Programming for Software Engineers

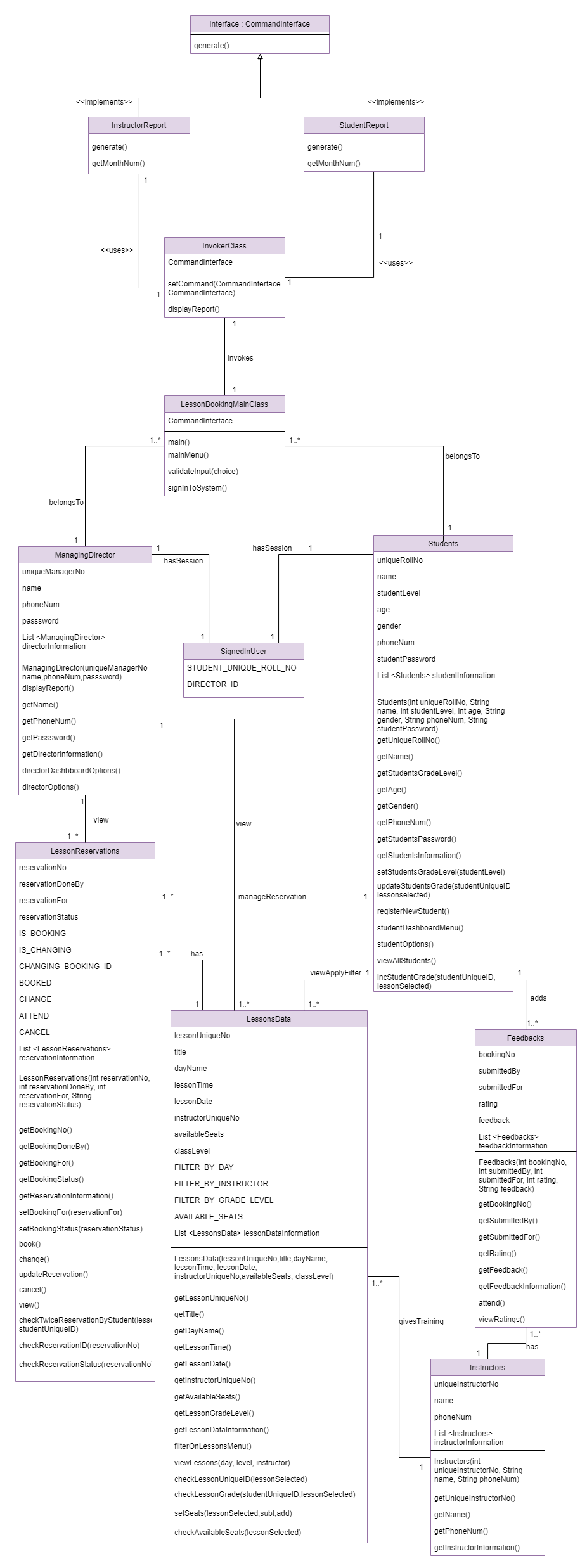
# INTRODUCTION

The assignment involves the development of a software system for the Hatfield Junior Swimming School (HJSS) to manage bookings made by learners. The system is designed to accommodate various functionalities, including creating an account, viewing timetables, booking lessons, and generating reports. The primary objective is to create a user-friendly platform that allows learners to book swimming lessons based on their grade level, coach's name, or specific day. The system also needs to store learner information, manage reservations, and record feedback and ratings for each lesson. Also, the software must be capable of generating detailed reports for both learners and instructors. The problem statement revolves around the need for an efficient and organized system to update the booking and management of swimming lessons at HJSS. The software system is required to handle multiple aspects, such as lesson scheduling, learner information management, and feedback collection, while providing a seamless user experience for both learners and instructors.

# SYSTEM DESIGN

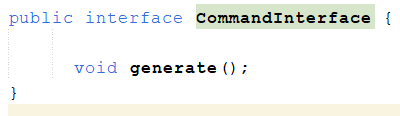
The overall system design approach for the Hatfield Junior Swimming School (HJSS) software involves creating a system that allows learners to book swimming lessons, view timetables, and receive feedback. The main classes in the system include Student, Instructor, Lesson, Reservation, and Feedback. The Student class holds attributes such as name, gender, age, emergency contact phone number, and current grade level. The Instructor class contains information about the coaches, including their names and average ratings received. The Lesson class includes details about the grade level, time slots, and availability. The Reservation class manages the booking of lessons for students, while the Feedback class records the reviews and ratings provided by learners after attending a lesson.

The **class diagram** highlights the links between main classes, showing how students can reserve lessons, attend them, and provide feedback (Visual Paradigm, 2019). The diagram also explains the relationship between instructors and the lessons they teach, as well as the association between lessons and the timetable.



In **design patterns**, the system develops the Command Design Pattern to generate reports according to user commands. This pattern involves creating an invoker class to receive commands from the user, such as generating a Student Report or an Instructor Report (McDonough, 2017). In addition, the system follows design principles such as reusability and readability by creating different methods and avoiding code repetition.

