# JDBC Assessment

Paul McHard - 2085227M

November 30, 2017

### State of Program

The program has been completed as required by the specification and achieves the desired functionality. The GUI is clearly layout and easy to read; the program is capable of displaying the desired information regarding courses, their instructors and capacities in one report, and information on members and the courses to which they are enrolled on another report. The program also allows members to be booked onto a course and updates the database accordingly, as well as preventing certain bookings in the case where the user is already booked onto the course or if the course is already fully booked.

### SQL Queries used

The program was designed to place more of the key data processing work on the shoulders of Java rather than SQL. This is reflected in the code as more complex systems; use of objects, if statements and nested looping to check inputs and data validity, are used in the program to deliver the necessary functionality. As the Java code handles the majority of this data processing, the SQL code is left with only interactions with the database; simple extraction and insertion of data into the relevant tables, as opposed to handling more specialised requests of the database and selection using more complex SQL statements, potentially involving nesting, joins or specifying criterion.

### Code Examples

```
--Extraction of information from database follows a repeating pattern,
--SELECT COUNT(*) is used firstly to get number of rows in a table,
--This is used to establish the size of relevant arrays in java.
--Then the entirety of the data from the given table is extracted using SELECT *,
--allowing the program to create full objects for each tuple in a given table.
--Process is repeated for each table we wish to use in the program.
--These are called in the program's update methods
--updating the objects and arrays with each new database interaction
--to avoid discrepancy between data stored in the program and in the database.
SELECT COUNT(*) FROM gym.member;
SELECT * FROM gym.member;

SELECT COUNT(*) FROM gym.course;
SELECT * FROM gym.course;

SELECT COUNT(*) FROM gym.member;
SELECT * FROM gym.member;

SELECT COUNT(*) FROM gym.coursebooking;
SELECT * FROM gym.coursebooking;

--This is the SQL code used for adding a new course booking.
```

```
--SELECT COUNT(*) is used to determine the unique booking number, by design
--the booking number is an integer, which indexes bookings incrementally from 1.
--INSERT INTO is the only method by which a booking is made. Adding the booking
--directly to the database, then updating the arrays and objects by re-accessing
--the information from the database helps avoid duplication of added data as well
--as bugs pertaining to discrepancy between the program data and database data.
SELECT COUNT(*) FROM gym.coursebooking;
INSERT INTO gym.coursebooking
    VALUES("+bookCount+","+bookMember.getID()+","+bookCourse.getId()+");
```

**Testing Performed**

| Test | Appendix Figure |
|---|---|
| Program produces list of all courses, their instructors, capacities and current enrollment. | 1 |
| Program produces list all members currently booked on a course, their names, ID's and the course their booked on. | 2 |
| Program allows a user to be booked onto a course | 3,4 |
| Program prevents a booking for a course which is at capacity | 5 |
| Program prevents a booking if user is already booked on course | 6 |

**Conclusion**

There was concern during development that this Java-centric approach, while successful, was inappropriate and missed the core learning objectives of the task. However, on discussion of this with the course head, it was determined that the specification facilitated many possible approaches, so this is perfectly valid. There is an argument for this method from a technical perspective; by performing all data processing and input checking in the Java code, the system is safer and more secure at dealing with invalid, potentially damaging database commands, as more rigorous design can be used to catch such instructions and prevent corruption of the database.

One key thing I have learned is that there is a large degree of freedom afforded by a project which involves the interaction of an object oriented language like Java and a database utilising a SQL, in terms of what prevalence is put on either language for development. These design decisions are important and, for a professional program, careful attention should be given to determine the best mix of Java and SQL to achieve the desired functionality and maintain a secure program and database.

