BOOKING SYSTEM

Contents

[***ASSUMPTIONS*** 2](#_heading=h.gjdgxs)

[***SYSTEM STRUCTURE*** 2](#_heading=h.30j0zll)

[***REFACTORING*** 4](#_heading=h.1fob9te)

[***DESIGN PATTERN*** 5](#_heading=h.2et92p0)

[Factory Method Pattern: 5](#_heading=h.tyjcwt)

[Abstract Report Class: 5](#_heading=h.3dy6vkm)

[ReportFactory Class: 5](#_heading=h.1t3h5sf)

[Creating Report Instances: 5](#_heading=h.4d34og8)

[***LearnerReport and CoachReport Classes*** 6](#_heading=h.2s8eyo1)

[LearnerReport Class: 6](#_heading=h.17dp8vu)

[CoachReport Class: 7](#_heading=h.3rdcrjn)

[***TEST CASES (JUNIT TESTING)*** 8](#_heading=h.26in1rg)

[TEST CASE- 1 8](#_heading=h.lnxbz9)

[OUTPUT 9](#_heading=h.35nkun2)

[TEST CASE- 2 9](#_heading=h.1ksv4uv)

[OUTPUT 10](#_heading=h.44sinio)

[TEST CASE – 3 10](#_heading=h.2jxsxqh)

[OUTPUT 11](#_heading=h.z337ya)

[TEST CASE - 4 11](#_heading=h.3j2qqm3)

[OUTPUT 12](#_heading=h.1y810tw)

[TEST CASE- 5 12](#_heading=h.4i7ojhp)

[OUTPUT 13](#_heading=h.2xcytpi)

[TEST CASE- 6 13](#_heading=h.1ci93xb)

# 

# ***ASSUMPTIONS***

* The system consists of two types of users: Admin and Learner.
* Both learners and admin are required to log in to the system to access its functionality.
* The creation of a new Learner is initiated by the admin. Upon creating a new learner, the admin provides the learner with a username and password to log in to the system. Subsequently, learners can access features such as checking timetables and booking classes.

# ***SYSTEM STRUCTURE***

The class diagram that is shown below represents the system structure. This class diagram shows several classes that together make up the entire system.

Classes of the system linked to this main class are listed below-

1. User: Denotes a person who uses the system to communicate, like coaches or learners. Depending on their jobs, they can register, log in, and access different functions.

2. Coach: Serves as the system's equivalent of a qualified instructor or trainer. With CoachReports, coaches are able to oversee the creation of lesson plans, monitor the progress of their students, and offer comments.

3. Admin: The person in charge of overseeing the platform as a whole, including user accounts, preferences, and upkeep of the system.

4. Lesson: In the learning platform, this symbol denotes a particular lesson or session. It might contain information on the lesson's topic, length, and required resources.

5. Report: This is an all-purpose report that the system generates and can contain a variety of data, including learner performance and system utilization statistics.

6. CoachReport: A report created expressly by a coach to offer criticism on a student's performance in a class.

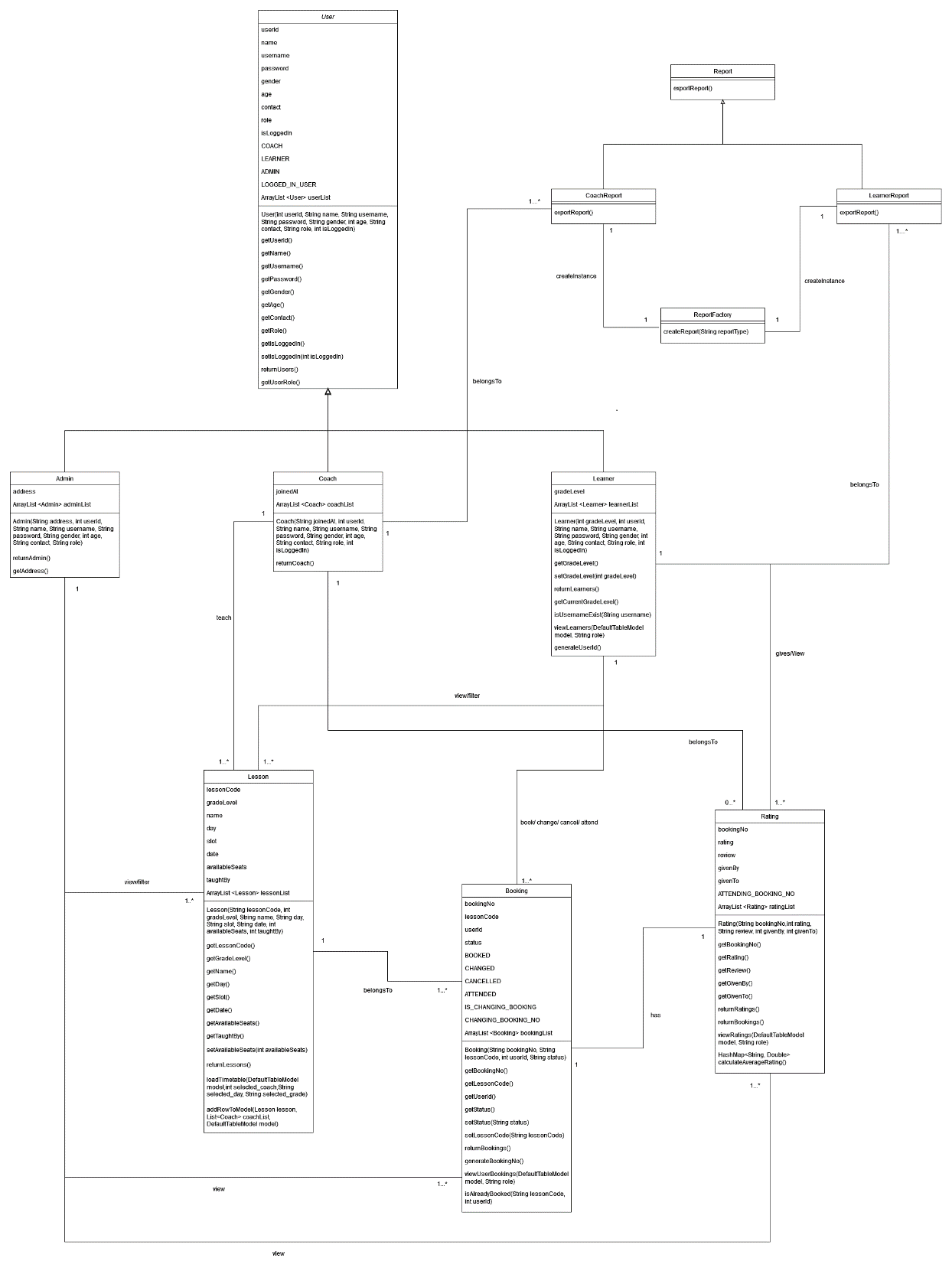
7. ReportFactory: This class generates several report formats according to user specifications and system information.

