

Programming and Program Design Report and Class Description
Student Lesson Booking System
Module Code: 7WCM0041
Sam Stokes

Problem Statement/Description

The assignment will test my knowledge and understanding of the key principles of Object-Oriented Programming in Java and challenge me to implement a successful design and solution to the given task. I am required to create a system/software for managing lesson bookings made by students with the ability to book, change, cancel and attend lessons whilst adhering to certain criteria such as restricting duplicate bookings and student attendance limits. The students should be able to give reviews and ratings for lessons attended and reports should be made available for a user to print in relation to these statistics every month. The system should be able to handle this criteria and store relevant data for the reports. The IDE I will use for this project is IntelliJ IDEA.

Assumptions

Upon studying the assignment brief, a few assumptions can be observed:

- There is no requirement for a log in system such as generating a student log in and password.
- There does not need to be a sign up/register function, meaning students can be created within the program itself.
- Because of these, the use of an external database is not necessary.
- The program needs only show a live working system with live data and all necessary criteria for one weekend. The rest of the data can be hard coded for the output reports.
- The output reports require the number of students per tuition lesson along with an average rating and the highest income. Ratings therefore will be compulsory for every student to complete. Reviews by students will be optional, as a report for reviews is not said to be compulsory in the brief – but required in the program regardless. Because of this, reports on reviews for each tuition lesson will still be available in the program.
- Reviews are assumed to be anonymous as in a real-world program, it would not be ethical for the user to know who said and rated what.
- As the developer, discretion over the subjects used is available.
- There does not need to be a separate admin log in to print reports. It is assumed the program can be run as a student and a user for reports simultaneously.
- As there needs to be at least 8 weekends of timetable, the whole year does not need to be hard coded.
- The lessons for Saturday and Sunday run at the same time every week
- A student only pays for a lesson once they have attended that lesson

Pseudocode

Starting the Program

1. Start the program. This will be the home screen.
2. User inputs a number to book a lesson, change a lesson, cancel a lesson, attend a lesson, print reports or exit the program.

Book a Lesson

1. Input number to start the lesson booking process.
2. User chooses a number to choose either to book on Saturday or Sunday.
3. User will then choose a number to choose to book at a specific time. Morning, Afternoon, Early Evening or Late Evening.
4. Upon entering a number, the system will display the lesson for the day and time chosen.
 - The system will then ask the user if they want to book onto the lesson. If the user selects 1, the system will check to see if the user is already booked onto the lesson to avoid a duplicate booking.
 - The system will also check to see if the maximum limit for the number of students booked onto the lesson has been reached.
 - If not, then the student will be booked onto the lesson successfully.
 - If the user selects 2, then the system will take no action and return to the home screen.

Change a Lesson

1. Input number to start the change lesson process.
2. User chooses a number to choose what day the lesson is booked for.
3. User will then choose a number to choose what time the lesson is booked for. Morning, Afternoon, Early Evening or Late Evening.
4. The system will then display that lesson for the chosen day and time.
 - The system will then check to see if the user is booked on to that lesson.

- If the student is booked onto the lesson, the system will confirm if the user wants to change the lesson.
- If the user selects yes, then the student will be removed from the lesson, and will run through the Lesson Booking process.
- If the student is not booked onto the lesson, they will be given the choice to book onto the given lesson.
- If the user selects 2 for no, then the Home Screen will be displayed.

Cancel a Lesson

1. Input number to start the Lesson Cancellation process
2. User chooses a number from to choose what day the lesson is booked for.
3. User will then choose a number from to select what time the lesson is booked for. Morning, Afternoon, Early Evening, Late Evening.
4. The system will display the Lesson for the given day/time.
5. The system will check to see if the student is booked onto the lesson.
 - If student is booked onto the lesson, it will ask the user if they want to cancel the lesson.
 - If user inputs 1, the student will be removed from the lesson. If the user inputs 2, no changes will be made.
6. If student is not booked on, no changes will be made and the system will return to the home screen.

Monthly Lesson Report

1. Input number to start the Monthly Lesson Report process.
2. User enters a month number from 1 (January) – 12 (December).
3. The system will display the month and weeks selected.
 - Along with the average monthly rating of each lesson
 - And the number of students who gave each rating.
4. The system will then ask if the user would like to see any lesson reviews.
 - If yes, the system will ask for which month again and will display the reviews for each lesson.

Monthly Champion Lesson Report

1. Input number to start the Monthly Champion Lesson Report
2. User enters a month number from 1 (January) – 12 (December).
3. The system will display the month and weeks selected.
 - Each lesson for that month will be displayed with the lesson that has generated the highest income for that month.

Program Description

It was decided to use one student who would act as the basis of the whole program and would be used to show what the program is capable of in a live example. This one student will cover all criteria required. This student will be used to show:

- A successful lesson booking.
- A successful change of lesson booking.
- A successful cancellation of a lesson booking.
- A successful attending of a lesson. i.e. the student can leave a rating and a review.
- The reviews/ratings will be visible in the output reports.
- The total income of a lesson will increase if the student is booked onto a lesson and visible in the output reports.
- A lesson cannot be booked if the student limit is reached.
- A lesson cannot be booked if the student is already booked onto the lesson.
- A rating/review cannot be left if the student is not booked onto the lesson.

The thought process behind this approach is that this one student can cover all criteria required by the assignment brief. If one student can cover all of the criteria, then the program will be functional for multiple student objects. In relation to the output reports and the teaching timetable covering 8 weekends, student objects were booked onto lessons via hardcoding, as well as ratings and reviews. It was decided to use the months of January and February to cover the 8 weekends of timetable and output reports, and the month of March to cover the live program. This is demonstrated and outlined using examples and screenshots in this report.

Design and Class Descriptions

BookingController

This class contains the method used to run the program (and hold all of the other methods from other classes together) and contains the three methods which are used to book, change and cancel a lesson. They are:

- `public static void runProgram(Student studentTest)`
- `public static void bookLesson(Student studentTest)`
- `public static void changeLesson(Student studentTest)`
- `public static void cancelLesson(Student studentTest)`

The class is associated with the StudentController class and the Reports class, as the method runProgram uses the methods contained in these classes. It is also associated with the Student class, as the methods use a Student object as a parameter of the methods and the Lists for the subjects are also contained there.

runProgram()

This method is used as the home screen and is where the user can access all relevant methods of the program. It is the method used to take the user back to the Home Screen when used at the end of a series of statements so the program does not just finish and stop running. Therefore, it acts as a loop when used in the program until the user inputs the number 7 to exit and terminate the program. A series of if...else statements are used with int conditions to run a method based on user input – using the Scanner class found in the java.util package. For example, if the user enters the number '2', then the changeLesson method will run.

The class uses a series of attributes. These attributes are:

- `String[] daySelection = {"Saturday", "Sunday"};`
- `String[] timeSelection = {"Morning", "Afternoon", "Early Evening", "Late Evening"};`
- `String[] saturdayLessons = {"English", "Maths", "Science", "French"};`
- `String[] sundayLessons = {"Maths", "Science", "French", "English"};`

bookLesson()

This method is used to book a student onto a lesson. The method starts by asking a user what day they would like to book using a for loop to scroll through the daySelection array of Strings. The method uses the Scanner class to assign the user input to a variable. The variable holds the number corresponding to the element position within the Array. The variable is also used for the conditions of the if...else statements. This technique is used throughout the method using different variables for different conditions.

These variables and conditions enable the user to choose to book on either Saturday or Sunday, choose a time of day they would like to book and display the lesson for that time and book the lesson depending on whether the choice number is equal to the lesson (using the `.equals()` method) after passing a series of conditions.