An interface is created 🡪

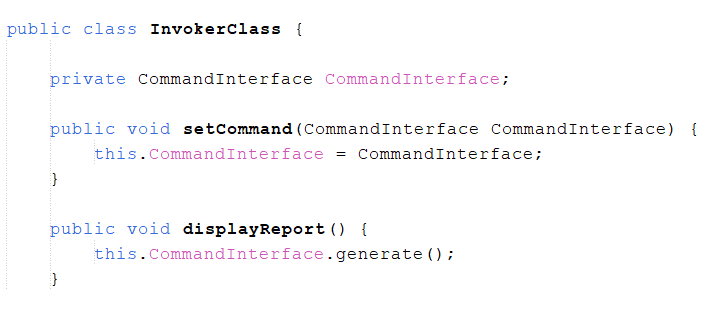
****

Concrete class are created to implement above and override its method for different functionalities🡪

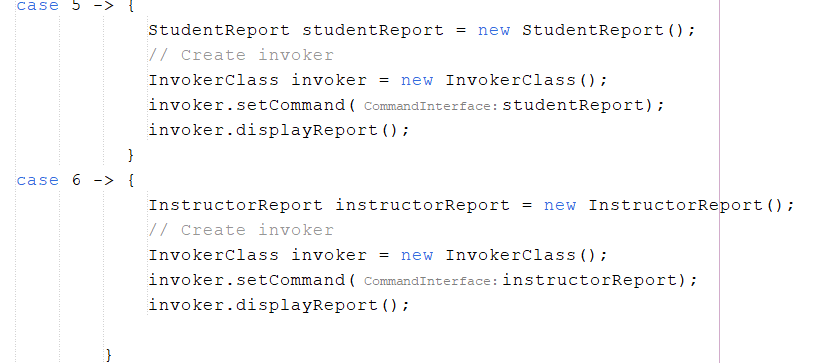
****

****

Invokes Class 🡪

****

Usage 🡪

****

The code has undergone a process called **refactoring**, which aims to improve its overall quality and maintainability (Almogahed and Omar, 2021). In this specific case, two major improvements have been implemented. Firstly, the Command Design pattern has been incorporated to facilitate the generation of reports based on user input. Instead of directly creating objects of various classes, an invoker class is now utilized to receive and process commands for either Student Report or Instructor Report. Consequently, this enhances the code's flexibility and adaptability to different user requirements.

Secondly, the codebase has been optimized by creating reusable methods and calling them as needed, rather than duplicating the same code across multiple methods. This approach not only saves time and effort but also significantly enhances the code's readability and comprehension for developers working on the project. By adhering to these refactoring principles, the overall quality of the code has been elevated, making it more efficient, maintainable, and easier to understand.

# IMPLEMENTATION DETAILS

The programming language used for implementation is Java, and the tools include Eclipse IDE for coding and JUnit for testing.

Key functionalities and algorithms implemented in the system🡪

* Managing bookings for swimming lessons at different grade levels and time slots.
* Viewing the timetable by day, grade level, or coach's name.
* Allowing learners to book, change, or cancel lessons within certain constraints.
* Recording and displaying feedback and ratings for each lesson and instructor.
* Generating reports for detailed information on learners, reservations, and feedback.

Challenges faced during implementation🡪

* Designing a user-friendly interface for learners to view and book lessons.
* Implementing the Command Design Pattern for generating reports based on user commands.
* Ensuring that the system accurately updates learner grade levels after attending lessons.

To address these challenges, focused on creating a clear and intuitive user interface, carefully designed the structure of the Command Design Pattern, and thoroughly tested the grade level update functionality.

The code utilizes the Command Design pattern to generate reports based on user input. Rather than creating separate objects for each class, an invoker class is employed to receive commands for either Student Report or Instructor Report. This approach allows for the creation of the corresponding class object on demand. Additionally, the code streamlines its structure by implementing reusable methods, reducing the need for repetitive code throughout the program. Consequently, the code's readability and comprehension are significantly improved.

# TESTING AND VALIDATION

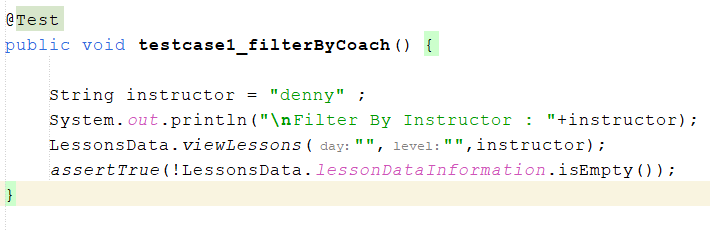
The approach to testing the system involves creating JUnit tests to validate the functionality of the software. The tests are designed to cover various scenarios and functionalities to ensure that the system operates as intended (Toure, Badri and Lamontagne, 2014).

The JUnit tests created🡪

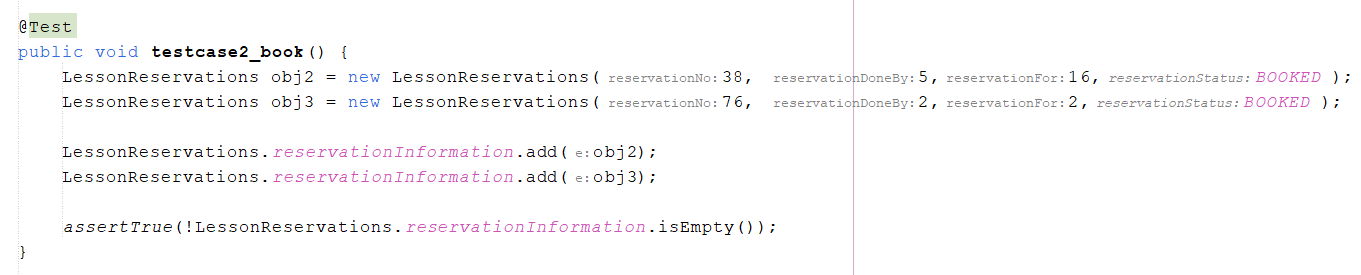
* Test Case to filter timetable by instructor name: This test validates that the timetable is displayed correctly for the selected instructor name.
* Test Case to reserve class: This test ensures that the selected lesson is successfully reserved for students.
* Test Case to change class: This test verifies that the reservation is changed with the new lesson.
* Test Case to cancel class: This test confirms that the reservation is successfully canceled.
* Test Case to attend class: This test validates that the reservation is attended, and feedback is submitted for the instructor.