8. LearnerReport: A report prepared for a student that may contain data on their performance, progress, and other pertinent details to assist them in tracking their educational path.

9. Learner: Indicates a person who is utilizing the system to further their education. They can follow their progress with LearnerReports, attend lessons, and get coaching input.

10. Booking: Denotes a learner's reservation for a particular class with a coach.

11. Ratings: Represents a feature within the system that allows users to rate lessons, coaches, or other aspects of the platform to provide feedback for improvement.



(***CLASS DIAGRAM***)

# ***REFACTORING***

Refactoring is a crucial process in software development that involves restructuring existing code to improve its readability, maintainability, and efficiency without altering its external behavior.

1. Code Extraction into a Common Method:

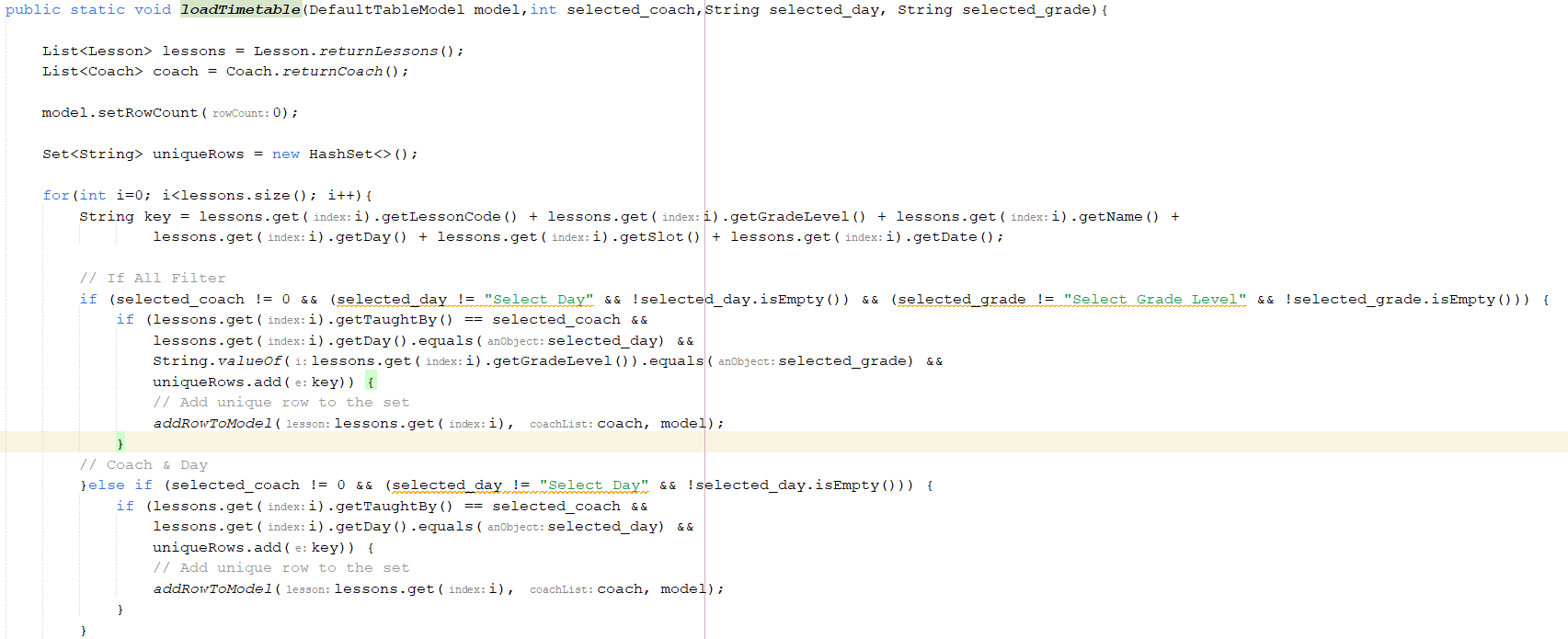
* The refactoring process involved extracting common code related to filtering timetable lists into a single method. This approach promotes code reusability and reduces redundancy by consolidating similar functionalities.
* By creating a common method for displaying timetables with and without filters, the codebase becomes more organized and easier to maintain. This consolidation simplifies the implementation of filtering logic and enhances the overall structure of the code.