The method will then go through the if...else statements to find the matching String using the variable. For example:

```
if (choice3 == 1 && saturdayLessons[choice2].equals("English"))
```

Once the String has been found, this is where the conditions come in place. The first condition checks for whether the student is already booked onto the lesson by using the `.contains()` method on the corresponding lesson list (found in the Student Class) using the parameters of the student the program is testing for – if this is true, a message is displayed and the program starts from the home page. If this is false, the next condition is tested, being whether the lesson has reached it's max capacity of 5 by using the `.size()` method with the condition being `>= 5`. If this is true, the lesson is fully booked and the program goes back to the homepage. If this is false, the student is added to the relevant lesson list found in the Student class using the `.add()` method (for example, `Student.studentListEnglishSat`).

changeLesson()

This methods process is the same as bookLesson, but differs when the conditions are being checked for a lesson. When a lesson is found after user input, the method will check to see if the user is booked onto that lesson using the `.contains()` method. If true, then the system will ask the user if they want to change the lesson. If the user selects yes, then the student is removed from the relevant lesson List in the Student class using the `.remove()` method and the bookLesson method is then run to begin the booking of a new lesson.

The situation of a user running this method and selecting a lesson that they are not booked on to is covered, as the user will have the choice to book onto the lesson if the `.contains` condition that checks the subject list for the student object returns false. The method will not at this point check for whether the student is booked onto the lesson (as that has already been done) but will simply check to see if there is still space using the `.size()` method. If the size is `>= 5` then the student will not be booked on. If not, the student will be booked on as normal.

cancelLesson()

This method's process is the same as `bookLesson` also. The only difference is that there is only one condition that is checked when the lesson is found by the system. The condition checks to see if the student is booked onto the lesson. If this returns true, the user will be asked to confirm that they want to cancel the lesson through user input and assigning the number to a variable – the `.remove` method is then used to remove the student from the relevant subject student list. If the condition is false, then the `runProgram` method is initialized and the home screen is displayed.

StudentController

This class contains the method for attending lessons, which is also where a user can add ratings and reviews for lessons they have attended.

attendLesson()

The method starts by asking the user what day they would like to book using a for loop to scroll through the `daySelection` array of Strings – an association with the `BookingController` is here as the `daySelection` attribute is used from the `BookingController` class. A for loop to display the time of day is also used here with the attribute `timeSelection` from the `BookingController` class. The method uses the `Scanner` class to assign the user input to a variable.

The process for selecting a lesson to attend is the same in relation to the variables, conditions and if...else statements as those found in the `bookLesson` method. The method will find the lesson for the date and time input by the user and ask the student whether they attended that lesson or not. If yes, the program will then check if that is true by using the `.contains()` method on the relevant lesson List. This is to ensure that not just any student can leave a rating on a lesson they did not attend.

At this point, a variable is used for the `scanner.nextInt()` method, which will only hold the user input value given by the user. The user is asked to give a rating from 1 to 5 inclusive. A condition is then used here to ensure the List used in the `Rating` class for the relevant lesson is not populated with any other numbers that are not 1 – 5. If the number input by the user is 1 – 5, then the list is populated with the rating. Otherwise, an error message is shown.

```
int rating = scanner.nextInt();
if (rating >= 1 && rating <= 5) {
    System.out.println("You have given English the rating of: " + rating);
    Rating.ratingEnglishMarch.add(rating);
}
```

This shows an association between the `StudentController` class and the `Rating` class.

As per the assumptions discussed, ratings are compulsory. After a rating has successfully been given, the user is asked if they would like to give a rating or not. Again, the variable is assigned a number, if the number matches that of the condition, then the answer is yes, and the if...else statement is entered.

I encountered a problem here when trying to assign the String entered by user input using the `scanner.nextLine()` function. The program would not allow me to enter a review. It appeared to be assigned an empty value of some sort. Upon further research, the variable I was trying to assign was possibly getting data from a new line character. I therefore included a variable named 'exception' to assign that data before the user input. I then placed the variable 'review' after to store the user input which fixed this issue.

```
if(choice4 == 1) {
    System.out.println("Please write a few sentences about how you found the lesson: ");
    String exception = scanner.nextLine(); //This is to avoid an exception of no user
input.
    String review = scanner.nextLine();
    Review.reviewMathsMarch.add(review);
}
```

If the student is not booked onto this lesson, then there is no chance for a student to leave a rating or a review for any lessons.

Main

This class contains the main method, and is the class used to add the students, ratings and reviews for the live example of the program for the month of March (week 9). The attributes of this class are HashMaps - which are used together in a nesting structure to categorise and differentiate the various student lists, time of day, days and week and Array lists of Strings to hold data to be included in the HashMaps. This means that the week can be printed, and shows all students who are booked onto a lesson on both Saturday and Sunday for that week. This will be demonstrated below.

HashMaps are part of Java's collections and provides basic implementation of the Map interface of Java. It holds data in a Key, Value pair. I decided to use HashMaps to store data, as it is versatile with the typing's you can store and thought it would be useful to group items by certain property.

I began by populating two Arrays with String elements covering the lessons for the time of day for both Saturday and Sunday. As per the assumptions, these do not change, so only one Array of each for all weeks would be required.

```
private static String[] saturdayLessons = {"Morning - English", "Afternoon - Maths", "Early Evening - Science", "Late Evening - French"};
private static String[] sundayLessons = {"Morning - Maths", "Afternoon - Science", "Early Evening - French", "Late Evening - English"};
```

I then created two most inner HashMaps that would be used to store the student lists and the subject/time of day. The idea being that the key would be for example "Morning – English" which would then hold the value of the List of Students booked onto that lesson for Saturday.

Adding a student to the English List for Saturday (hard coding).

```
Student.studentListEnglishSat.add(Student.one);
```

I discovered to add students to the list which is found in the Student Class (association), it was required to be in a static state, which is why for hardcoding week 9 for March to use as the live example, Students added to their respective lesson list, rating list and review list were added in the main method.

Time and Subject - Student List

```
protected static Map<String, List<Student>> timeAndSubjectSun = new HashMap<>();
protected static Map<String, List<Student>> timeAndSubjectSat = new HashMap<>();
```

Adding Time and Subject and corresponding Student List to the HashMap variable for Saturday

```
timeAndSubjectSat.put(saturdayLessons[0], Student.studentListEnglishSat);
timeAndSubjectSat.put(saturdayLessons[1], Student.studentListMathsSat);
timeAndSubjectSat.put(saturdayLessons[2], Student.studentListScienceSat);
timeAndSubjectSat.put(saturdayLessons[3], Student.studentListFrenchSat);
```

Adding Time and Subject and corresponding Student List to the HashMap variable for Sunday

```
timeAndSubjectSun.put(sundayLessons[0], Student.studentListMathsSun);
timeAndSubjectSun.put(sundayLessons[1], Student.studentListScienceSun);
timeAndSubjectSun.put(sundayLessons[2], Student.studentListFrenchSun);
timeAndSubjectSun.put(sundayLessons[3], Student.studentListEnglishSun);
```

Day - Time and Subject - Student List

```
private static Map<String, Map<String, List<Student>>> saturdayDay = new HashMap<>();
private static Map<String, Map<String, List<Student>>> sundayDay = new HashMap<>();
```

Adding the HashMap 'timeAndSubjectSat' with the Key of "Saturday" to another HashMap called 'saturdayDay'. When 'saturdayDay' is printed, it will show all of the students booked onto every lesson for that day. This is also the same for Sunday. This is an example of using nested HashMaps.

```
saturdayDay.put("Saturday\n", timeAndSubjectSat);
sundayDay.put("\nSunday\n", timeAndSubjectSun);
```

Saturday - Sunday

```
protected static Map<Map, Map> saturdayAndSunday = new HashMap<>();
```

This is joining both Saturday and Sunday together into one collective.

```
saturdayAndSunday.put(saturdayDay, sundayDay);
```

Week Number - Weekend

```
protected static Map<String, Map<Map, Map>> week9 = new HashMap<>();
private static String[] weekNo = {"\nWeek 1", "\nWeek 2", "\nWeek 3", "\nWeek 4", "\nWeek 5", "\nWeek 6", "\nWeek 7", "\nWeek 8", "Week 9\n"};
```

This is the final level in relation to the nested HashMaps. This is where the week number along with all of the lessons with students booked onto them are stored.

```
week9.put(weekNo[8], saturdayAndSunday);
```

When the program is run using a Student object, the student can book on to a lesson (as long as the criteria allows) and the week9 HashMap variable will get updated with the new student for that week. This is demonstrated below by printing the week9 elements before the program has run and after the program has run.