The test results ensured the correctness of the implementation by comparing the expected output with the actual output. If the actual output matched the expected output, it indicated that the system was functioning as intended. Any discrepancies between the expected and actual outputs were investigated and addressed to ensure the accuracy and reliability of the system.

| **TestID** | **Test Case** | **Test Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| 1 | Test Case to filter timetable by instructor name. | instructor name | The timetable should be displayed of the selected instructor name. | The timetable is  displayed of the selected instructor name. |

****

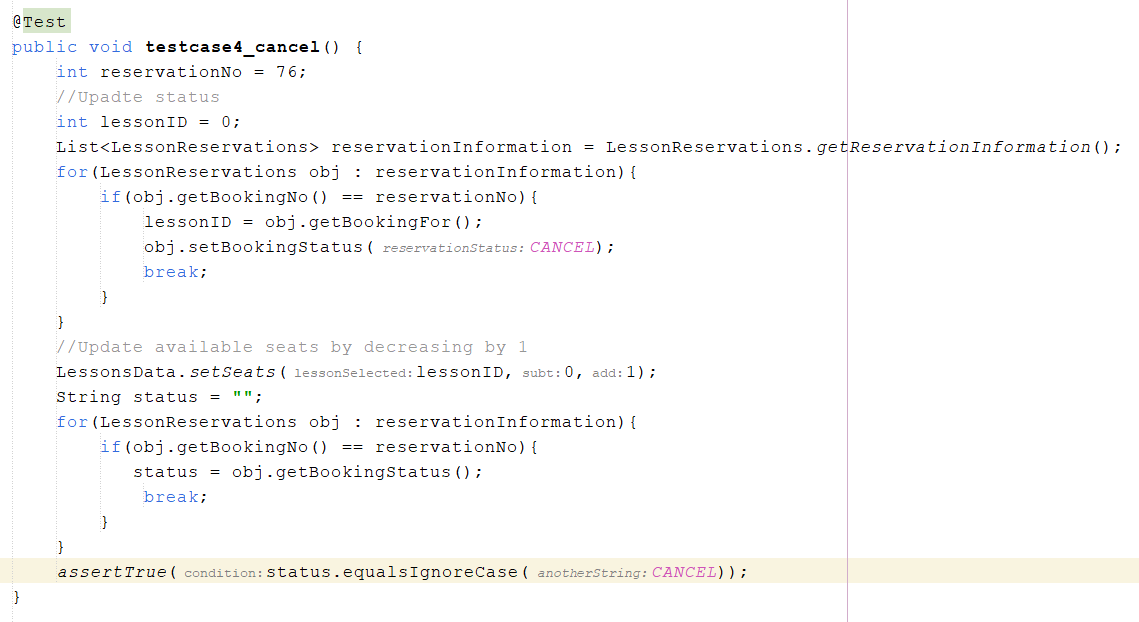
| **TestID** | **Test Case** | **Test Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| 2 | Test Case to reserve class | studentUniqueID, lessonUniqueID | The selected lesson should be reserved for students. | The selected lesson is  reserved for students. |

****

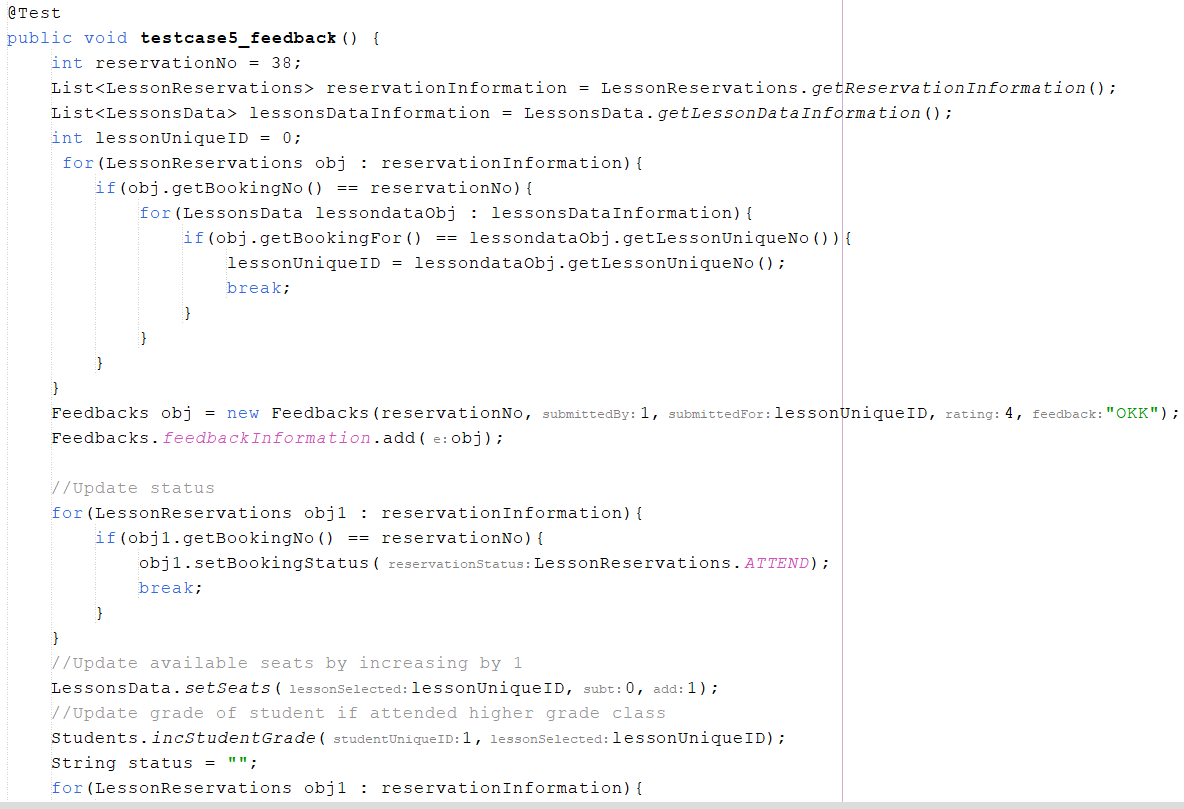
| **TestID** | **Test Case** | **Test Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| 3 | Test Case to change class | reservationNo, newLessonUniqueID | The reservation should be changed with the new lesson. | The reservation is changed with the new lesson. |

