A **database server** with a **desktop client** in Java

Peter Guld Leth
Steffan Christensen
Mihai-Alexandru Ciungu
A220

Overview

This PFI mini-project is a database intended for containing quotes. It has two parts, the Client part and a fully functional Server that can store and remember the quotes. In order for the program to work, an instance of the server must run. Shown in Figure 1, the program has some basic functionality expected from a database: the user can add, remove and edit a quote and also refresh the list if necessary.

A note on running the code: The project uses *Gradle*[1] as the project-/package manager, meaning the code can be compiled and run from within the "code" folder using `gradle run` on the command line. An instance of the Server package has to be run from within its directory using Gradle for the Client package to work as intended. Alternatively, handed in alongside the code are *windows batch scripts* for each program, which can be run by double-clicking them, located in the "distributions/*package*/bin" folders.

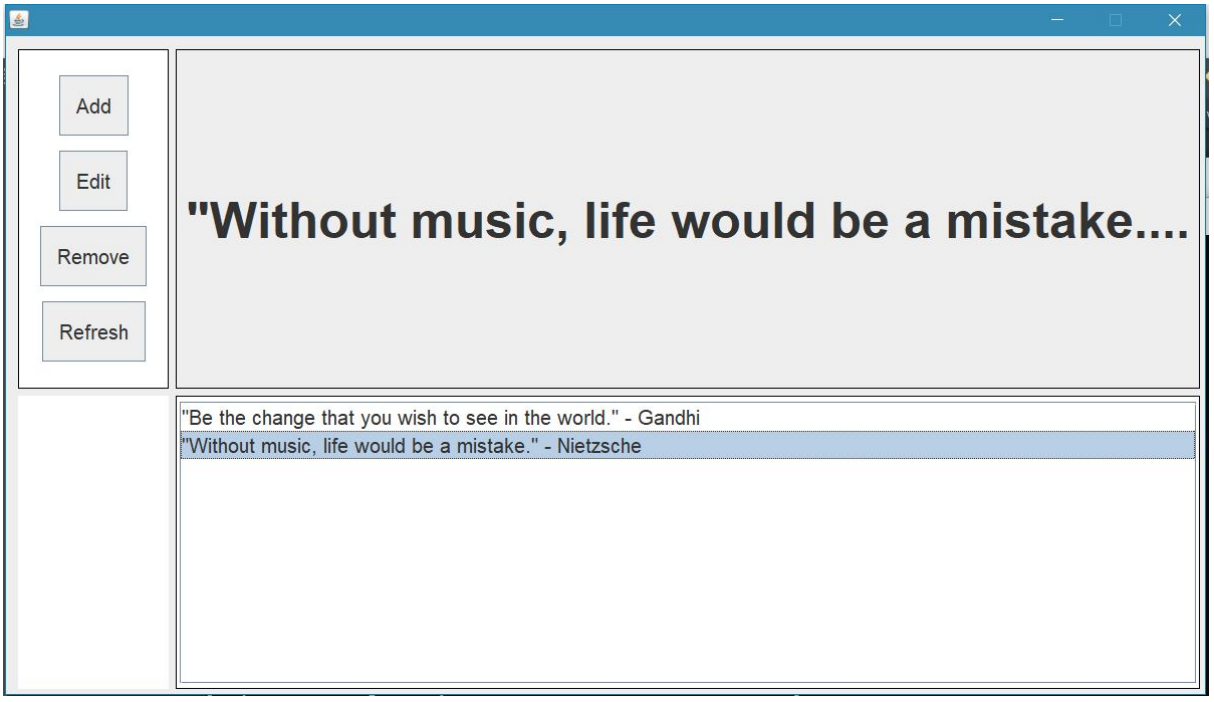


Figure 1. The database interface (client package), containing the buttons, the text display/text field panel and the quote overview panel.

[1] Gradle is available for free here: https://gradle.org/

Client package

As shown in Appendix A.a, the client package consists of 7 classes with the majority of them being dedicated to the user interface:

1. The ***App class***, Serves as the entry point class
2. The ***Gui class***, The main class for the user interface serving as the frame which other container classes are added.
3. The ***ButtonPanel class***, one of the container classes contains 4 buttons used for modifying the database
4. The ***Dbdisplay class***, one of the container classes used to display selected entries from the database
5. The ***Dbinput class***, a container class with a text field giving for the user to add and edit entries
6. The ***Dbtable class***, one of the container classes has a list that shows the different entries in the database.
7. The ***Server class***, Sends and retrieves data from the remote server

When the program is started, the App class will initialize a new instance of the class Gui, which then constructs the graphics environment. Starting by creating an instance of the necessary container classes, as well as initializing 4 instances of the JPanel class, which are used to set up the layout for the window using a GroupLayout. The Gui class also sets up the size for the window and creates listeners for several different components inside the different containers.

