Matthew Henderson 2031944H

Data Theory and Applications Assessment

Functionality

The programme fully meets the specifications laid out in the brief. Users can view the details of any course, view the list of all members booked onto a course, and book any member onto a course. It performs validations to ensure that the same member can’t be booked onto a course twice, that the limit of 40 bookings cannot be exceeded, and that the course capacity cannot be exceeded. The design decision was taken to validate database entries with Java code, and not with database constraints.

GUI and SQL statements

Figure 1 in Appendix 1 shows the main GUI window that is displayed when the programme is run. It contains a JComboBox containing all the courses which the gym offers. To populate this component, the following SQL statement is used to retrieve the name and id of each course from the database. Note, to avoid confusion all queries are shown as they are appear in the .java file.

```
"SELECT name, courseID FROM Courses";
```

The name and courseID of each course is used to instantiate a course object. An array of all the course objects is then passed to the GUI and getter methods are used to display the course names. Similarly, the following SQL statement is used to retrieve all the gym members from the database. This time an array of Member objects is passed to the GUI. Again getter methods are used to display the name of each member in the relevant JComboBox.

```
"SELECT fname, sname, memberID FROM Member";
```

In the GUI in figure 1, two buttons can be seen next to the course JComboBox. The first is the details button. Upon pressing the details button, the following SQL queries are executed:

```
"SELECT day, time, cost, capacity, fname, sname "
+ "FROM Courses "
+ "INNER JOIN Instructor "
+ "ON instructorID = instructor "
+ "WHERE courseID='" + courseID + "'";
```

```
"SELECT COUNT(member) "
+ "FROM MemberCourses "
+ "WHERE course = '" + id + "'";
```

The query on the left retrieves all the details relating to the selected course. It also performs an inner join on the instructor ID so that the first name and last name of the instructor can be retrieved. The query on the right gets the total number of people that are currently booked onto that courses. The results from these queries are set as instance variables in the course object, this is in turn used to populate a new JFrame displaying the details. This JFrame can be seen in figure 2.

The second button seen in figure 1 is the class button. This displays a table of all the members that are currently booked onto the selected course. Upon clicking the button, a query is used to return an array of all the members booked onto the selected course. This information is then passed to a new JFrame. This JFrame can be seen in figure 3. The query used to get the member information can be seen below.

```
"SELECT memberID, fname, sname "
+ "FROM Member "
+ "INNER JOIN MemberCourses "
+ "ON memberID = member "
+ "WHERE course = '" + id + "'";
```

The final button on the GUI is the book button. When pressed it books the currently selected member onto the currently selected course. It does so using the following SQL statement.

```
"INSERT INTO MemberCourses(member, course) Values (" + memberID + "," + courseID + ")";
```

The final query that was used to add functionality to the programme can be seen below. This query returns the total number of booking that have been made so that maximum limit of 40 booking cannot be exceeded.

```
"SELECT COUNT(booking_number) FROM MemberCourses";
```

Testing

The programme was tested to ensure that it correctly dealt with a number of different scenarios. Table 1 below show a selection of the test scenarios used.

Table 1

Scenario	Result	As Expected?
Member is already booked on the course	An error message was displayed (figure 4) and the booking was not made	Yes
The course is at full capacity	An error message was displayed (figure 5) and the booking was not made	Yes
Cannot connect to database	Error message was displayed (figure 6) and the GUI was not filled with data	Yes
Already 40 bookings in the system	Error message was displayed informing the user and the booking was not made.	Yes
Member booked onto course	The MemberCourses table of the database was updated and the member was shown in the class list.	Yes
Details button pressed	The full course details were shown in a separate JFrame. These matched the values in the database.	Yes
Class button was pressed	A list of the members booked on the course was shown in a separate JFrame. These matched the database	Yes

Conclusion

The programme implements a course booking system for a gym. It allows users see the details of each course, view a list of all the members that are booked onto a course, and finally book any member on the system onto a course. The programme also performs validations on the bookings that are made.

There is currently no way to delete a booking from the system. This is something that could be added if the programme were to be developed further.

The main thing that I have learned from the task is how to use a java database connectivity API to interact with a database. Although I understood all the SQL throughout the course, I did not understand until now how it was implemented in a real programme.