****

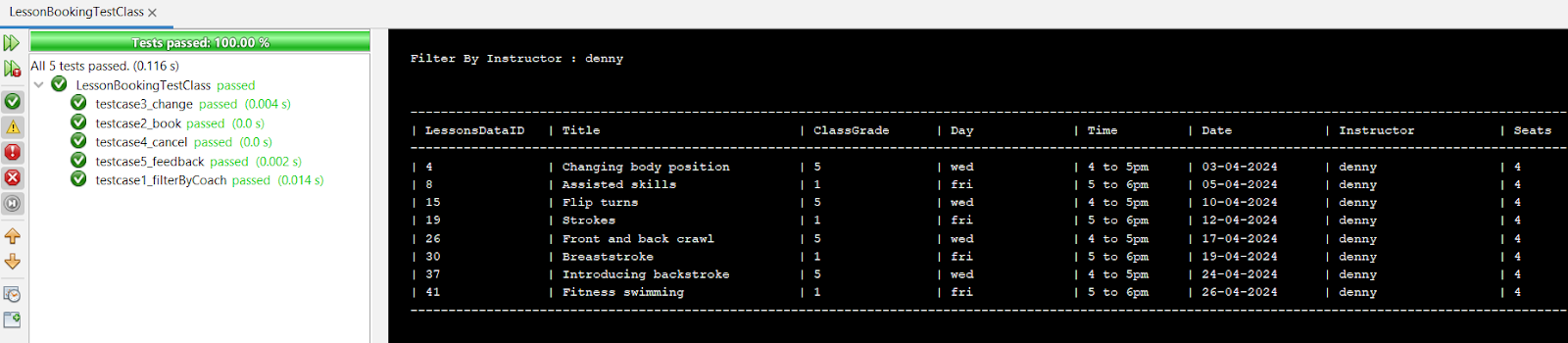
| **TestID** | **Test Case** | **Test Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| 4 | Test Case to cancel class | reservationNo | The reservation should be cancelled. | The reservation is cancelled. |

****

| **TestID** | **Test Case** | **Test Data** | **Expected Output** | **Actual Output** |
| --- | --- | --- | --- | --- |
| 5 | Test Case to attend class | reservationNo, rating, feedback for instructor | The reservation should be attended and feedback should be submitted. | The reservation is attended and feedback should be submitted. |

****

**OUTPUT 🡪**

****

# ASSUMPTIONS AND LIMITATIONS

Assumptions made during the development of the system include that the student will create an account with the system to access its functionality, and that the Managing Director will have access to view all students, reservations, feedback given to instructors, and check reports. As well as, The student can perform different functions by signing with the system.

Limitations and constraints🡪

* Learners are only allowed to book lessons at their current grade level or one level higher, and that they are not allowed to reserve a class higher or lower than their current grade level.
* The system has a constraint on the number of learners and coaches, with around 15 learners and 3-4 coaches being pre-registered. The system also has a limitation on the age of learners, which needs to be between 4 and 11 years old.

# REFLECTION AND LESSON LEARNED

# The report involves designing a software system for managing swimming lesson bookings at the Hatfield Junior Swimming School (HJSS). The system should allow learners to book, cancel, and attend lessons, as well as provide feedback and ratings for the instructors. The system should also generate reports for learners, reservations, and feedback.