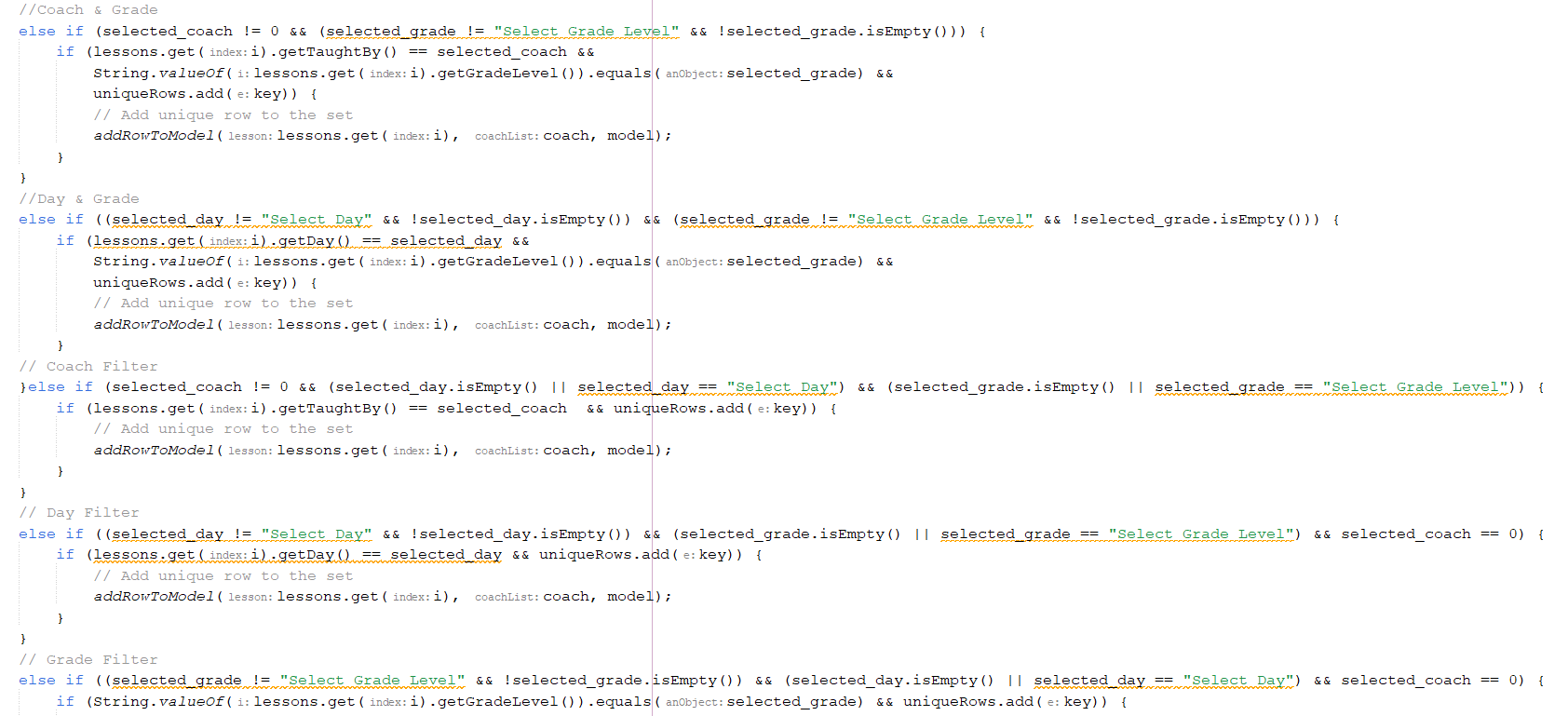
2. Parameterized Filtering:

* The refactored code now utilizes parameters to determine the filtering criteria for the timetable display. When a user selects a filter (such as coach name, grade level, or day), the corresponding parameter is passed to the common method to apply the filter.
* This parameterized approach enhances flexibility and scalability, as the method can adapt to different filtering requirements based on the provided parameters. It streamlines the filtering process and accommodates variations in filtering criteria without the need for extensive code modifications.

3. Reduction in Code Complexity:

* By refactoring to use a common method with parameterized filtering, the codebase benefits from reduced complexity and improved readability. The logic for filtering timetables based on different parameters is centralized, making it easier to understand and maintain.





# ***DESIGN PATTERN***

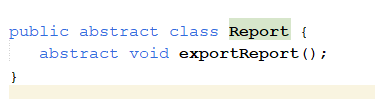
## Factory Method Pattern:

The ***Factory Method Pattern*** is a creational design pattern that provides an interface for creating instances of a class Fortuin (2010b) . It allows subclasses to alter the type of objects that will be created, promoting flexibility and extensibility in object creation.

The process of implementing this pattern in this system are given below

## Abstract Report Class:

An abstract `Report` class is created with the declaration of an abstract method. This abstract class serves as a blueprint for specific types of reports that will be generated in the system.



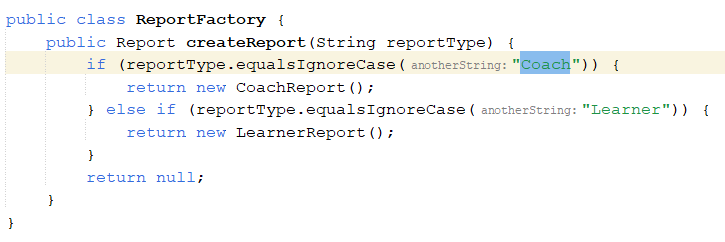
## ReportFactory Class:

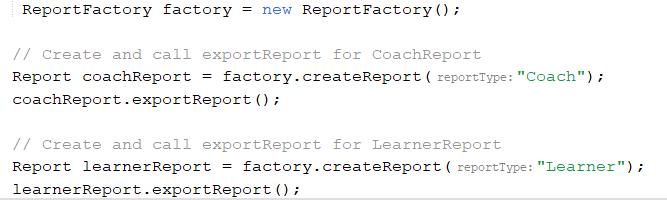
A `ReportFactory` class is implemented to handle the creation of different types of reports. This factory class contains a method, `createReport()`, which takes a `reportType` parameter and returns an instance of a specific report based on the provided type.

## Creating Report Instances:

When `createReport()` is called with a specific `reportType`, the factory method determines which type of report to instantiate:

* If the `reportType` is `Coach`, an instance of the `CoachReport` class is created.
* If the `reportType` is `Learner`, an instance of the `LearnerReport` class is created.