## Appendix - Testing Screenshots

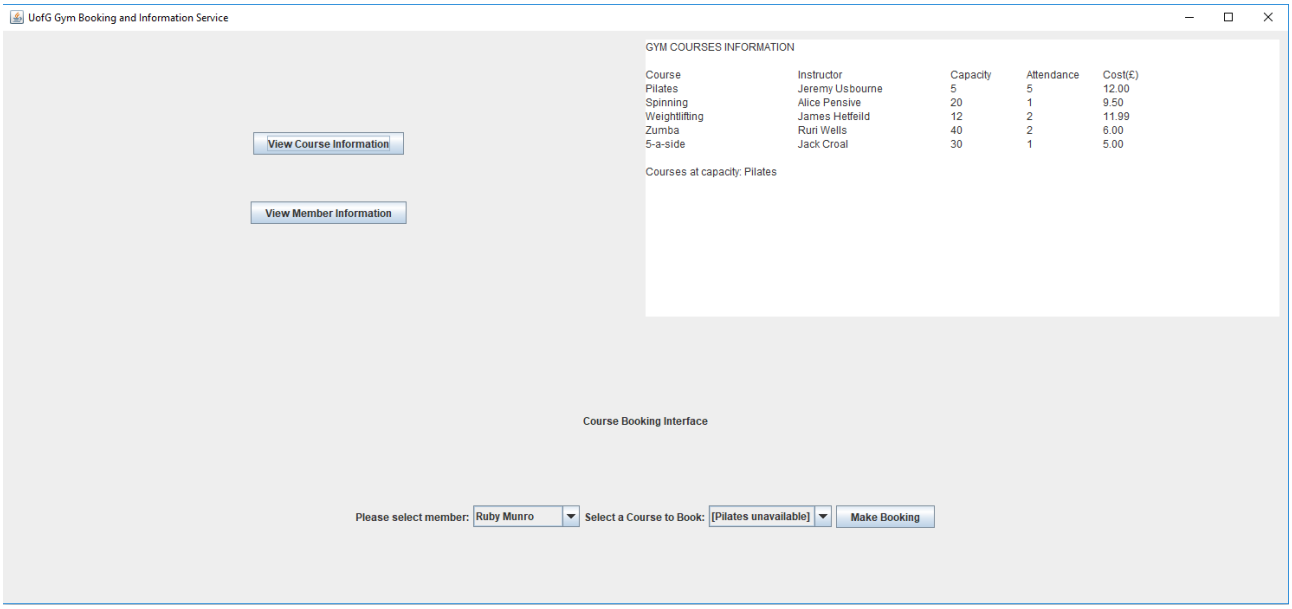


*Figure 1*