The listeners are used to run code that changes the state of the program when a certain event is triggered. For example when the selection of the list component is changed or when a button is pressed. From this point onwards the program will wait for the user to perform an action and trigger an event. Selecting an item from the list will display an enlarged version of the entry, this is done using the list selection listener that runs code retrieving the selected entry from the listmodel.

The user can also use the buttons on the interface to edit the database. This is done using action listeners that call different methods from the Dbtable class that updates the database. Two of these buttons do not directly call the method, but instead swap the top JPanel to the Dbinput panel, this allows the user to type in text whenever the function requires it the textfield. Pressing enter while the text field is focused then triggers the event for the text field's actionlistener, which then retrieves the text from the text field and determines which method it should be applied to using a switch case. This then updates the database and the list and brings the program back to a state where it is waiting for the user to perform an action.

Description of the Dbtable panel:

Dbtable: The list in the Dbtable class is made up of several components: one of these being the JList component. This is a very flexible and customizable component that can be set up in several ways within the Dbtable class. We use the list selection to be single selection which means that only one item in the list can be selected at any time. We also choose to disable horizontal wrapping and set the list to have all entries visible at once by setting the visible row count to be -1.
In order to ensure that all entries in the list are accessible to the user even within our limited space we add the list to a scrollpane which is a type of container which then can be added to the container and set up to fit within the size of the panel.

With the JList component there are several ways of adding items into the list with the most simple one being adding the contents of an array to the list. In Dbtable we decide to add data using a listmodel which is another component that we add to the JList. This gives us much of the functionality from Java's List which means that we easily can add new entries and edit existing ones. This is used together with our server interface to update the list based on the contents of the database. Within the class there are also several methods used to update the database and one for updating the list. The way we update the list is by first emptying the list using the listmodel clear command and then adding each element from the database into the list using a for-each loop

Server package

Class hierarchy
As shown in Appendix A.b, the server package consists of three distinct classes:

1. The ***App class***, the entry point of Java App execution.
2. The ***Database class***, acting as an interface to the *h2 SQL driver*[2].
3. The ***ConnectionThread class***, a subclass of *Thread*, processing a client request in a separate thread. The reason for doing this is explained below.

When the program is started the *App Class*, from which the code begins execution, initializes an instance of the *Database class*. Execution is then halted until a client connection attempt is received, upon which the *App Class* initializes and runs an instance of the *ConnectionThread class*, passing along the socket connection and the *Database Class* instance as well. This means the *Database Class* only has one instance during execution, which is then shared amongst all concurrent *ConnectionThread* instances.

The protocol
The protocol is a very simple layer built atop the TCP protocol, with network messages being interpreted line-by-line. The first line is the intent of the call. This can be either `get`, `add`, `edit` or `delete`, corresponding to which actions to be performed on the database. These are then followed with request parameters, one parameter per line. The server then responds with a success-code, either one or zero, followed by the updated table, preceded by a `data` line to signify this.

We chose this rather than the more high-level HTTP protocol, because we anticipated to have to maintain persistent two-way connections. This was not feasible, but we still decided to stick to the protocol, even though the final request-response flow ended up a lot like standard HTTP.