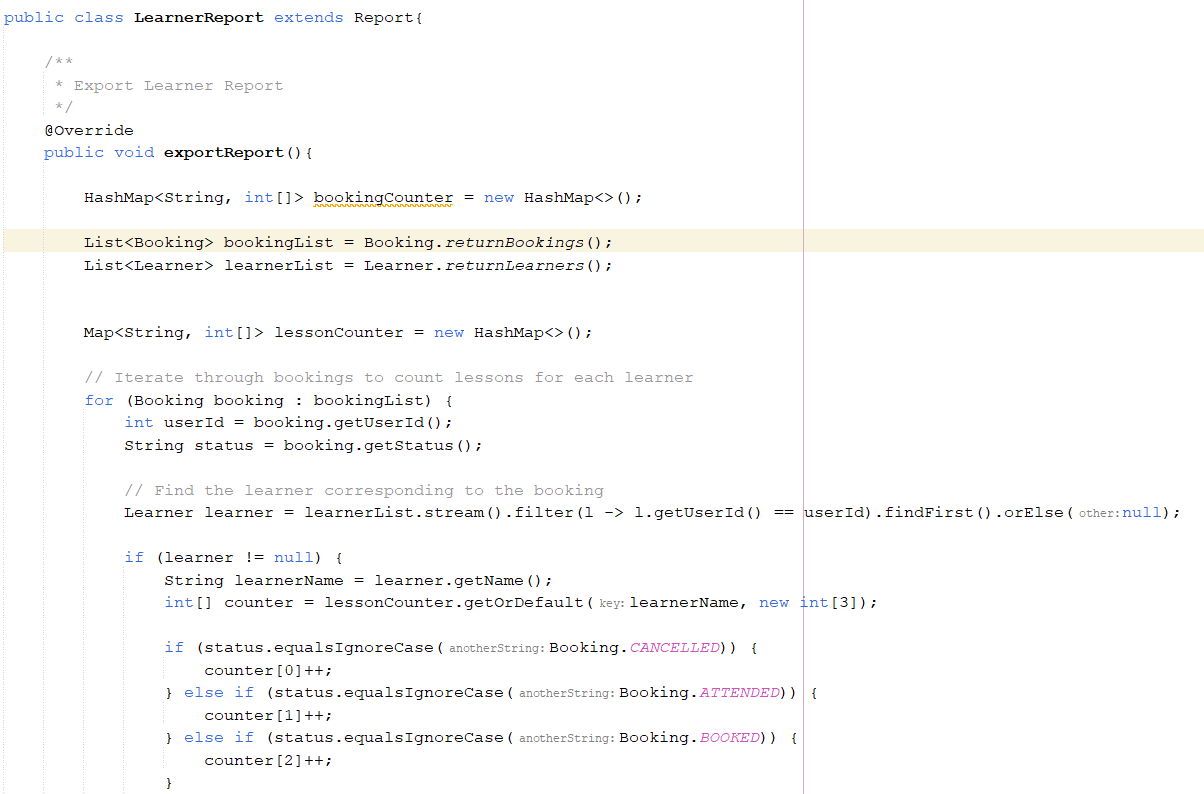


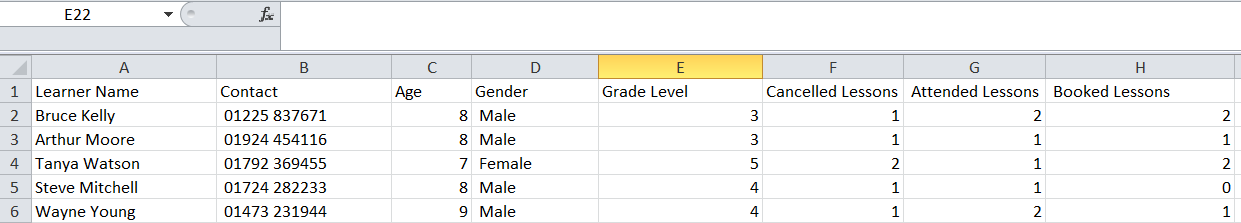


# ***LearnerReport and CoachReport Classes***

## LearnerReport Class:

Contains a method that downloads a CSV file displaying a report for each learner. This report includes information on total booked, attended, and canceled lessons for each learner.

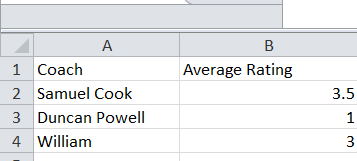




## CoachReport Class:

Includes a method that downloads a CSV file showing a report for each coach. This report presents the average rating received from learners while attending lessons with the coach.





# ***TEST CASES (JUNIT TESTING)***

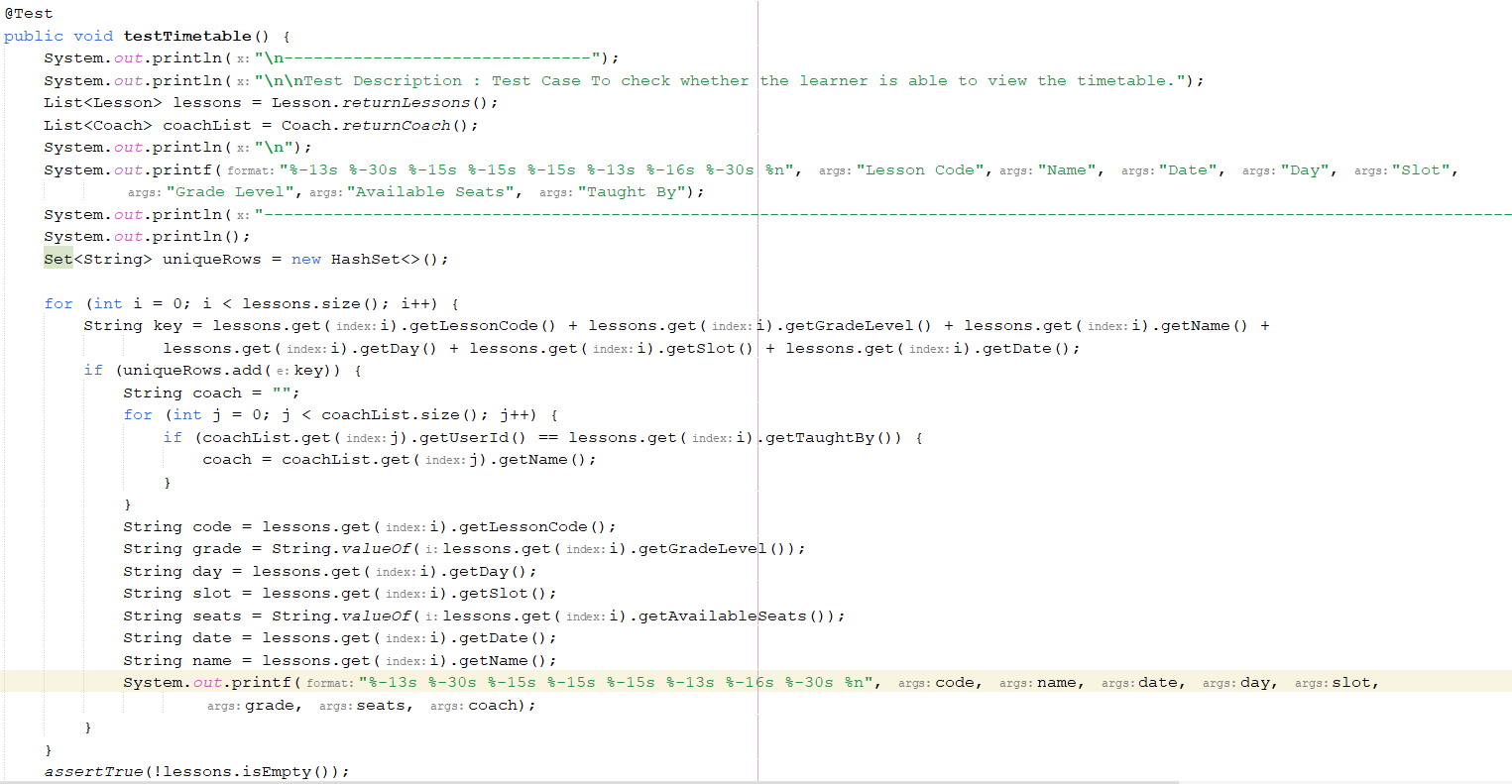
### TEST CASE- 1

Test Description**:** To check whether the learner is able to view the timetable.