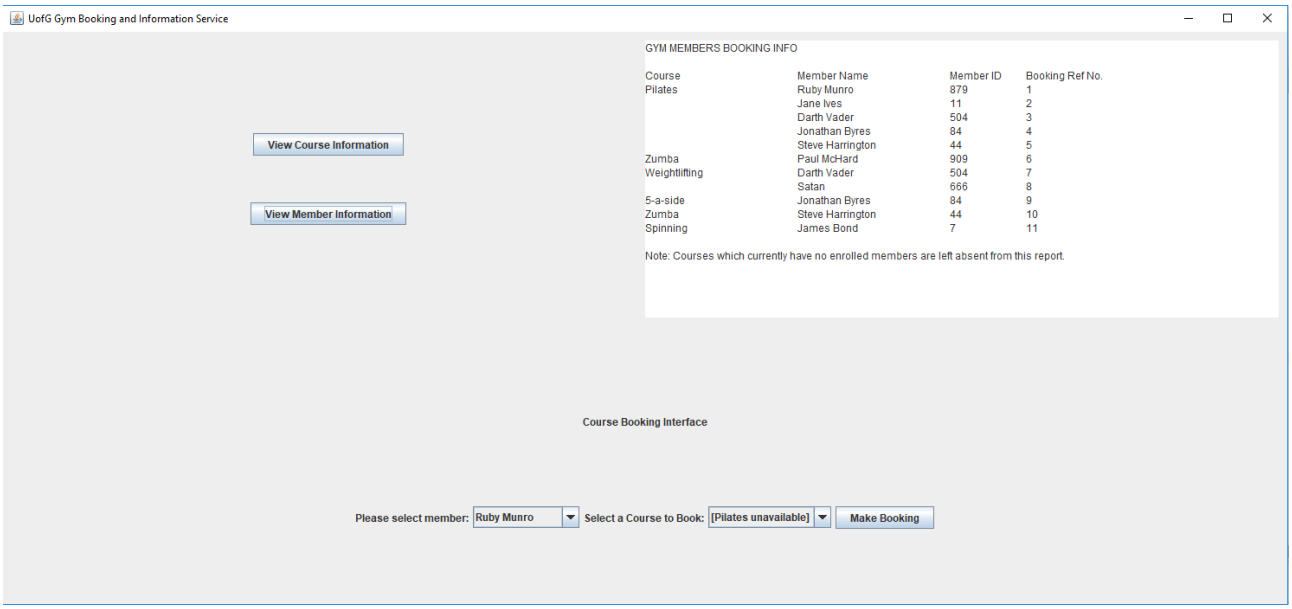


*Figure 2*

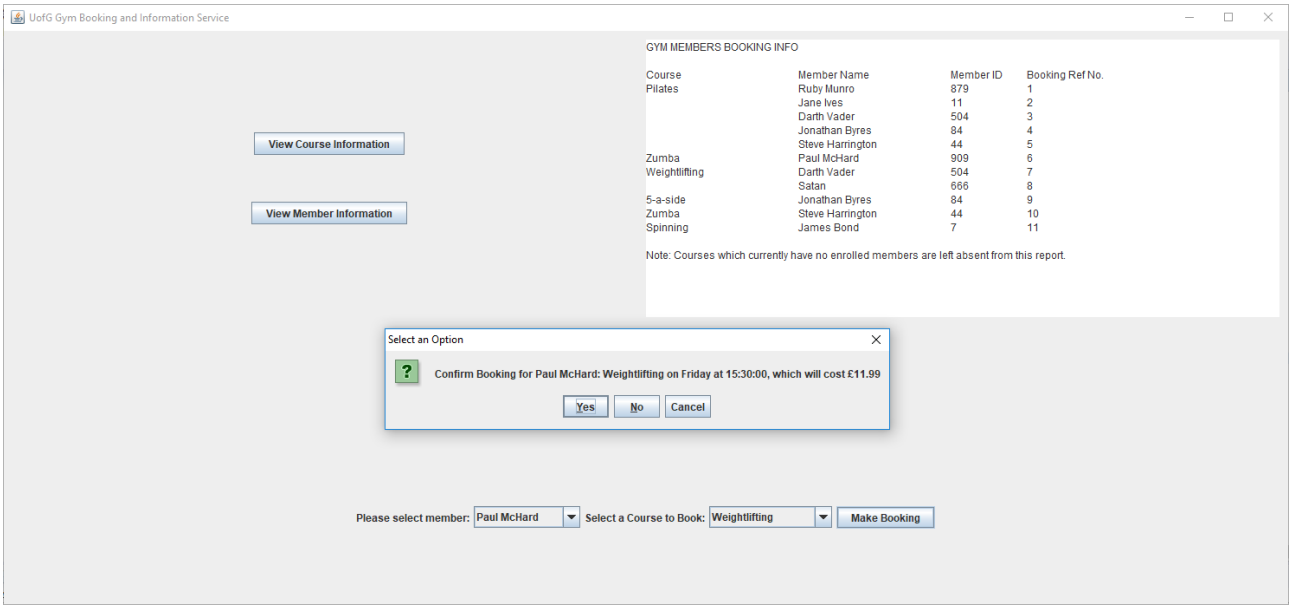