Print out of the Students Booked on to Saturday and Sunday Lessons Before the Program is Run

```
java x BookingController.java x StudentController.java x Student.java x Rating.java x

//Start program from here
System.out.println(week9);
BookingController.runProgram(Student.test);
System.out.println(week9);

Main > main()

Main x
Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath
C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
{Week 9
={Saturday
={Afternoon - Maths=[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe'], 2
Late Evening - French=[Student: id = '11', studentName = 'Jordan Loren', Student: id = '14', studentName = 'Johnnie Fuller', Student: id = '15', studentName = 2
Didier Smith'], Morning - English=[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName = 2
Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson', Student: id = '16', studentName = 'Userinput Test'], Early Evening - Science=[Student: id = '9',2
studentName = 'Jason Webb', Student: id = '7', studentName = 'Jean Clayton', Student: id = '13', studentName = 'Frank Lampard']]}={
Sunday
={Afternoon - Science=[Student: id = '14', studentName = 'Johnnie Fuller', Student: id = '8', studentName = 'Jane Doe'], Morning - Maths=[Student: id = '10',
studentName = 'Vincent Elliot', Student: id = '13', studentName = 'Frank Lampard', Student: id = '3', studentName = 'Kenny Kormack'], Late Evening -
English=[Student: id = '6', studentName = 'Dean Tate', Student: id = '8', studentName = 'Jane Doe'], Early Evening - French=[Student: id = '7', studentName =
Jean Clayton', Student: id = '12', studentName = 'Roy Floyd', Student: id = '9', studentName = 'Jason Webb']]}}}}

Extra Tuition Centre Student Booking System

-----

You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson

n TODO Terminal Version Control Messages
Completed successfully in 3 s 820 ms (a minute ago) 113 chars, 1 line break 114:1 CRLF UTF-8 4 spaces Git: master
```

Here I have underlined the Students booked onto French on Sunday. Notice how the Student object 'test' is not booked onto this lesson and is not on the List.

```
java x BookingController.java x StudentController.java x Student.java x Rating.java x

//Start program from here
System.out.println(week9);
BookingController.runProgram(Student.test);
System.out.println(week9);

Main > main()

Main x
4. Attend Lesson
5. Monthly Lesson Report
6. Monthly Champion Report
7. Exit Program

Enter a Number: 7
End of Program
{Week 9
={Saturday
={Afternoon - Maths=[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe'],
Late Evening - French=[Student: id = '11', studentName = 'Jordan Loren', Student: id = '14', studentName = 'Johnnie Fuller', Student: id = '15', studentName =
Didier Smith'], Morning - English=[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName =
Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson', Student: id = '16', studentName = 'Userinput Test'], Early Evening - Science=[Student: id = '9',
studentName = 'Jason Webb', Student: id = '7', studentName = 'Jean Clayton', Student: id = '13', studentName = 'Frank Lampard']]}={
Sunday
={Afternoon - Science=[Student: id = '14', studentName = 'Johnnie Fuller', Student: id = '8', studentName = 'Jane Doe'], Morning - Maths=[Student: id = '10',
studentName = 'Vincent Elliot', Student: id = '13', studentName = 'Frank Lampard', Student: id = '3', studentName = 'Kenny Kormack'], Late Evening -
English=[Student: id = '6', studentName = 'Dean Tate', Student: id = '8', studentName = 'Jane Doe'], Early Evening - French=[Student: id = '7', studentName =
Jean Clayton', Student: id = '12', studentName = 'Roy Floyd', Student: id = '9', studentName = 'Jason Webb', Student: id = '16', studentName = 'Userinput
Test']]}}}}

Process finished with exit code 0

n TODO Terminal Version Control Messages
Completed successfully in 3 s 820 ms (9 minutes ago) 66:1 CRLF UTF-8 4 spaces Git: master
```

Here I have underlined the Students booked onto French on Sunday once again after the program has run. Notice now that the 'test' object has been added to the student List for Sunday.

Student

This class is where the Lists are held for the different subjects. There is a List initialized for each Lesson for either Saturday or Sunday. The lists are of type 'Student' in order to hold the Student objects which are also created in this class. The constructor allows for a Student ID and Name to be passed in using the fields when a new student object is created. This is where Students are added to or removed from a lesson list when the program is executed.

Reports/Rating/Review/Prices

The reports class is responsible for the two methods that print out the reports and use the Rating, Review and Prices classes also to achieve the required reports. Because of this, it has direct associations with the Rating, Review and Prices classes. The class contains two attributes - a scanner object for user input when selecting a month to print and a String Array containing the months of the year.

Both of the methods ask the user to select a number between 1 – 12 corresponding to the month number of the year. However, for this example, only January, February and March are used. The idea being that this would work the same way for all other months of the year but for the sake of this project, this has been limited to the first three months.

monthlyLessonReport()

The method asks for the user to input a number between 1 – 12 (as discussed, for the purpose of this project, only 1 – 3 can be used). If...else statements are used here to determine which month should be displayed based on user input.

Once a condition has been within the parameters of the if...else statement, the method will print out the monthly average rating for each lesson by using a function I developed called `getAverage()` found in the Rating Class. It works by passing in the list as a parameter, and calculates the average of all of the integers in the list. The report will also print the number of ratings in the list using the `.size()` method. This covers the requirement of displaying the average rating and displaying the number of students who gave those ratings.

The method then asks the user if they would like to view any reviews. If the user selects yes, then again, the user is required to input a month from 1 – 12. If...else statements with conditions are used the same way here, and the reviews are displayed for each lesson of that month from the Lists found in the Review class.

monthlyLessonChampionReport()

The method also asks for the user to input a number between 1 – 12 and uses if...else statements to display a certain month based on user input. The report displays each lesson for that month and the total income for each subject. The way this is done, is in the Prices class, a variable is given the value of the student subject list size for each day (Saturday and Sunday). This gives the total number of students for one lesson in a week.

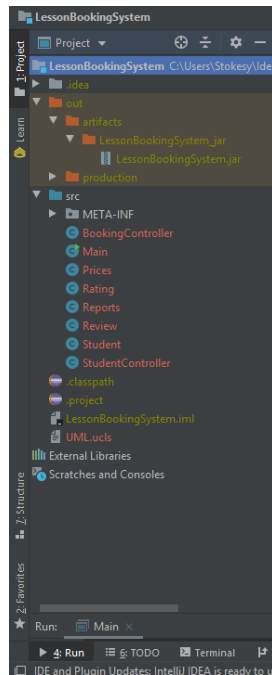
```
private static int englishSatSize = Student.studentListEnglishSat.size();  
private static int englishSunSize = Student.studentListEnglishSun.size();
```

These variables are then added together, and multiplied by the lesson price to give the total income for that week.

```
static int englishTotalMarch = (englishSatSize + englishSunSize) * englishPrice;
```

This is done for every lesson in the month of March. The lesson prices are declared and initialized at the top of the class as fields.

