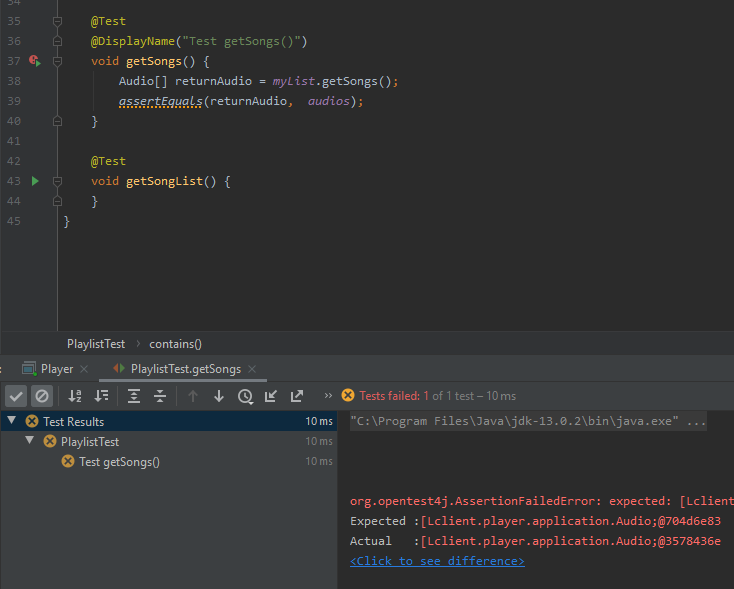
### Client Module

Audio object tests



Playlist tests

When testing the getSongs() method, which returns Audio[], the expected result was different from the actual result, when using assertEquals(). To check the reference of each contained array item, we can use assertArrayEquals().



*assertEquals() above, assertArrayEquals() right.*