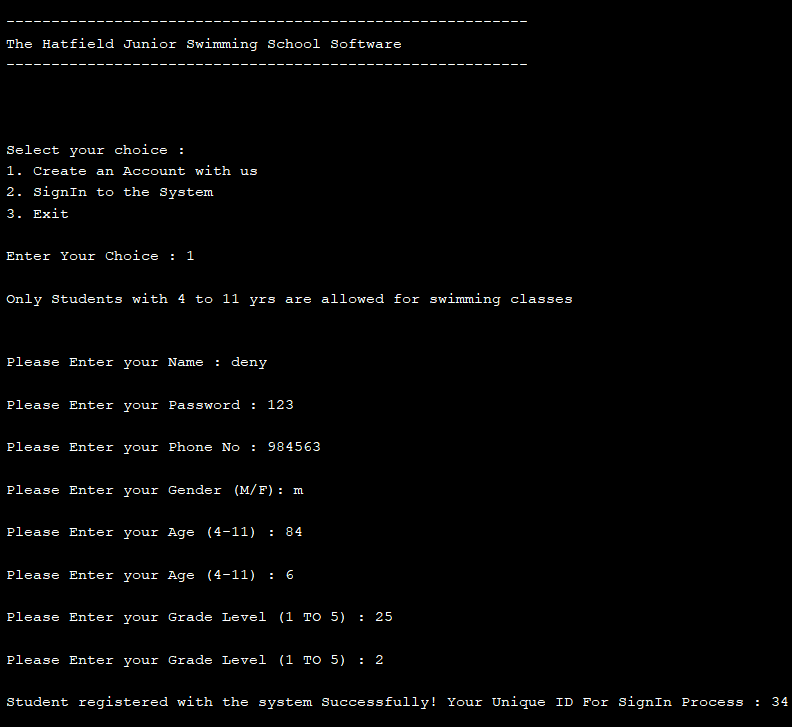
* In terms of technical lessons learned, the assignment required the implementation of the Command Design pattern to generate reports according to user commands. This involved creating an invoker class to receive commands and create corresponding objects. Also, the project involved creating concrete classes to implement interfaces for different functionalities, such as reserving, canceling, and attending lessons.
* From a project management perspective, the assignment highlighted the importance of refactoring code to improve readability and understanding. By implementing the Command Design pattern and creating different methods, the readability and understanding of the code were increased.
* If given the opportunity to revisit the project, one improvement would be to further enhance the user interface and user experience of the software system. This could involve creating a more intuitive and user-friendly interface for learners to view and book lessons, as well as for managing directors to view reports and feedback. In addition, conducting more extensive JUnit testing to cover a wider range of test cases would be beneficial to ensure the robustness and reliability of the system.

# CONCLUSION

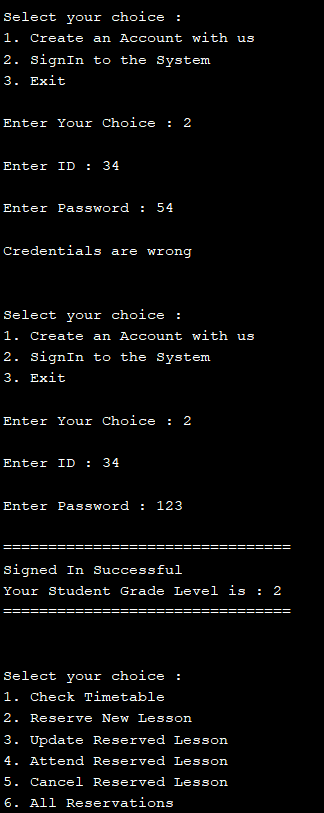
The report discusses the implementation of a software system for managing swimming lesson bookings at the Hatfield Junior Swimming School. Key points include the need for learners to create an account, the ability to view and book lessons based on day, grade level, or coach, and the requirement for learners to provide feedback and ratings after attending lessons. The system also includes the Command Design Pattern for generating reports and an interface for implementing different functionalities. The system aims to streamline the booking process, provide detailed reports on learner activities, and enhance the overall management of swimming lessons at the school. Overall, the system contributes to efficient lesson scheduling, learner engagement, and data-driven decision-making for the school.