For the months of January and February, this is achieved slightly differently. Due to the assumption that it is compulsory for every student to leave a rating after a lesson is attended, the rating lists for the months of January and February for each lesson are used to get the total number of students for that lesson in that month with the `.size()` method. This is then multiplied by the price field of the corresponding lesson to get the total for each lesson of that month.



Running the Program

In my case, I am using windows, so the program is started from the Command Prompt window by using the 'java -jar' command, and then pasting the path where the program JAR file is located.

This is the 'Home Page' of the program. As you can see, the student object that is currently being run is displayed and is shown as 'logged on'.

```

C:\Users\Stokesy>java -jar C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\artifacts\LessonBookingSystem_jar\LessonBookingSystem.jar

Microsoft Windows [Version 10.0.17763.914]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Stokesy>java -jar C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\artifacts\LessonBookingSystem_jar\LessonBookingSystem.jar

-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson
3. Cancel a Lesson
4. Attend Lesson
5. Monthly Lesson Report
6. Monthly Champion Report
7. Exit Program

Enter a Number: 

```

This student object has the identifier 'test' and is found in the Student class. test is the object I will use to demonstrate the program in a live example, however, in the Main class and main method where the program starts to run, the parameters could be changed to any other student object, and will work the same way.

```
//Run Program from here.
BookingController.runProgram(Student.test);
```

Booking a Lesson

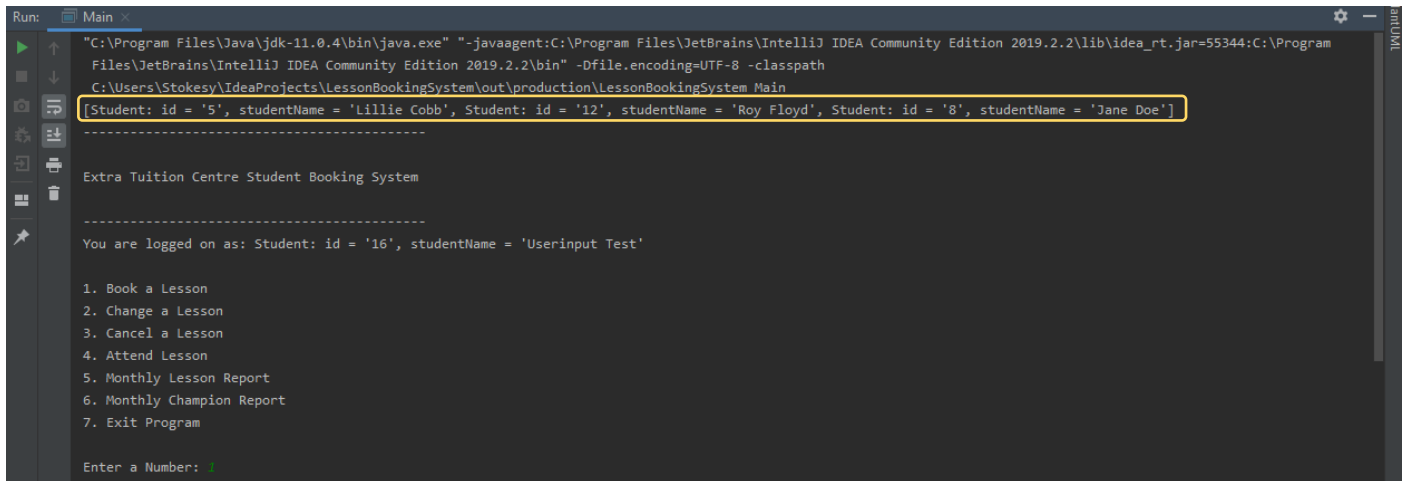
The following will demonstrate the student object 'test' successfully booking onto a lesson.

```
System.out.println(Student.studentListMathsSat);
BookingController.runProgram(Student.test);
System.out.println(Student.studentListMathsSat);
```

If the user runs the following code in the Main class using an IDE, the student list for the lesson Maths on Saturday will be printed before the program is run. The user will see that the 'test' student object is not present in the List when printed. The user can then use the program to book the student onto the Maths lesson for Saturday by inputting the following numbers

• 1 – 1 – 2 – 1

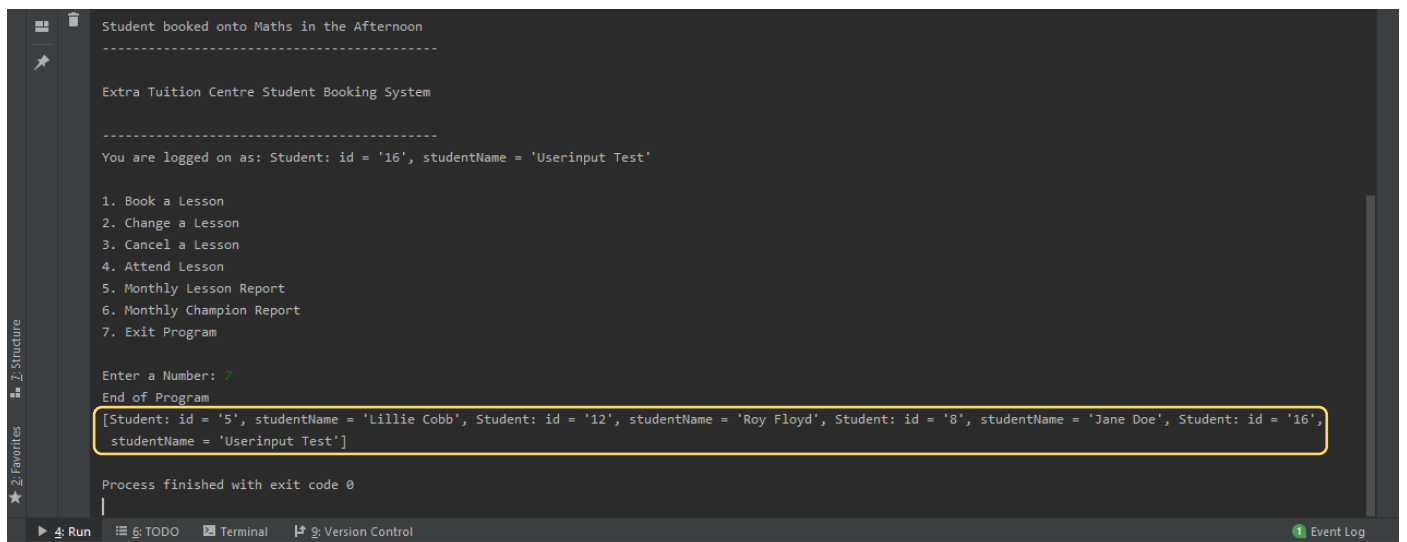
The user can then exit the program by pressing 7. The List will be printed again, and will now show the test object successfully booked onto the lesson and added to the relevant lesson List.



```
Run: Main x
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=55344:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe']
-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson
3. Cancel a Lesson
4. Attend Lesson
5. Monthly Lesson Report
6. Monthly Champion Report
7. Exit Program

Enter a Number: |
```



```
Student booked onto Maths in the Afternoon
-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson
3. Cancel a Lesson
4. Attend Lesson
5. Monthly Lesson Report
6. Monthly Champion Report
7. Exit Program

Enter a Number: |
End of Program
[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe', Student: id = '16', studentName = 'Userinput Test']

Process finished with exit code 0
```

The Student object 'test' is already hardcoded and booked onto the English Lesson on Saturday. Therefore, the following will demonstrate that a Student who is already booked onto a lesson cannot book onto a lesson again:

```
System.out.println(Student.studentListEnglishSat);
BookingController.runProgram(Student.test);
System.out.println(Student.studentListEnglishSat);
```

Running the above code will print out all of the Students booked onto English for Saturday. As you can see in the screenshot below, the Test object is already an element in the List.

```

public static void main(String[] args) {
    //Start program from here
    System.out.println(Student.studentListEnglishSat);
    BookingController.runProgram(Student.test);
}

```

```

"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=63384:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson', Student: id = '16', studentName = 'Userinput Test']
-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson
3. Cancel a Lesson
4. Attend Lesson

```

The user can then input the following to demonstrate that a Student cannot double book a lesson:

• 1, 1, 1, 1

A message displaying “Student already booked on” is shown. The user can then use the IDE to exit out of the program. The Student List will be printed again, where the Student test object has not been added to the List, thus avoiding duplicate bookings.