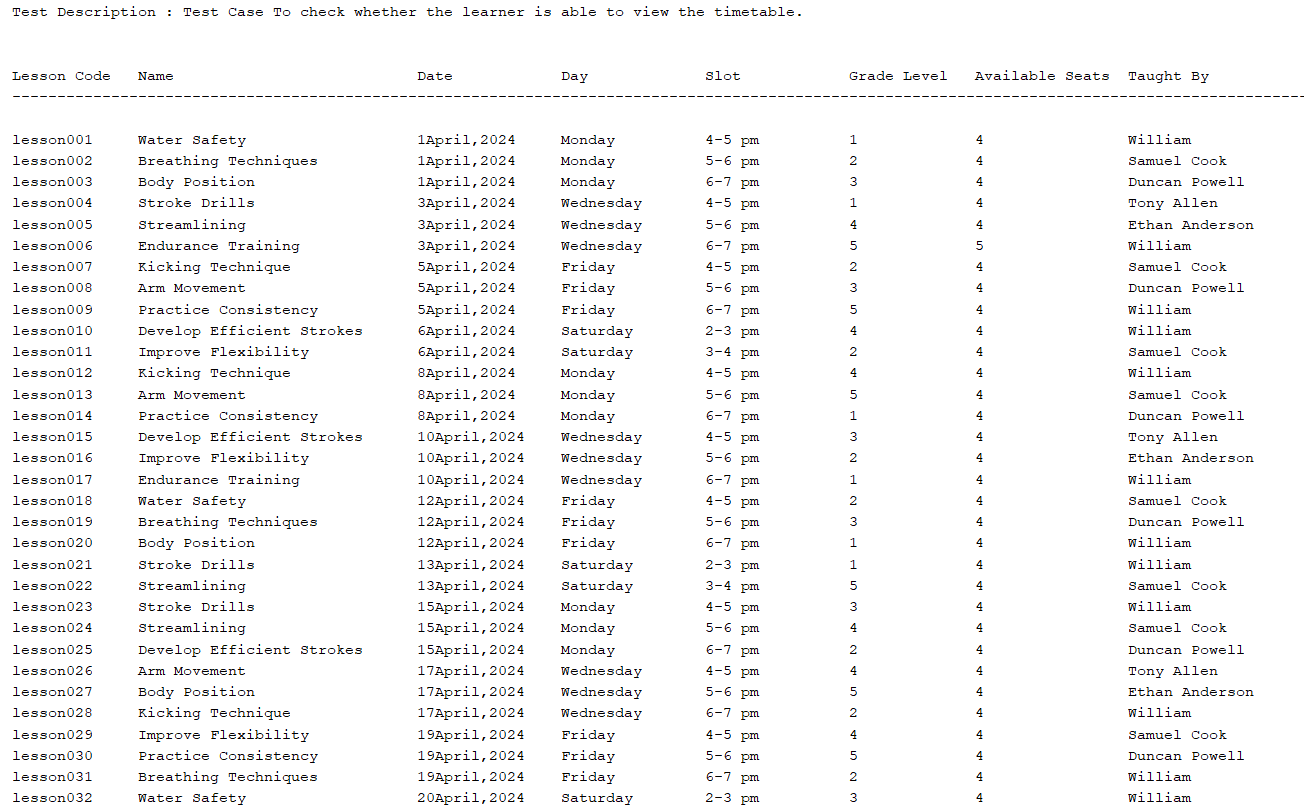
Test Data**:** N/A

Expected Output**:** The timetable with all lessons should be displayed.

Actual Output**:** The timetable with all lessons is displayed.



### OUTPUT



### TEST CASE- 2

Test Case: To check whether the learner is able to filter the timetable by a given day.

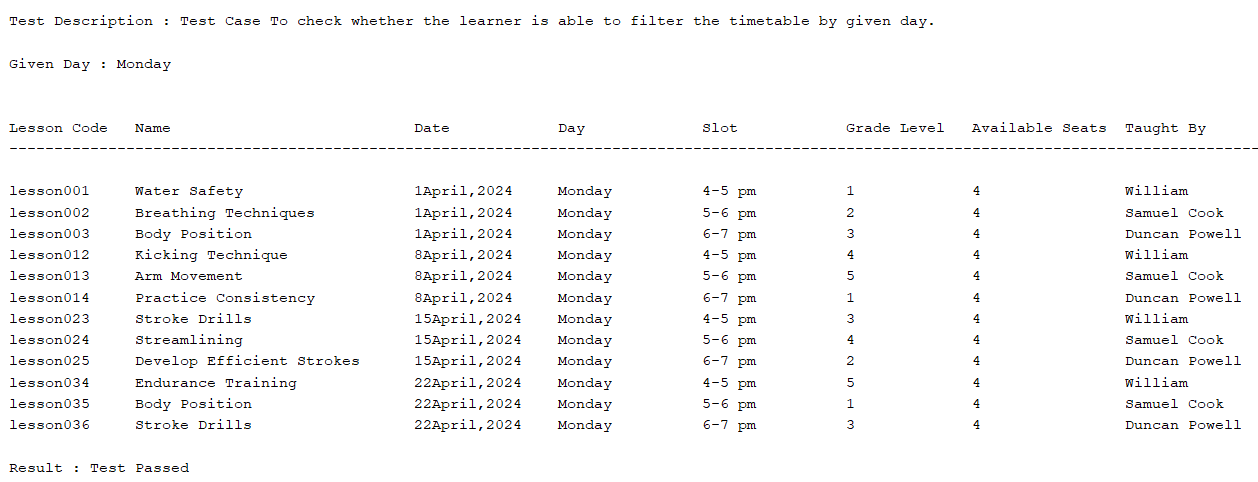
Test Data: day

Expected Output: The timetable with the given day should be displayed.

Actual Output: The timetable with the given day is displayed.