# SCREENSHOTS

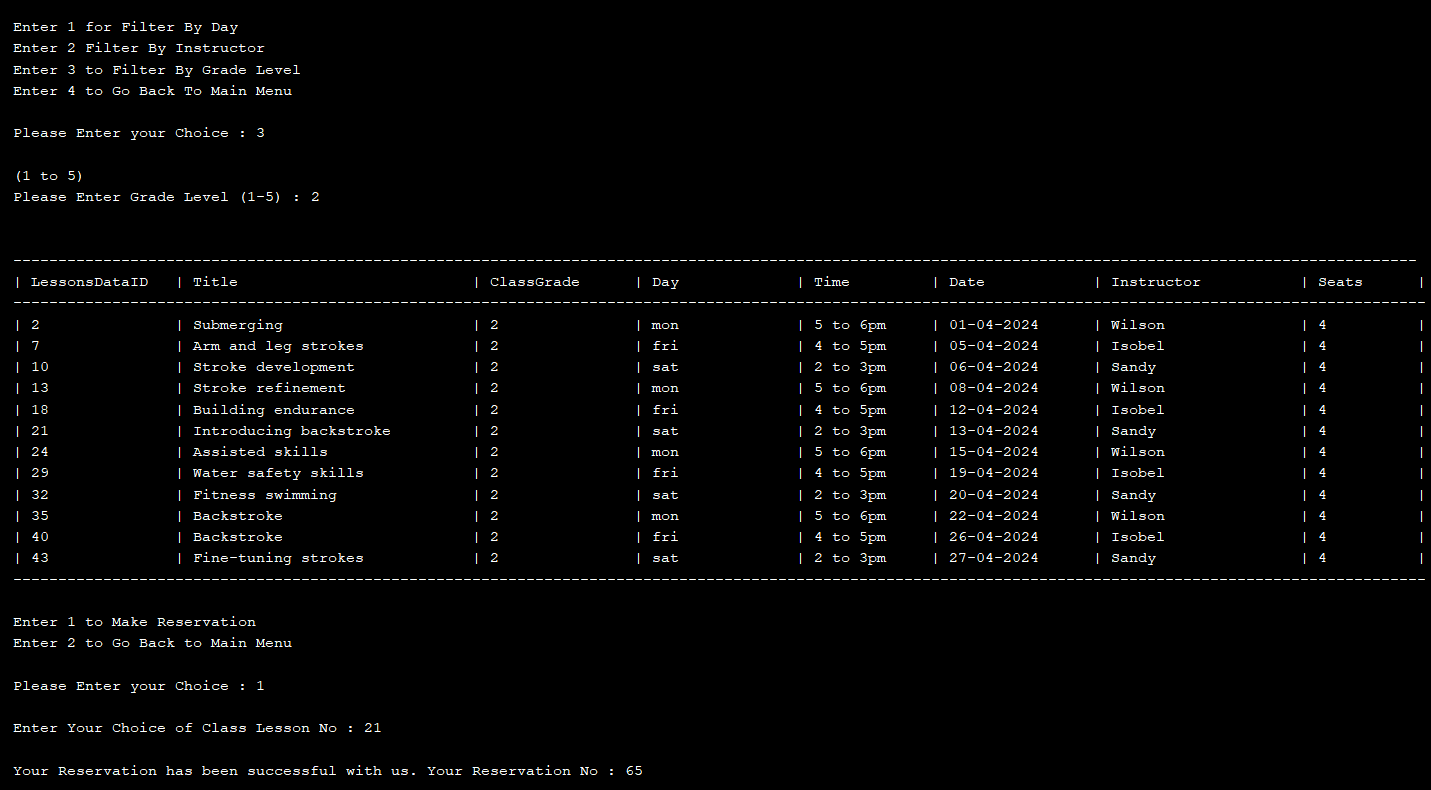
Register process 🡪



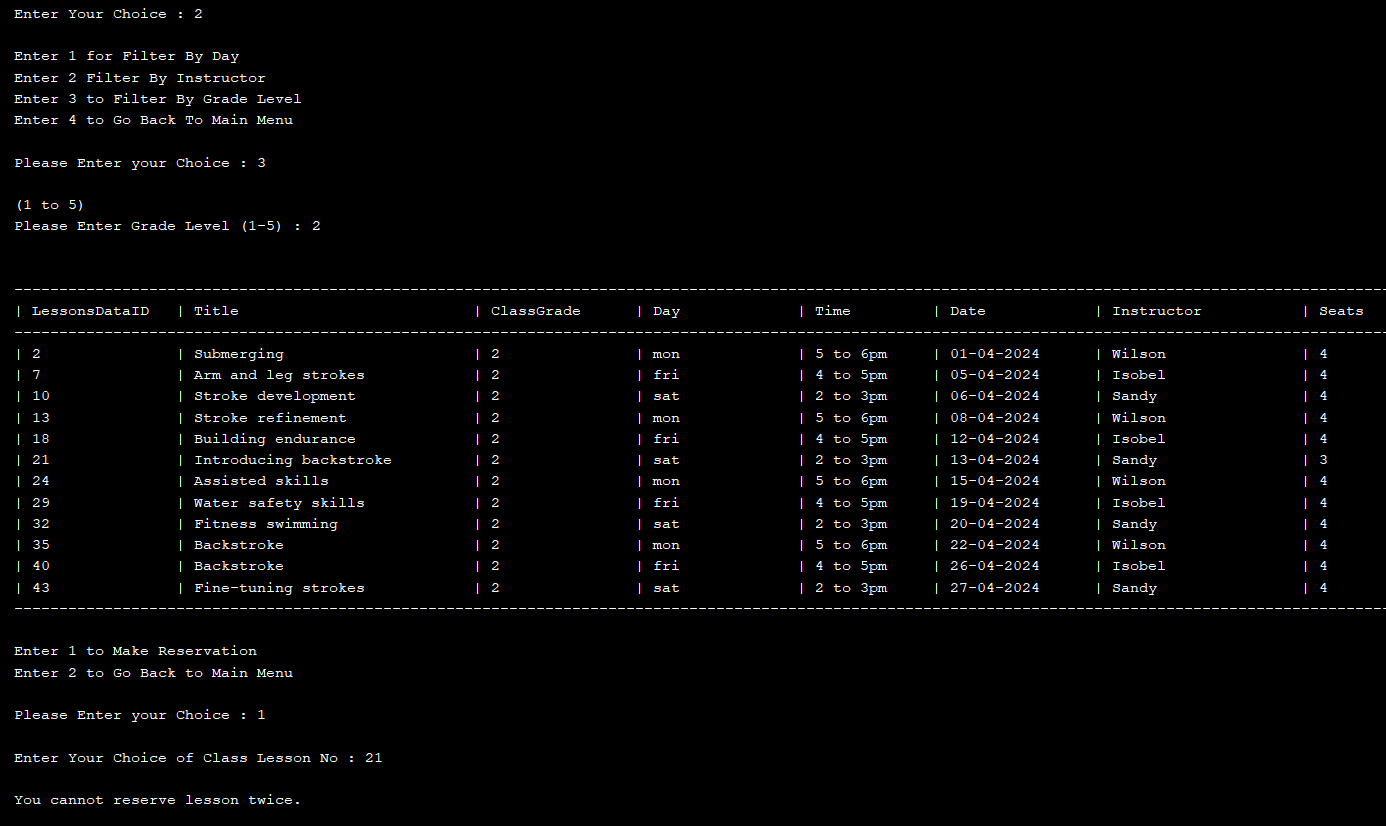
Login Process 🡪



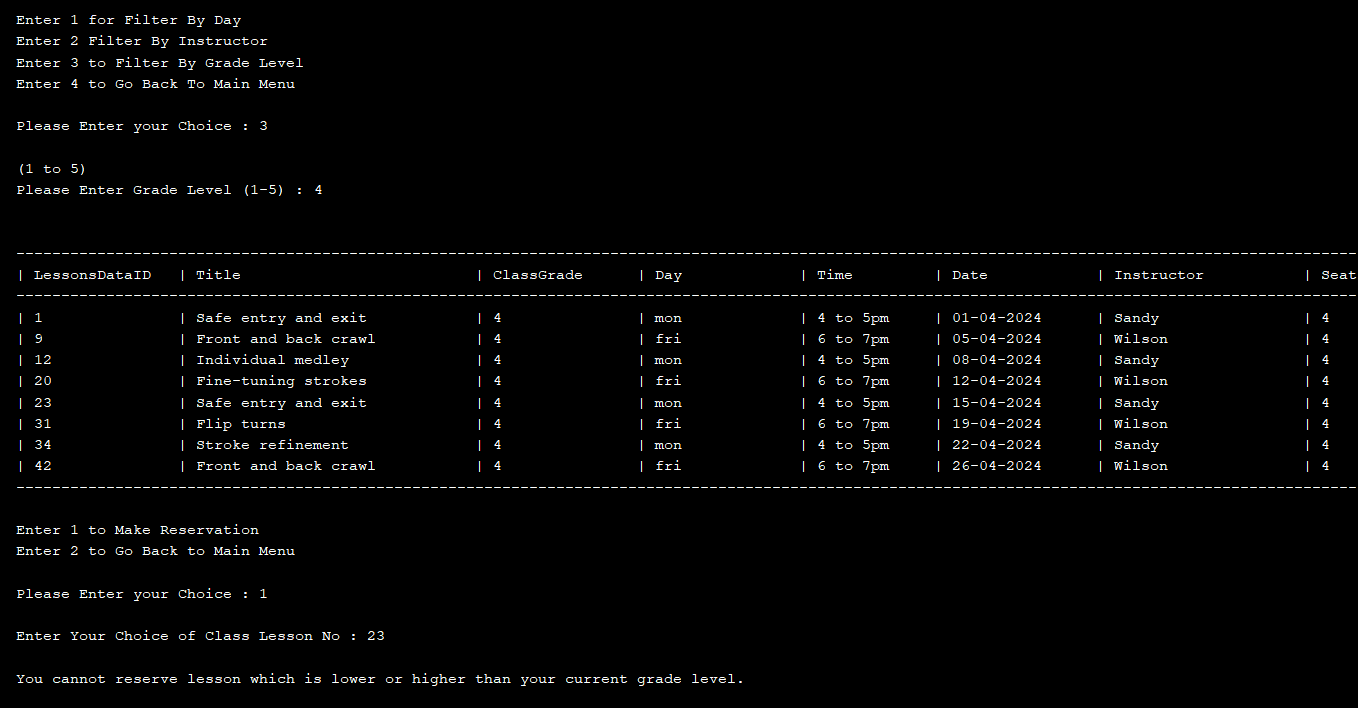