*Figure 3*

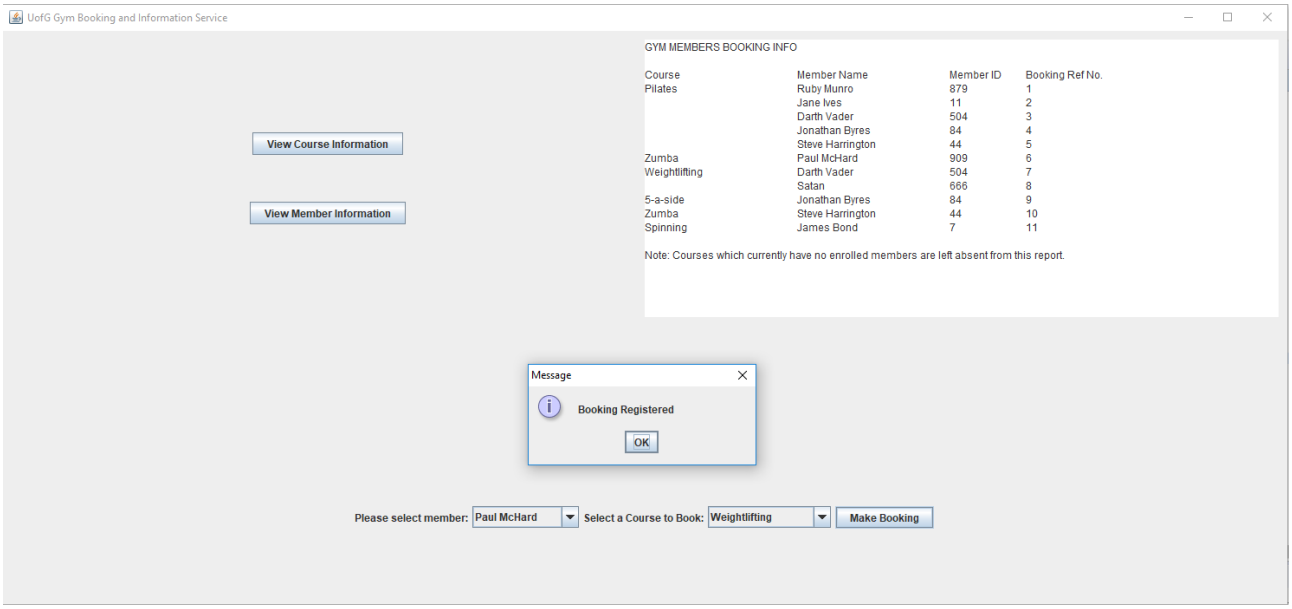


*Figure 4*

