Coursework Report

Introduction

This coursework required that I design and produce a database that can be used to store gym bookings for a fictional gym as well as the queries to filter and sort such a database. I also needed to produce the code for a program to run said queries as well as adding and updating entries in the database. It also required that I make a second program with a graphical user interface that simplifies the creation of queries for use by the fictional staff.

What I have produced I believe fulfils these requirements. I have made the database, the query making code and the GUI code

My work consists of multiple components, each a single layer in a multi-layer project. Moving through the layers going from most hidden from the user to least hidden from the user, the first is a database implemented in SQL. This is interacted with via SQL queries generated by a server program. This program generates the queries based on inputs send by a client program which in turn receives input from a GUI attached to the client program. Via this process, the user of the program needs no knowledge of how any of the programs work nor how the database works or is laid out. Despite this, they will be able to easily perform all actions requested by the project specification, such as creating, editing, deleting and viewing bookings.

The layout of this report shall be as follows

* Database design

Here I will explain how I came up with the layout of the database, why I chose to make it that way and how that plan turned into code.

* Software design

Here I will explain how I designed the software required and my thought process for doing so

* How I tested my project
* Conclusion
* References

Database Design

For this project I decided to use a database comprised of 3 relations. One relation would hold all data on the gym’s clients, these being ‘clientID’, ‘firstname’ and ‘lastname’. The second relation would hold all data on the gym’s trainers, these being ‘trainerID’, ‘firstname’ and ‘lastname’. The last relation would hold all bookings made by the gym under the headings ‘bookingID’, ‘clientID’, ‘trainerID’, ‘date’, ‘timeStart’, ‘timeEnd’ and ‘focus’.

(‘clientID’ and ‘trainerID’ are foreign keys here from their respective relations).

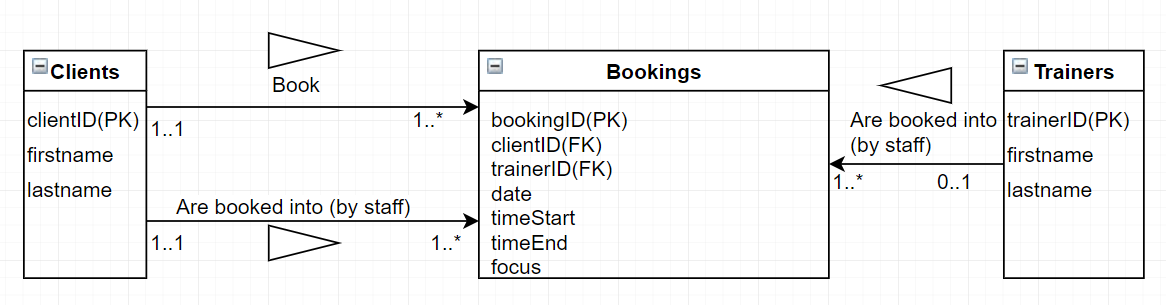
This database needed to be in third normal form (3NF)\*. While my original design looked as though it was already in third normal form due to each table having an attribute for ID and a simple design, I still double checked.

Firstly, it must be in first normal form because I cannot make an SQL database with values in 2 or more boxes at once – each box must contain one and only one value.

Secondly, I determined that there were no partial dependencies as there is only one attribute being used as the primary key on all relations and therefore it was in second normal form.

Thirdly, I determined that since every non-primary-key attribute was only dependant on the primary key and not on any other attribute, the database must be in third normal form and normalisation was complete

That meant it was easy to add information to the relation without causing update anomalies which means it is possible to safely add and update entries on the database. The requirement that the gym staff can edit the database is fulfilled.  
  
\*The textbook *Database Systems: A Practical Approach to Design, Implementation, and Management, Global Edition* defines third normal form as ‘*A relation that is in first and second normal form and in which no non-candidate-key attribute is transitively dependent on any candidate key*’.1



A digital copy of the hand-drawn UML diagram I made for this database.

As far as I can tell, there is no need to write or update data stored in the Client and Trainer relations, only reading off IDs to confirm bookings are valid.