Reserve class 🡪



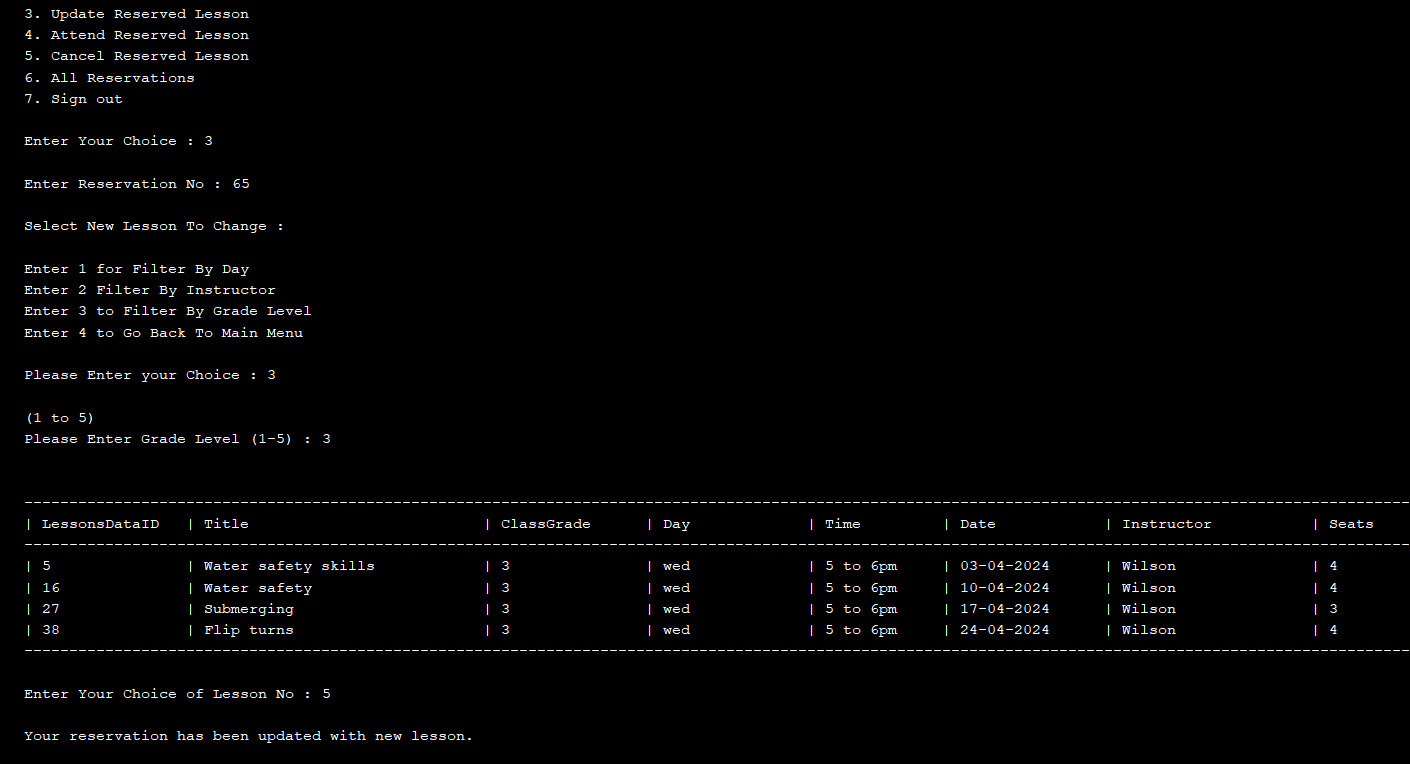
Twice reservation 🡪



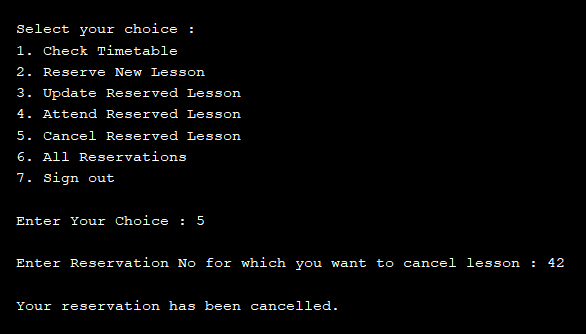
Not allowed to reserve class higher