The following will also show that a Student cannot book onto a lesson if the lesson has already reached it’s maximum capacity of 5. The Maths lesson for Sunday has been hard-coded with 5 students. If the user runs the following code in the IDE, the Maths List for Sunday will be printed before the program is run.

```

System.out.println(Student.studentListMathsSun);
BookingController.runProgram(Student.test);
System.out.println(Student.studentListMathsSun);

```

```

public static void main(String[] args) {
    //Start program from here
    System.out.println(Student.studentListMathsSun);
    BookingController.runProgram(Student.test);
    System.out.println(Student.studentListMathsSun);
}

```

```

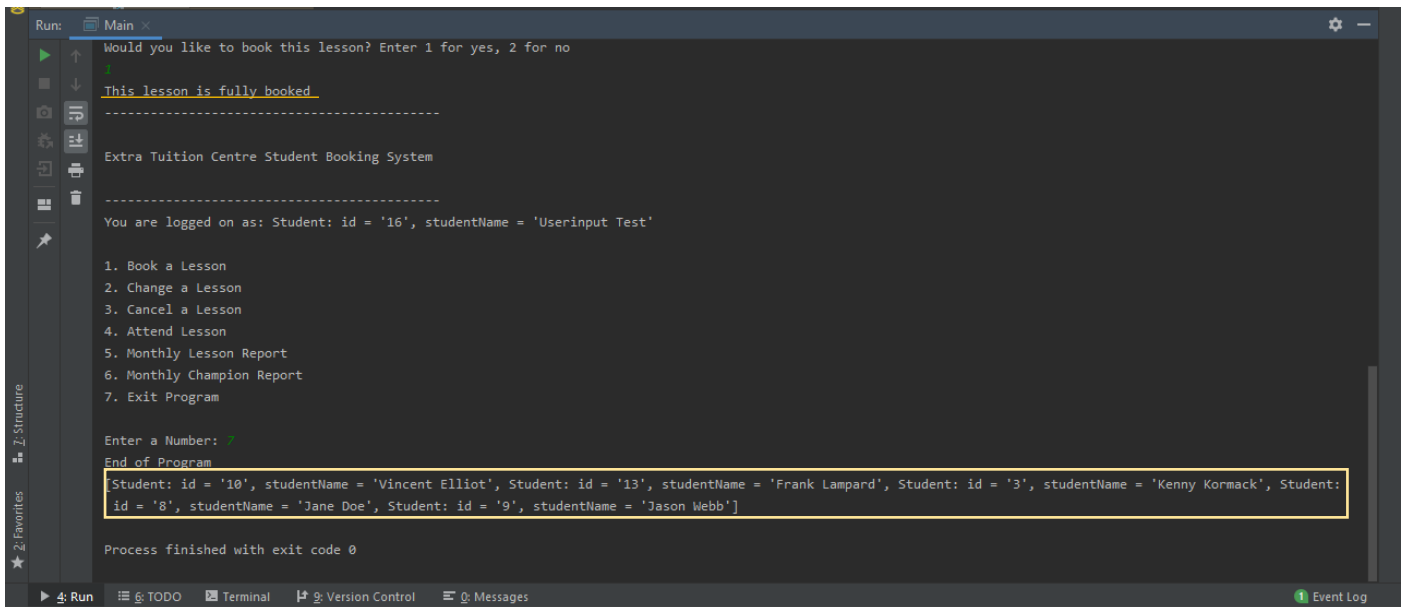
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=63576:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[Student: id = '10', studentName = 'Vincent Elliot', Student: id = '13', studentName = 'Frank Lampard', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '8', studentName = 'Jane Doe', Student: id = '9', studentName = 'Jason Webb']
-----
Extra Tuition Centre Student Booking System
-----

```

This is the print out of the list before the program runs. If the user inputs the following, the system will try to book the Student onto the list:

• 1, 2, 1, 1

The program will then print “This lesson is fully booked”, and the object will not be added to the list. If the user then exits the program, the list will again be printed.



```
Run: Main x
Would you like to book this lesson? Enter 1 for yes, 2 for no
1
This lesson is fully booked.
-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'

1. Book a Lesson
2. Change a Lesson
3. Cancel a Lesson
4. Attend Lesson
5. Monthly Lesson Report
6. Monthly Champion Report
7. Exit Program

Enter a Number:
End of Program
[Student: id = '10', studentName = 'Vincent Elliot', Student: id = '13', studentName = 'Frank Lampard', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '8', studentName = 'Jane Doe', Student: id = '9', studentName = 'Jason Webb']

Process finished with exit code 0
```

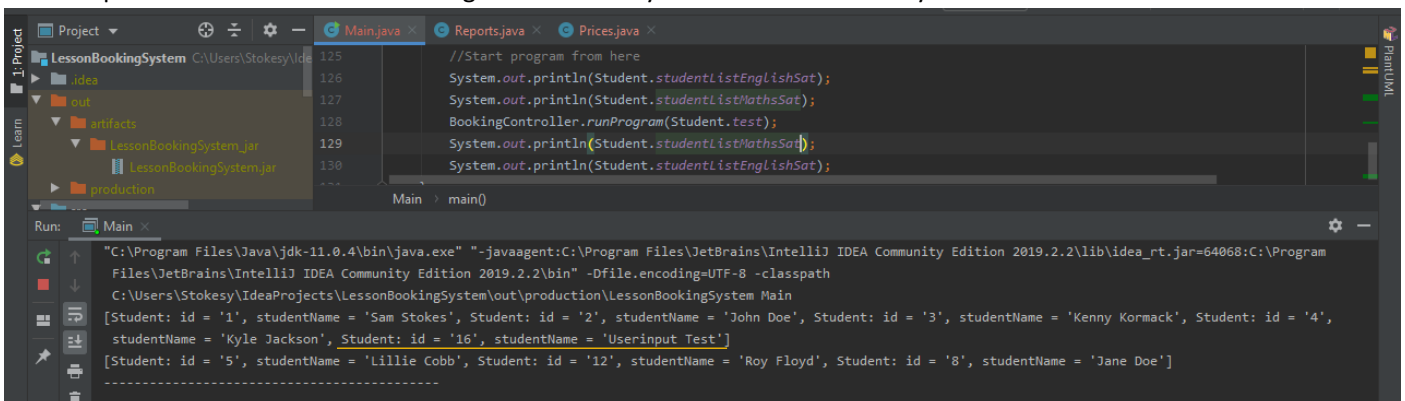
As shown, the student has not been added to the list.