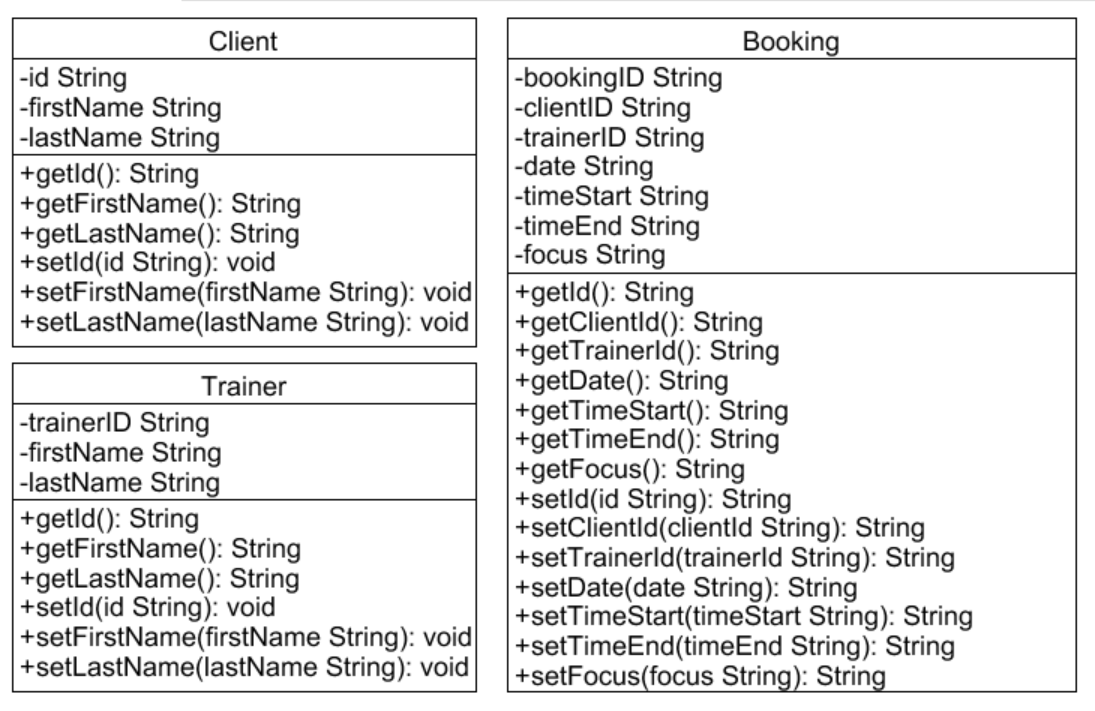
Software Design

My plan for the software was multi-step. First, I created the SQL that would generate the database as detailed above. Then I created Java classes to store info on Bookings, Clients and Trainers in transfer between the GUI and database. Next came Java code that took instances of the classes as input, then generated SQL queries that checked if the input was valid with regards to pre-existing data on the database and other queries that performed the requested action of add/update/delete/list.

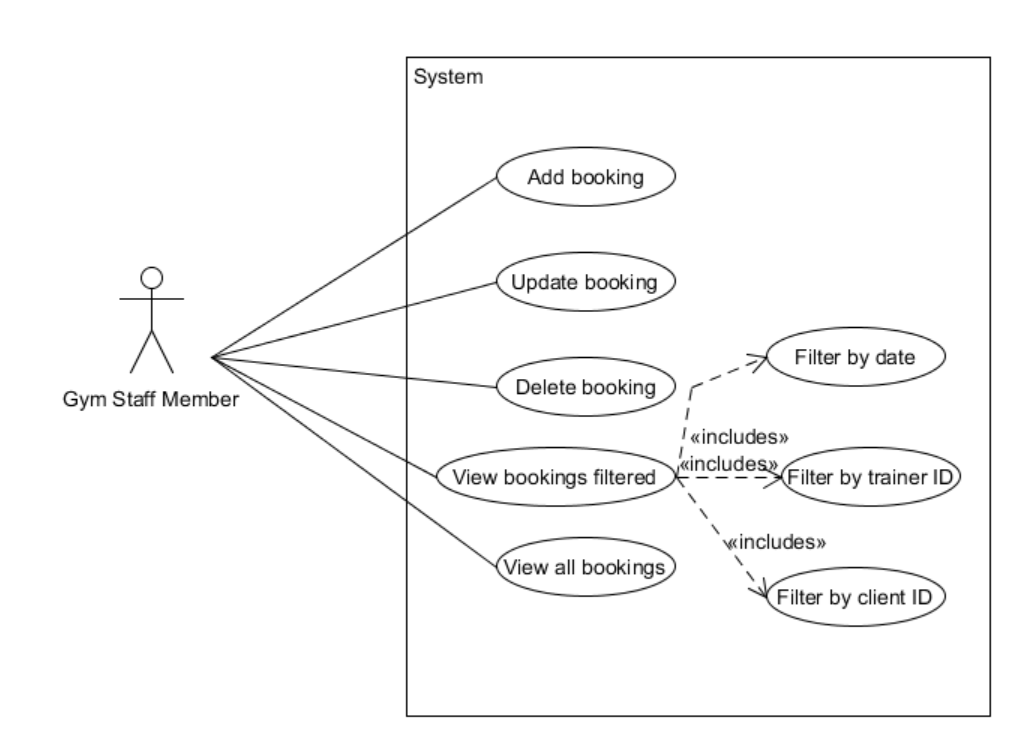
After all this was written, I added code for a GUI using JavaFX that enabled input with TextFields and had Buttons programmed to extract data from the TextFields, create class instances using said data and run the query-making functions on those class instances, returning the SQL output as output.