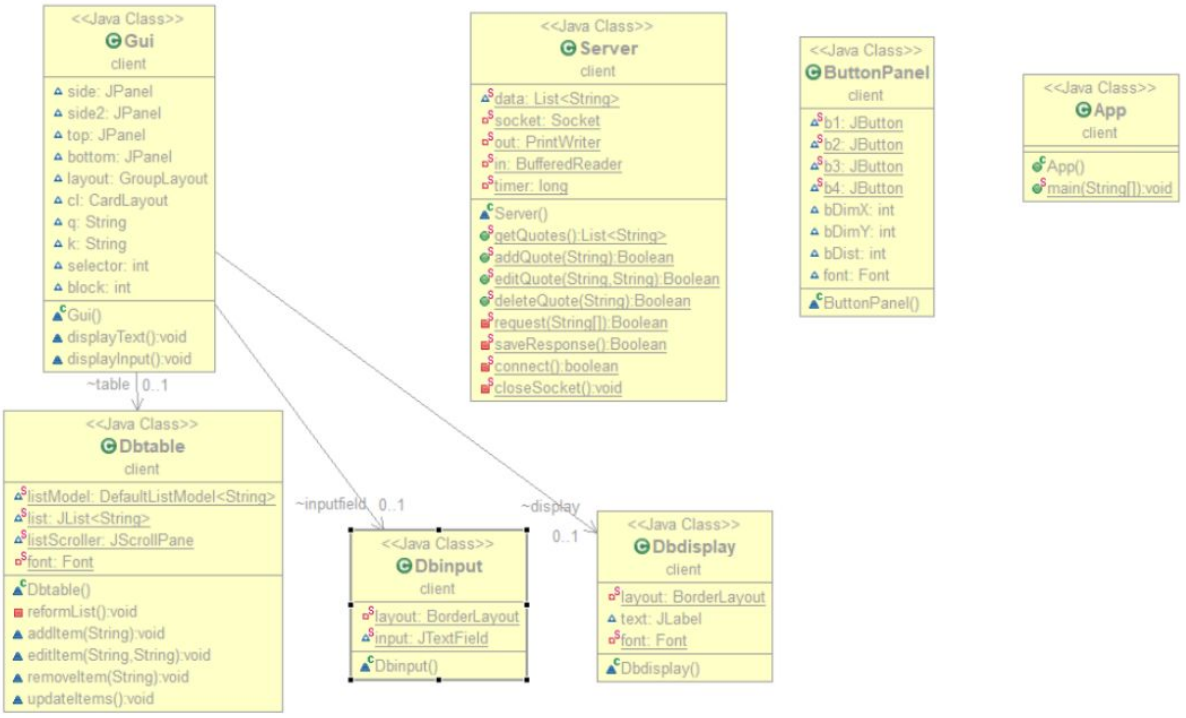
Why spawn new threads?
Spawning and managing threads can be difficult. Additionally, these processes all create connections that must be closed properly before discarding (a.i. TCP Socket connections), further complicating cleanup procedures - so why bother?

Originally, the ambition with the Server package was to establish and maintain persistent connections to the client, distributing updated data organically as changed were carried out, from all connected clients. We were however unable to achieve this, but spawning and maintaining *ConnectionThreads* were still a viable strategy, since this allowed the main thread to spend as much time as possible in the halting `ServerSocket.accept()` call, minimizing the chance of an unsuccessful connection from a client, because the main thread was occupied with responding to a previous query.

[2] the *h2 SQL driver* is an external dependency for working with SQL databases in Java and will not be explained further in this report for the sake of brevity. View the *Database Class* code for examples of use, just note that the h2 driver runs the database *in-memory*.

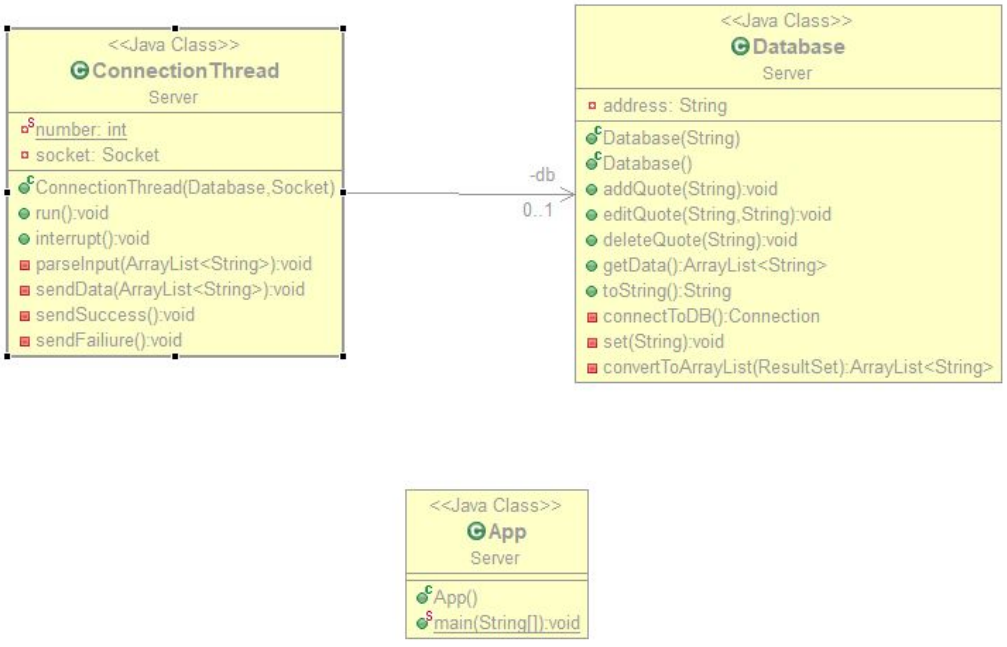
Appendix A: UML diagrams

a.



UML Class diagram for the Client package.

b.



UML Class diagram for the Server package.