### OUTPUT

****

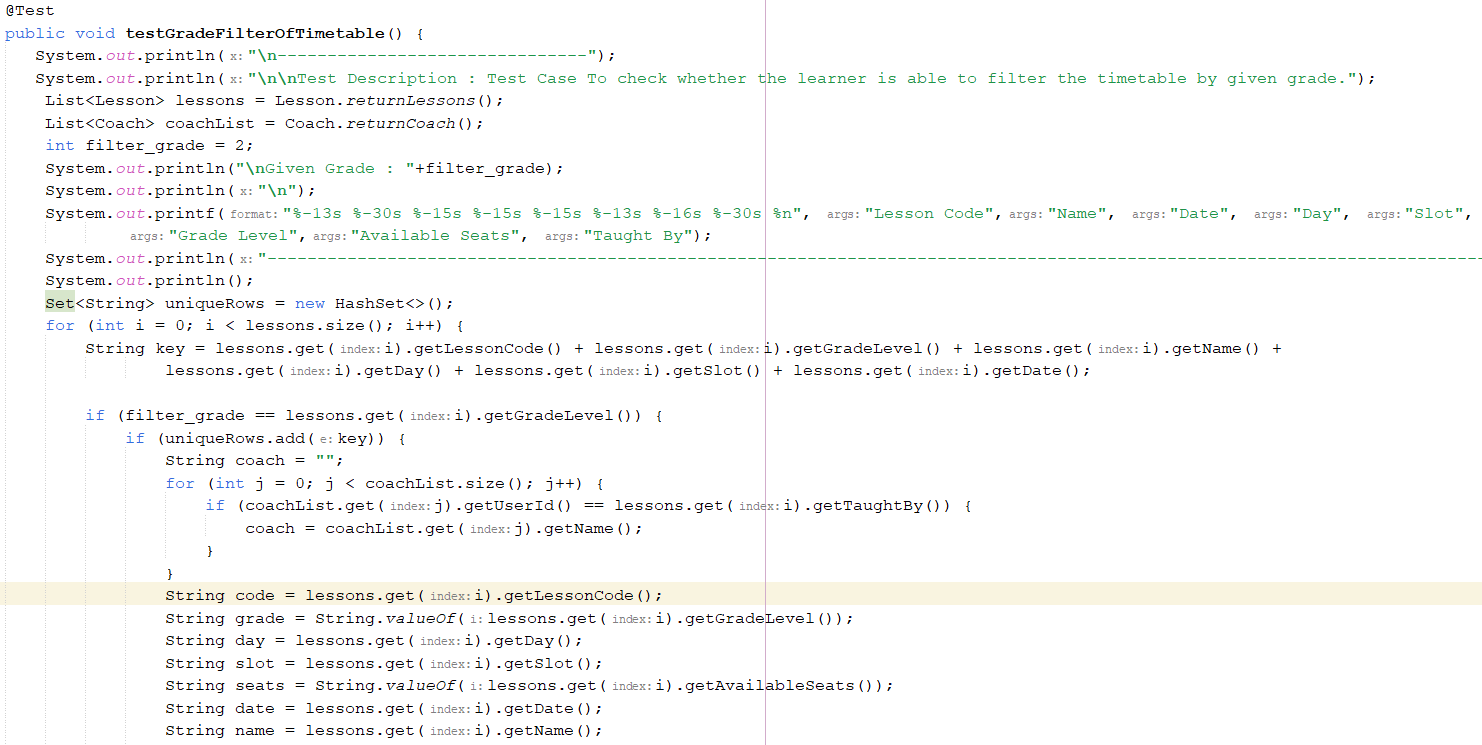
### TEST CASE – 3

Test Case: To check whether the learner is able to filter the timetable by given grade level.

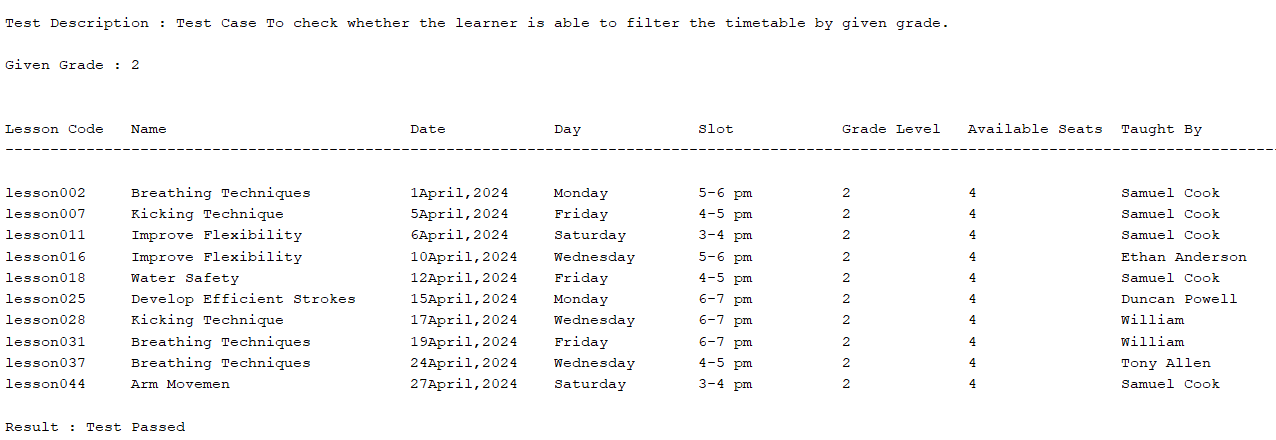
Test Data: Grade Level

Expected Output: The timetable with the given grade level should be displayed.

Actual Output: The timetable with the given grade level is displayed.