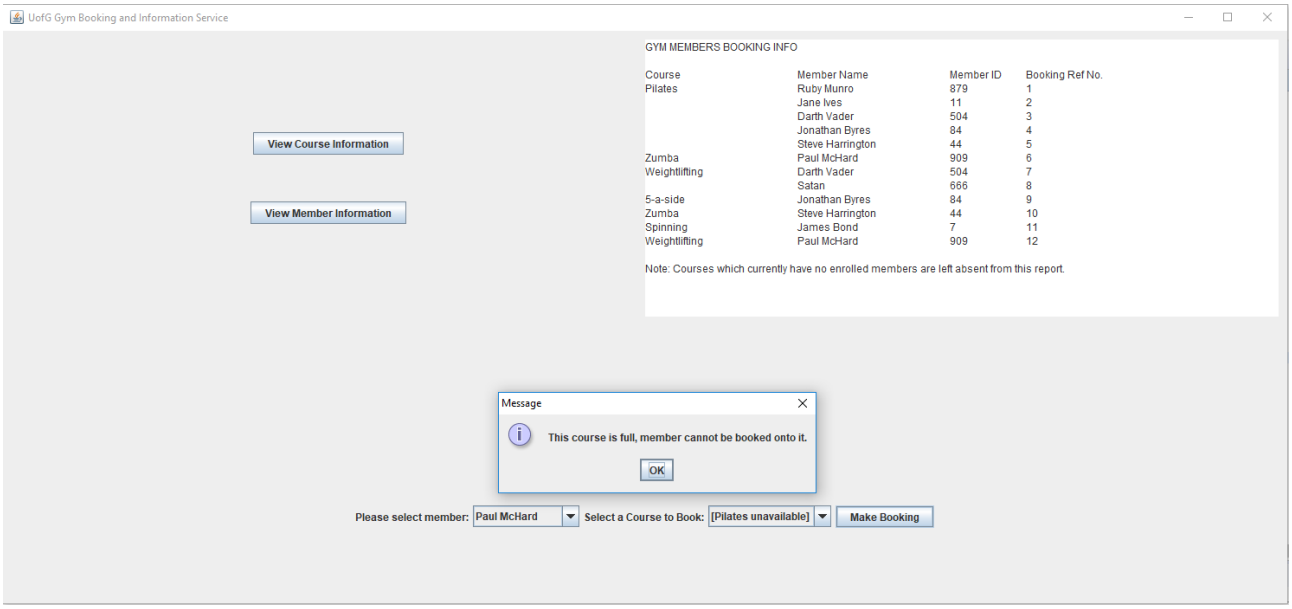


*Figure 5*

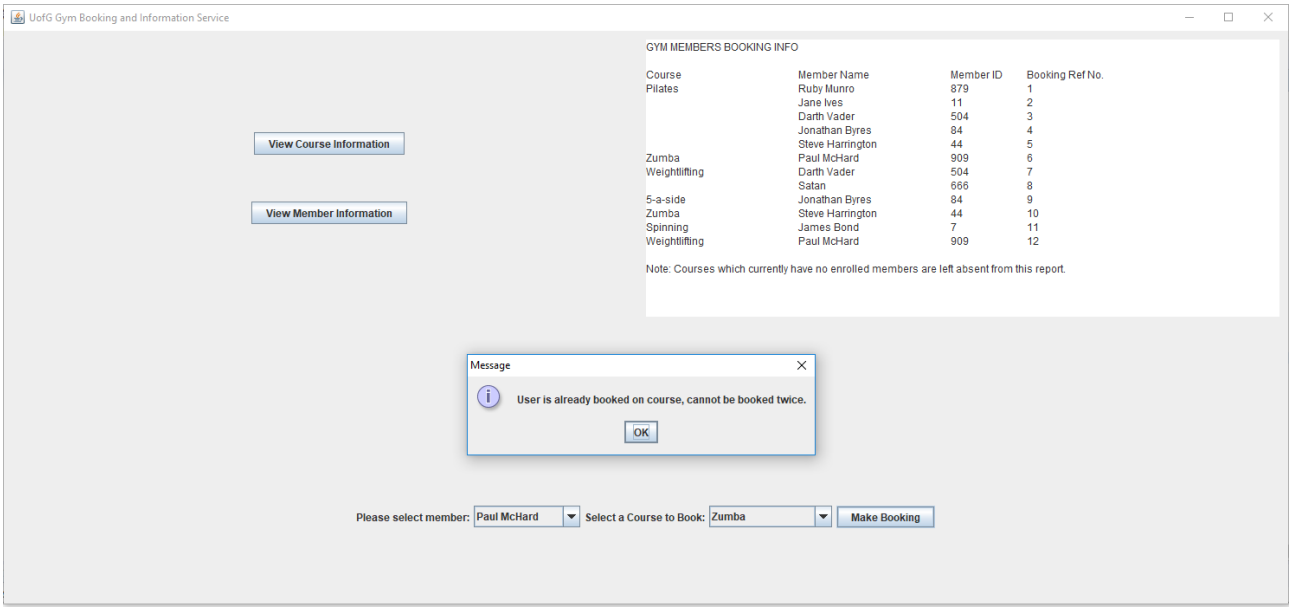


*Figure 6*