The software as a whole is capable of the following functions as specified in the specification:

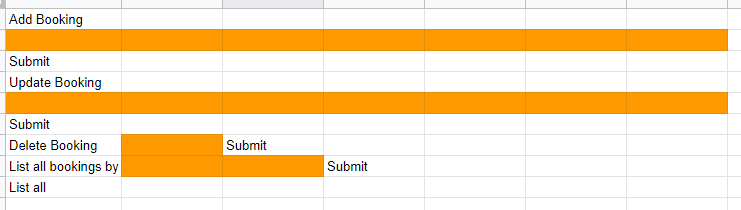
* Add a new booking
* Update an existing booking
* Delete an existing booking
* List all bookings for a specific trainer/client/date
* List all bookings

Java Classes UML Diagram

Unfortunately, since none of the classes inherit from any other class here or have an attribute of the same type as a class here, there are no lines between them. I am fully aware of the existence of the arrows on class diagrams but cannot see a way to use them here.



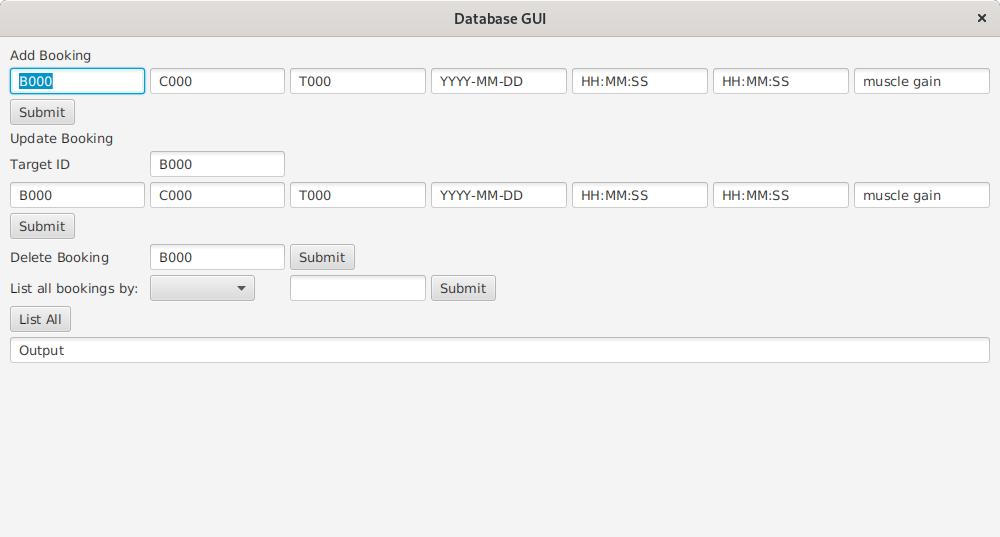
Use Case Diagram for intended use



GUI Layout Diagram

Here the word ‘Submit’ or ‘List all’ designates a button, an orange box designates a TextField, and all other text is a Label.

In practice, the TextField next to ‘List all bookings by’ has been coded as a ChoiceBox to restrict user entry to only approved categories but this change was not originally planned.



Screenshot of GUI made in JavaFX

The software will be divided into 2 physical programs, one for the Client and one for the Server. The Client program will contain the GUI and code to extract data from the GUI and produce class instances from that data. The Server program will contain the code to convert class instances to SQL statements and run it on the database.

Data is transferred between the two programs with ObjectInputStreams and ObjectOutputStreams.

I can also send an instruction to the server on what action to perform on the booking data with a String as a String is also an object in Java.

Testing

I regret to say that I have not managed to implement or perform any unit tests on the code with JUnit. As of my introduction to JUnit, most of my code was already written and that which wasn’t written was not easily broken into chunks to test as it was extensions to previous functions to complete functionality.

I can however say that I have tested all the code in the project as best I could manage prior to learning about JUnit and can confirm that the GUI, database, and JDBC code all worked in isolation upon creation.

Conclusion

I believe I have managed to implement most of the functionality requested in the project brief.

The main limitations of my current design is that I was unable to prove that the Socket and ObjectStream code actually works nor test the work as a whole as my Virtual Machine crashed on Thursday evening and I could not fix it in time to test the code.

In all honesty, I feel my work is incomplete but lack the time to finish it.

If I was to approach a similar project again in the future I would take the time to implement JUnit tests or the like earlier on and I would seek help if possible for the parts I found challenging (Like the socket code) earlier on in development

References

1 – Connolly, Thomas, and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management, Global Edition*, Pearson Education Limited, 2015. ProQuest Ebook Central, <http://ebookcentral.proquest.com/lib/mdx/detail.action?docID=5174902>.

Obtained on 2020-01-09 07:37:12.