****

### OUTPUT

****

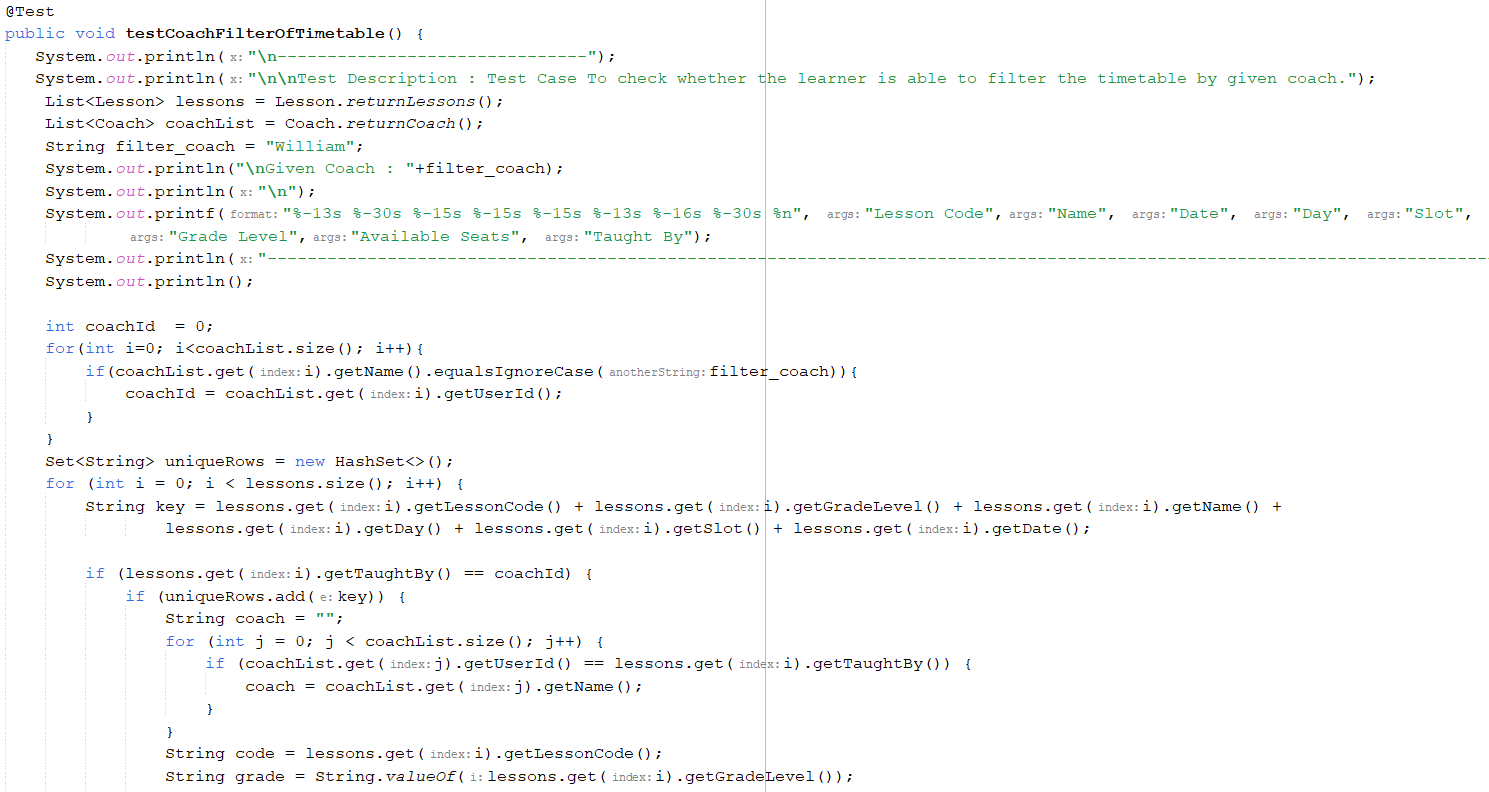
### TEST CASE - 4

Test Case: To check whether the learner is able to filter the timetable by a given coach.

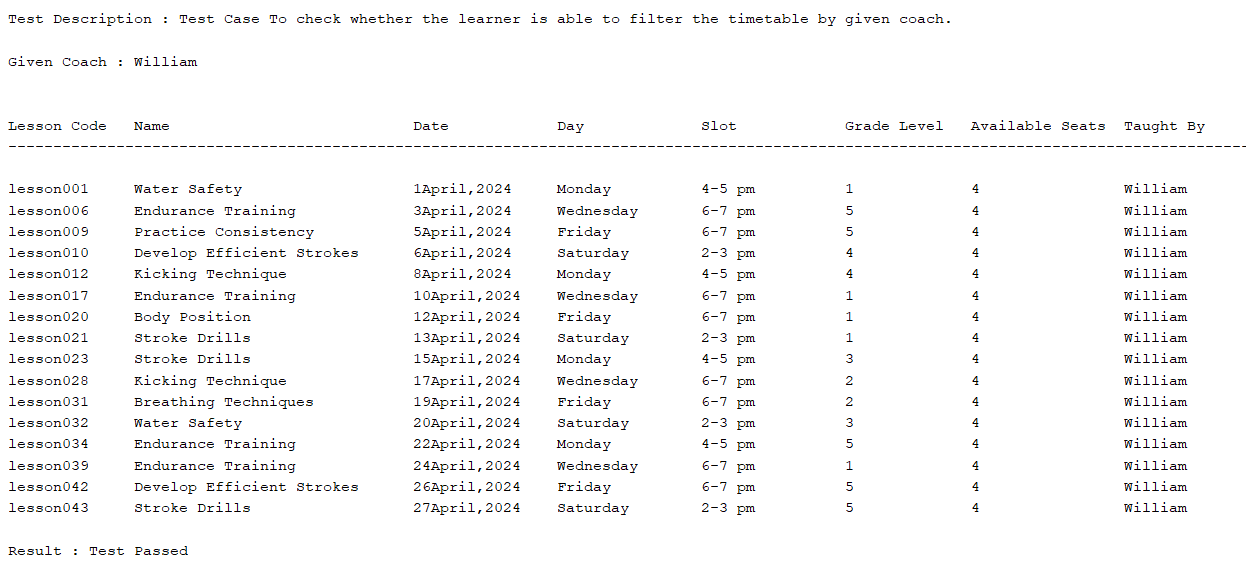
Test Data: Coach Name

Expected Output: The timetable with the given coach name should be displayed.

Actual Output: The timetable with the given coach name is displayed.