Appendix 1 – Screen Shots

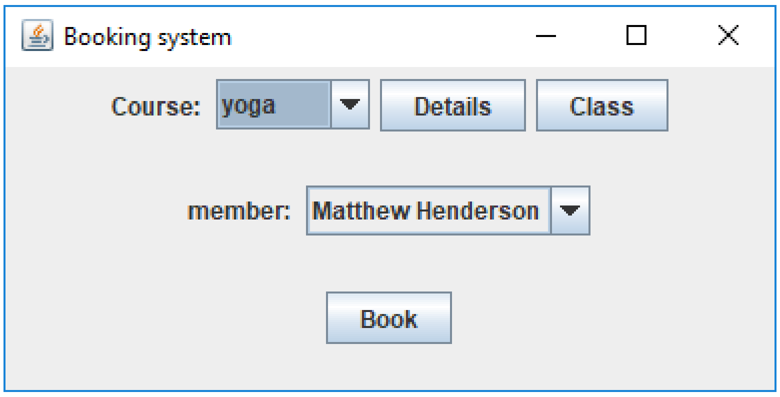
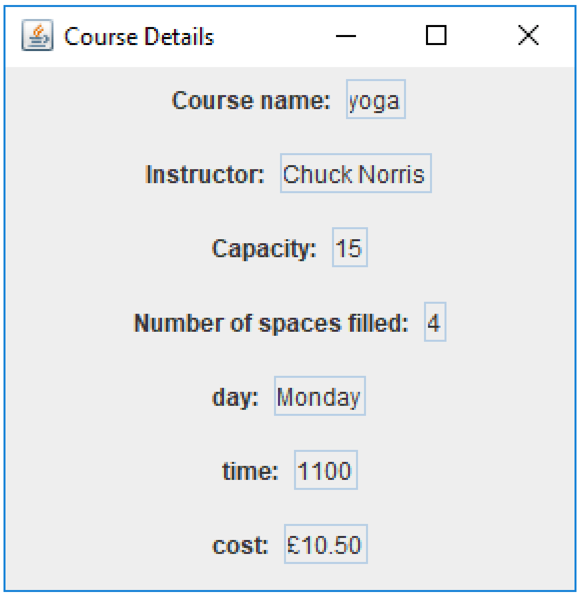


Figure 1. Main GUI Window



Course Details

Course name: yoga

Instructor: Chuck Norris

Capacity: 15

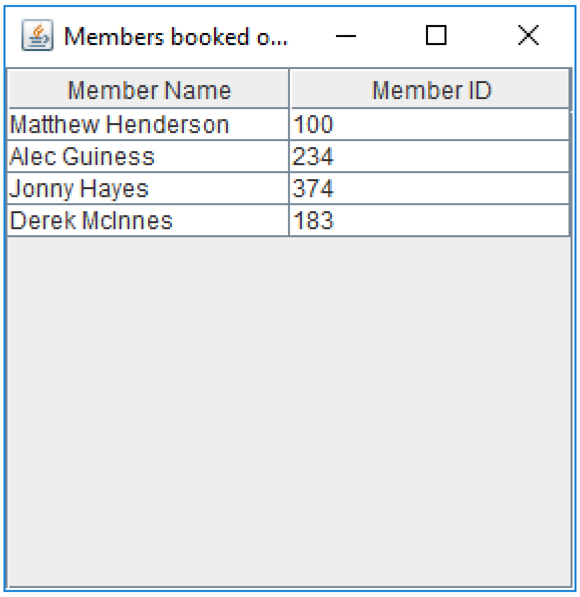
Number of spaces filled: 4

day: Monday

time: 1100

cost: £10.50

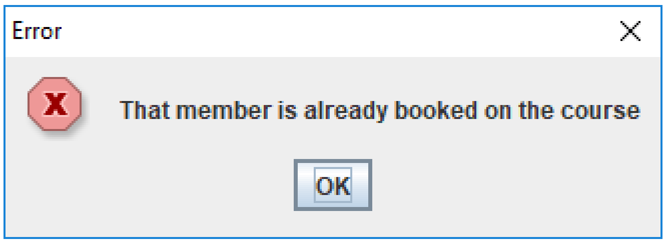
Figure 2. Course Details JFrame



Members booked o...

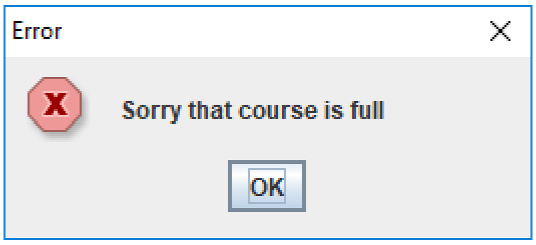
Member Name	Member ID
Matthew Henderson	100
Alec Guiness	234
Jonny Hayes	374
Derek McInnes	183

Figure 3. Members Booked On Courses


Error

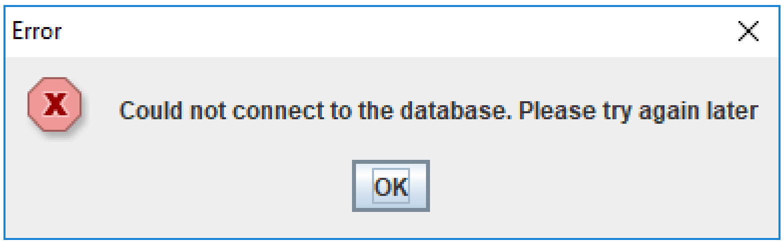
That member is already booked on the course

Figure 4. Member Already On Course Error Message


Error

Sorry that course is full

Figure 5. Course Full Error Message


Error

Could not connect to the database. Please try again later

Figure 6. Connection Failed Error Message

Appendix 2 – Database Schema

Member(memberID, fname, sname, membership_type, email, phone)

Instructor(instructorID, fname, sname, email, phone)

InstructorQualifications(instructor, qualification)

Course(courseID, name, day, time, cost, capacity, instructor)

MemberCourses(member, course, booking_number)

Facility(name, cost)

MemberFacility(facility, member, booking_number, date, time, duration, activity)