Changing a Lesson

The following will demonstrate a student changing a lesson by running the following code in an IDE:

```
System.out.println(Student.studentListEnglishSat);
System.out.println(Student.studentListMathsSat);
BookingController.runProgram(Student.test);
System.out.println(Student.studentListEnglishSat);
System.out.println(Student.studentListMathsSat);
```

This will print out the Student List for English on Saturday and Maths on Saturday.



```
Project LessonBookingSystem C:\Users\Stokesy\Idea
  .idea
  .out
  artifacts
    LessonBookingSystem.jar
  production
    LessonBookingSystem.jar

Main.java x Reports.java x Prices.java x
125 //Start program from here
126 System.out.println(Student.studentListEnglishSat);
127 System.out.println(Student.studentListMathsSat);
128 BookingController.runProgram(Student.test);
129 System.out.println(Student.studentListEnglishSat);
130 System.out.println(Student.studentListMathsSat);
...
Main -> main()

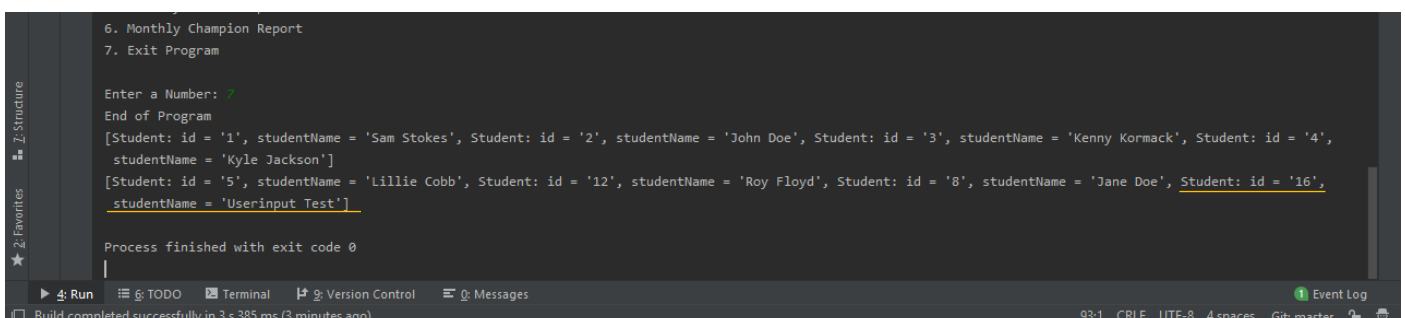
Run: Main x
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=64068:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson', Student: id = '16', studentName = 'Userinput Test']
[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe']
-----
Process finished with exit code 0
```

As you can see, the Student object 'test' is booked onto English, but not for Maths.

If the user then inputs the following

- 2, 1, 1, 1, 2, 1, 1

The System will check to see if the student is booked onto the English lesson, upon confirming they are booked on, the program will then remove the student from the list, and the bookLesson() method will start. Following the user input above, the Student will then be booked onto Maths. Thus a successful change of lesson booking. The screenshot below shows the test object has been removed from English, and added to Maths:



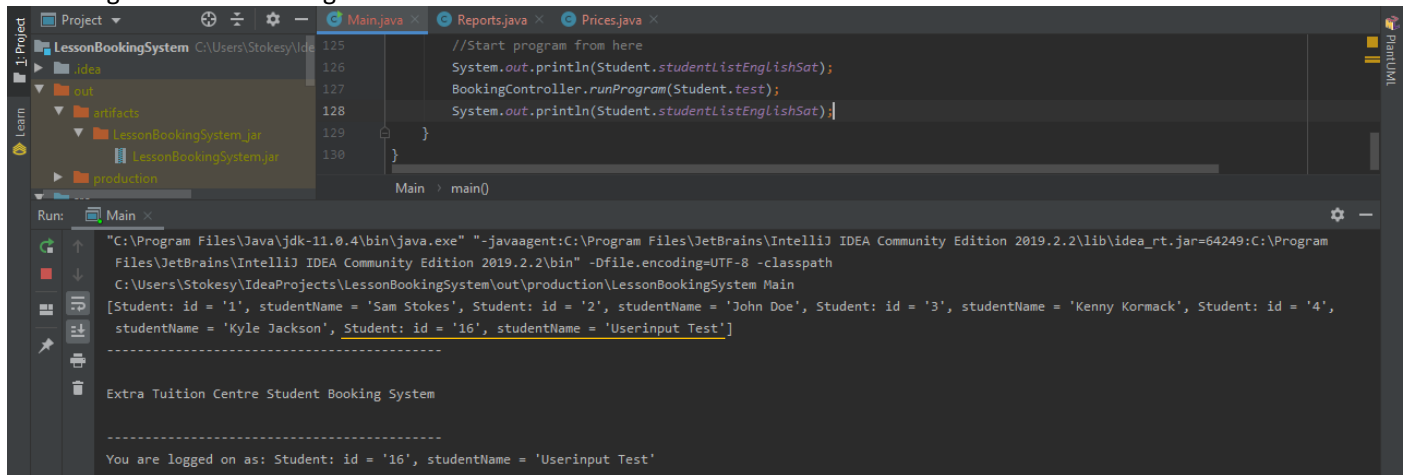
```
6. Monthly Champion Report
7. Exit Program

Enter a Number:
End of Program
[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson']
[Student: id = '5', studentName = 'Lillie Cobb', Student: id = '12', studentName = 'Roy Floyd', Student: id = '8', studentName = 'Jane Doe', Student: id = '16', studentName = 'Userinput Test']

Process finished with exit code 0
```

Changing a Lesson

A student may simply wish to just cancel a lesson, not change a lesson. The following will demonstrate a student cancelling a lesson and being removed from a list.



The screenshot shows the IntelliJ IDEA interface. The top pane displays the `Main.java` file with the following code:

```
125 //Start program from here
126 System.out.println(Student.studentListEnglishSat);
127 BookingController.runProgram(Student.test);
128 System.out.println(Student.studentListEnglishSat);
129 }
130 }
```

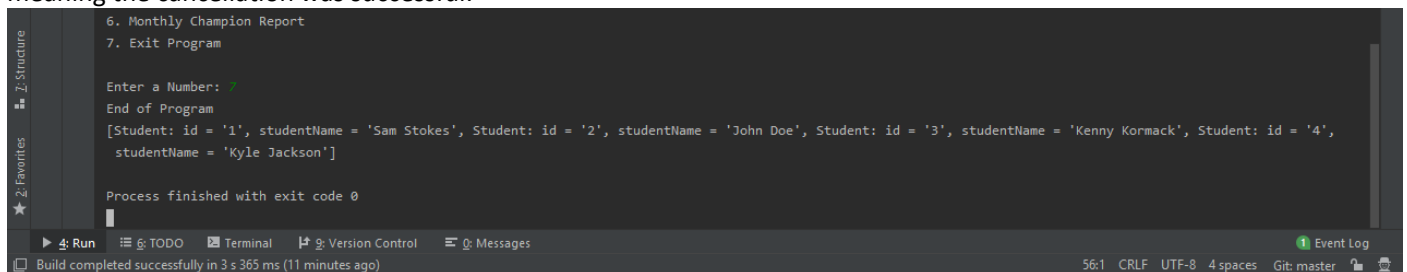
The bottom pane shows the Run console output for the `Main` class. The output includes the command used to run the program and the resulting student list:

```
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=64249:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[Student: id = '1', studentName = 'Sam Stokes', Student: id = '2', studentName = 'John Doe', Student: id = '3', studentName = 'Kenny Kormack', Student: id = '4', studentName = 'Kyle Jackson', Student: id = '16', studentName = 'Userinput Test']
-----
Extra Tuition Centre Student Booking System
-----
You are logged on as: Student: id = '16', studentName = 'Userinput Test'
```

A user can cancel a booking by inputting the following:

- 3, 1, 1, 1

After exiting the program, the Student List is printed once again, and the Student object 'test' has been removed, meaning the cancellation was successful:



The screenshot shows the IntelliJ IDEA interface. The top pane displays the `Main.java` file with the following code:

```
125 //Start program from here
126 System.out.println(Rating.ratingEnglishMarch);
127 System.out.println(Review.reviewEnglishMarch);
128 BookingController.runProgram(Student.test);
129 System.out.println(Rating.ratingEnglishMarch);
130 System.out.println(Review.reviewEnglishMarch);
131 }
```

The bottom pane shows the Run console output for the `Main` class. The output includes the command used to run the program and the resulting student list:

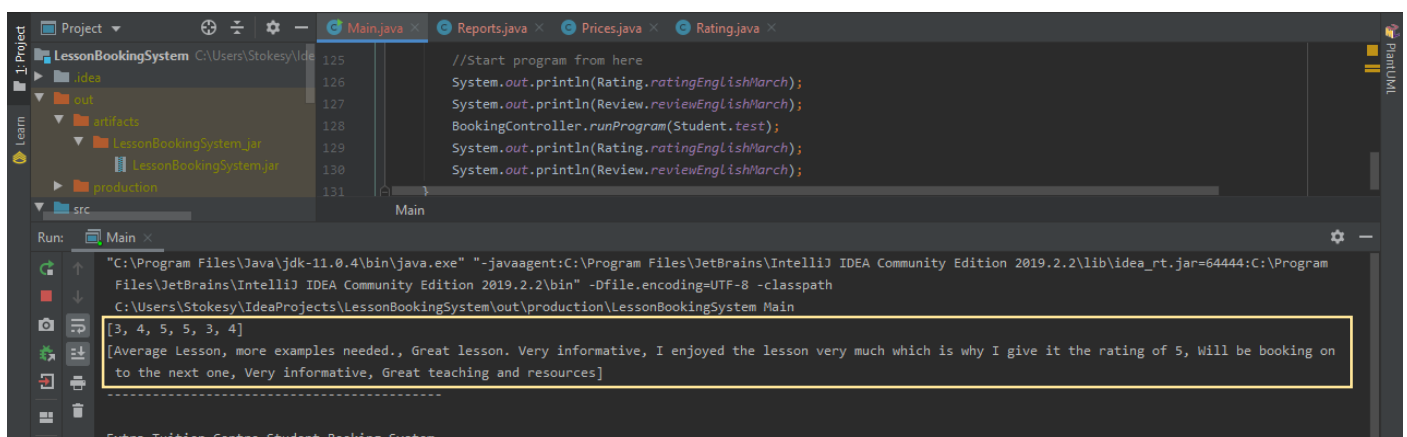
```
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=64444:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[3, 4, 5, 5, 3, 4]
[Average Lesson, more examples needed., Great lesson. Very informative, I enjoyed the lesson very much which is why I give it the rating of 5, Will be booking on to the next one, Very informative, Great teaching and resources]
-----
Extra Tuition Centre Student Booking System
-----
```

Attending a Lesson

A student is able to attend a lesson and leave a rating and a review of the lesson they attended. A user can run the following code in the main method to demonstrate this:

```
System.out.println(Rating.ratingEnglishMarch);
System.out.println(Review.reviewEnglishMarch);
BookingController.runProgram(Student.test);
System.out.println(Rating.ratingEnglishMarch);
System.out.println(Review.reviewEnglishMarch);
```

This will print the ratings given by students for English on Saturday and also print all of the reviews.



The screenshot shows the IntelliJ IDEA interface. The top pane displays the `Main.java` file with the following code:

```
125 //Start program from here
126 System.out.println(Rating.ratingEnglishMarch);
127 System.out.println(Review.reviewEnglishMarch);
128 BookingController.runProgram(Student.test);
129 System.out.println(Rating.ratingEnglishMarch);
130 System.out.println(Review.reviewEnglishMarch);
131 }
```

The bottom pane shows the Run console output for the `Main` class. The output includes the command used to run the program and the resulting student list:

```
"C:\Program Files\Java\jdk-11.0.4\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\lib\idea_rt.jar=64444:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Stokesy\IdeaProjects\LessonBookingSystem\out\production\LessonBookingSystem Main
[3, 4, 5, 5, 3, 4]
[Average Lesson, more examples needed., Great lesson. Very informative, I enjoyed the lesson very much which is why I give it the rating of 5, Will be booking on to the next one, Very informative, Great teaching and resources]
-----
Extra Tuition Centre Student Booking System
-----
```

The program will then run and the user will be able to leave a rating and a review by inputting:

- 4, 1, 1, 1,

The user can then enter a rating of 1 to 5 which will be added to the list. The average is calculated when the report is printed in a different function. After entering a rating, enter the number 1 to leave a review.

```
Please write a few sentences about how you found the lesson:
This is a review demonstration
Student not booked onto this lesson
[3, 4, 5, 5, 3, 4, 5]
[Average Lesson, more examples needed., Great lesson. Very informative, I enjoyed the lesson very much which is why I give it the rating of 5, Will be booking on
to the next one, Very informative, Great teaching and resources, This is a review demonstraion]

Process finished with exit code 0
```

Monthly Lesson Report

By running the code below, the user is able to print out the number of students and the average rating given by those students for every lesson of a given month. The user is also able to print the reviews for the lessons of that month also. Both the ratings, number of students and reviews are updated live using the Student object 'test' whilst running the program. Running the code below in the main method and following the user input demonstrates this. The 'test' student object is booked onto English Saturday, so this is the lesson we will be using for this example.

```
BookingController.runProgram(Student.test);
```

User Input:

- 5, 3 (Prints ratings for March), 1, 3 (Prints reviews for March)

```
Project
LessonBookingSystem
  .idea
  out
  artifacts
    LessonBookingSystem.jar
Main.java
BookingController.java
Reports.java
StudentController.java
Rating.java
Prices.java

//Start program from here
BookingController.runProgram(Student.test);

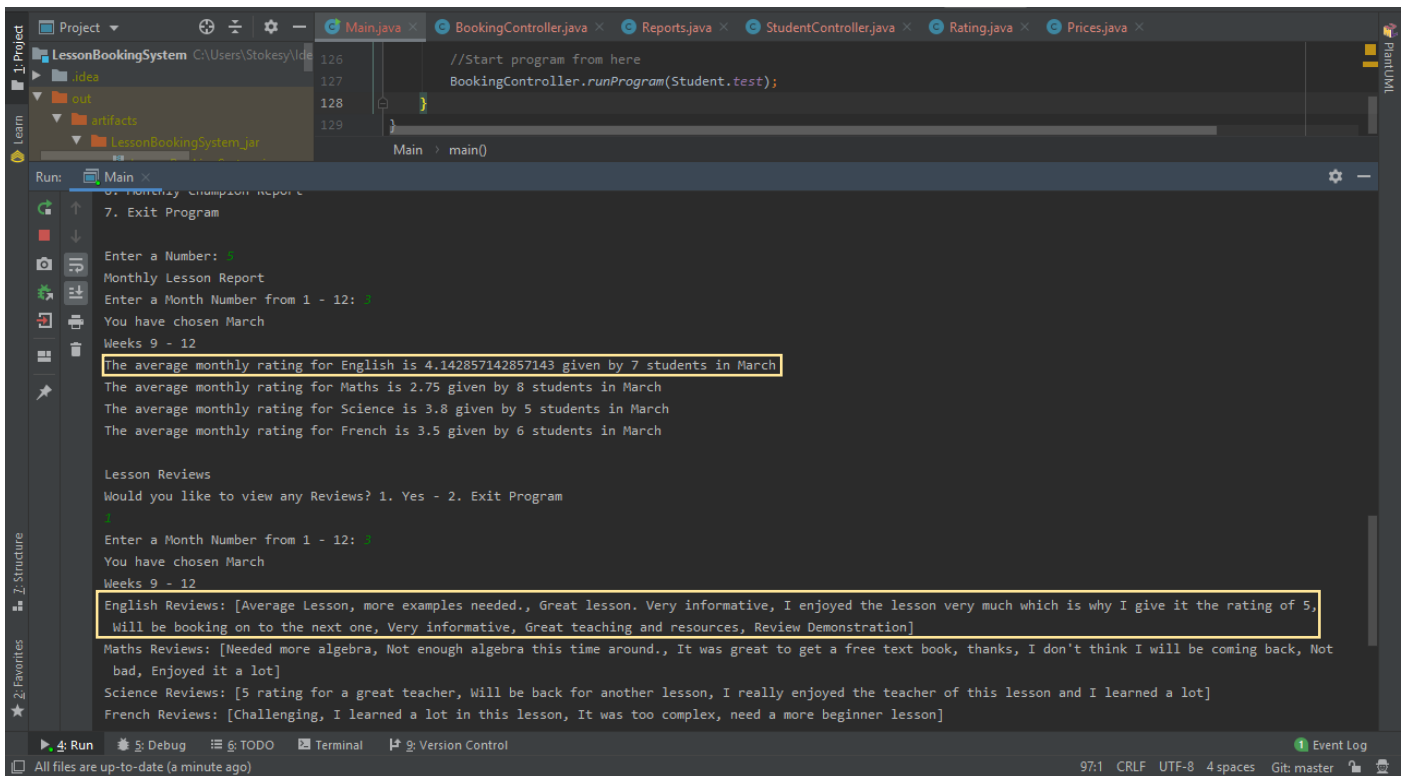
Run: Main
Enter a Number: 5
Monthly Lesson Report
Enter a Month Number from 1 - 12: 3
You have chosen March
Weeks 9 - 12
The average monthly rating for English is 4.0 given by 6 students in March
The average monthly rating for Maths is 2.75 given by 8 students in March
The average monthly rating for Science is 3.8 given by 5 students in March
The average monthly rating for French is 3.5 given by 6 students in March

Lesson Reviews
Would you like to view any Reviews? 1. Yes - 2. Exit Program
1
Enter a Month Number from 1 - 12: 3
You have chosen March
Weeks 9 - 12
English Reviews: [Average Lesson, more examples needed., Great lesson. Very informative, I enjoyed the lesson very much which is why I give it the rating of 5,
Will be booking on to the next one, Very informative, Great teaching and resources]
Maths Reviews: [Needed more algebra, Not enough algebra this time around., It was great to get a free text book, thanks, I don't think I will be coming back, Not
bad, Enjoyed it a lot]
Science Reviews: [5 rating for a great teacher, Will be back for another lesson, I really enjoyed the teacher of this lesson and I learned a lot]
French Reviews: [Challenging, I learned a lot in this lesson, It was too complex, need a more beginner lesson]
```

The user can now continue to use the program and attend the English Lesson for Saturday and leave a rating and a review with the user input:

- 4, 1, 1, 1, (Leave a Rating), 1, (Enter a String)

The user can then print the Monthly Lesson Report for March to see the updated review and rating in the report results for March.



As you can see, the rating average has gone up after leaving a rating of 5 and the last review for English now reads “Review Demonstration”.

Monthly Champion Lesson Report

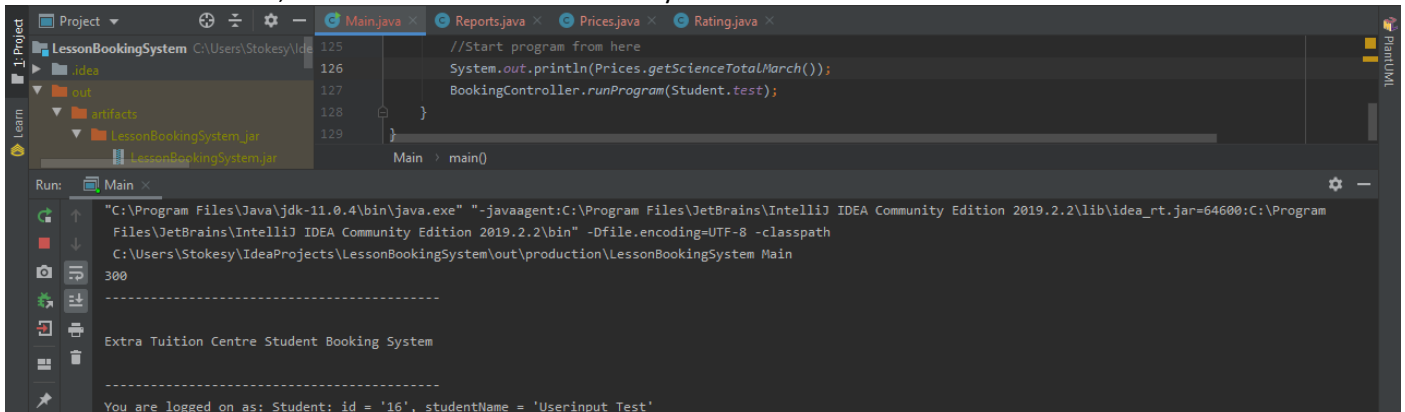
By running the code below in the main method, this will demonstrate the monthly income being displayed in the output reports. It will show the system updating the total income after a student books a lesson using the system.

```

System.out.println(Prices.getScienceTotalMarch());
BookingController.runProgram(Student.test);

```

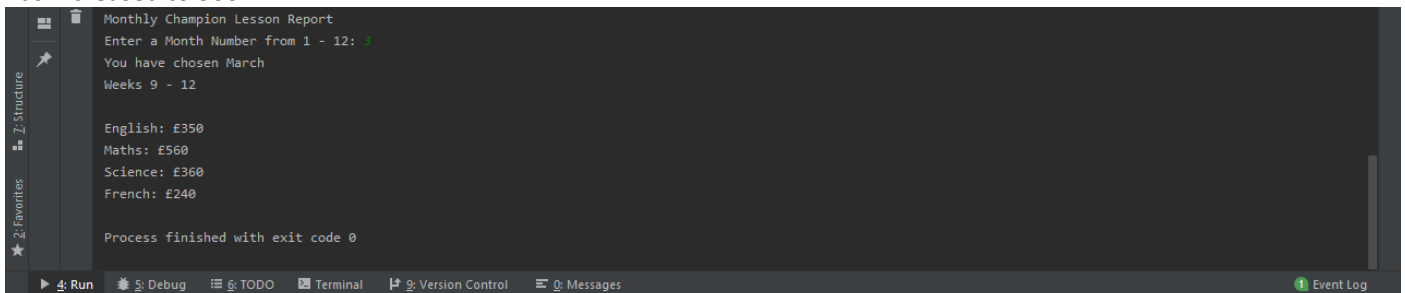
In the screenshot below, the total for science before the system has run is 300.



A user can book the Science lesson and print the updated champion report by using the following input:

- 1, 1, 3, 1, 6, 3

The system will then print the Champion report for March. Notice how the total income for the Science Lessons in March has increased to 360.



Exit Program

This simply stops the program from running.

Conclusion

It was an exciting project to work on, and I learned a lot from designing this project. Seeing the whole process and completing the project from the very beginning to the end was a valuable insight which I believe cannot be taught from books etc. I was able to complete the proposed functionality in the brief and the project works as per the specification. Reflecting on the project and thinking about room for improvement, I believe the code within the project could be tidied up a little more and made clearer from an outside perspective. Going back over the code, I believe there are other ways where I could have implemented code to make it faster and more efficient – these are something I will look at and explore in more detail. Also, I believe the class structure could be improved also. This is something I have identified and would like to work on for future developments. This project will become a basis for my programming development, and I will continue to work on it, improve it and apply new techniques learned as I develop my skills further in Java.