****

### OUTPUT

****

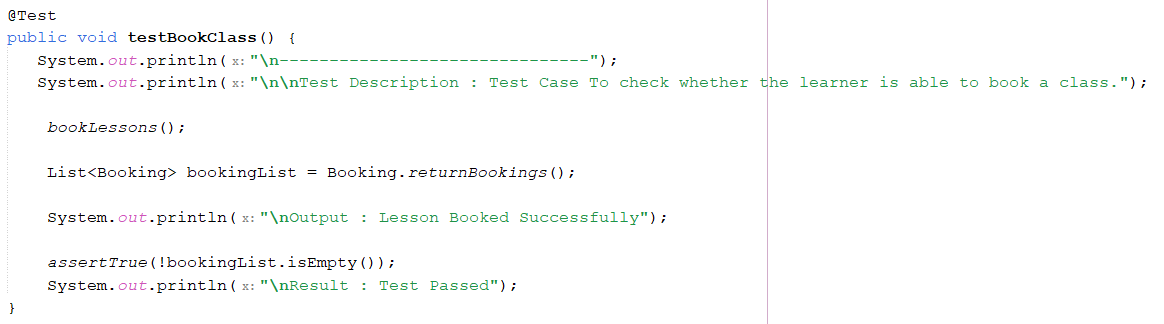
### TEST CASE- 5

Test Case: To check whether the learner is able to book a class.

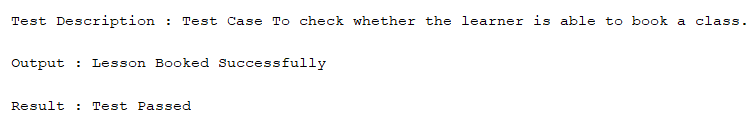
Test Data: LessonCode, UserID

Expected Output: The learner should be able to book a lesson.

Actual Output: The learner is able to book a lesson. Also, the seat availability for the selected lessonCode is decreased by 1.

****

### OUTPUT

****

### TEST CASE- 6

Test Case: To check whether the learner is able to change a booking with the new lesson and the old lesson which was booked earlier, the seat availability will be increased for that lesson by 1.

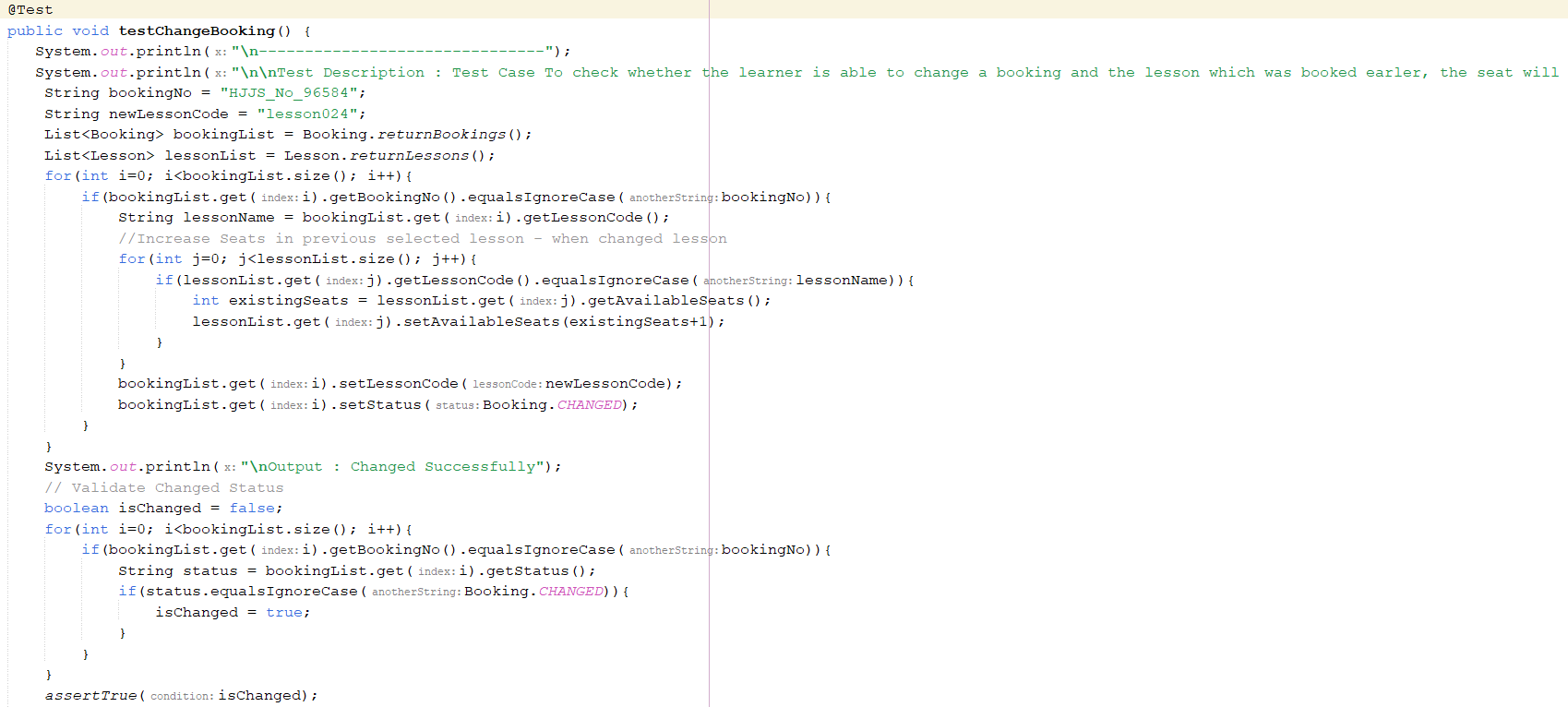
Test Data: BookingNo, newLessonCode

Expected Output:

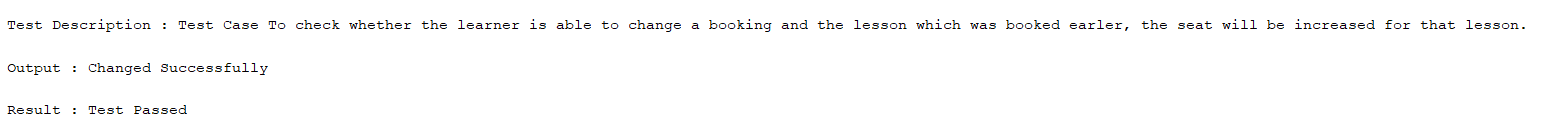
* The learner should be able to change a given booking number.
* The booking status should be changed to “Changed”.
* The seat availability for the new lessonCode should be decreased by 1.
* The seat availability for the old lessonCode should be increased by 1 so that it can be available to other learners to book.

Actual Output:

* The learner is able to change a given booking number, and the booking status is changed to “Changed”.
* The seat availability for the new lessonCode is decreased by 1.
* The seat availability for the old lessonCode is increased by 1 so that it can be available to other learners to book.

****

### OUTPUT

****