than /lower than current grade level 🡪



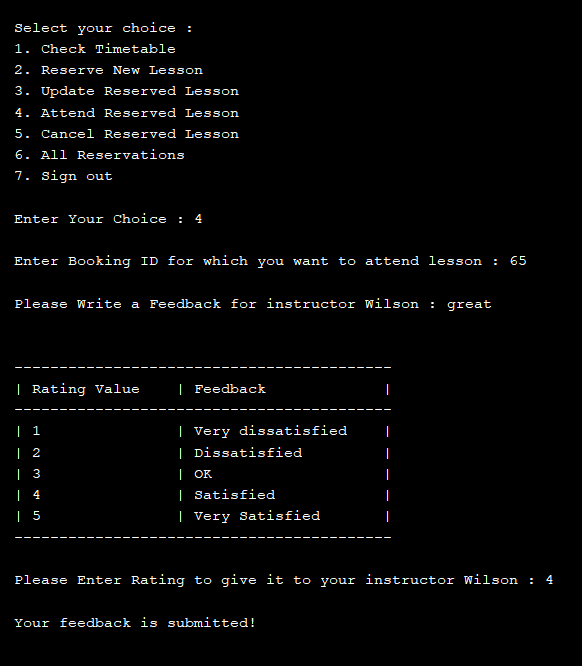
Change reservation 🡪



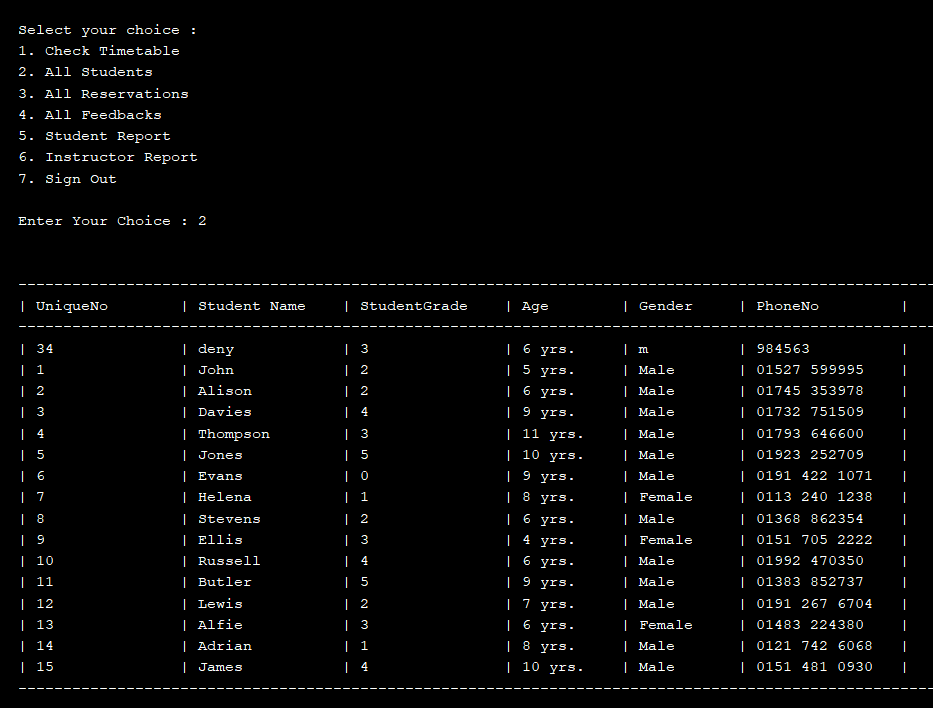
Cancel reservation 🡪



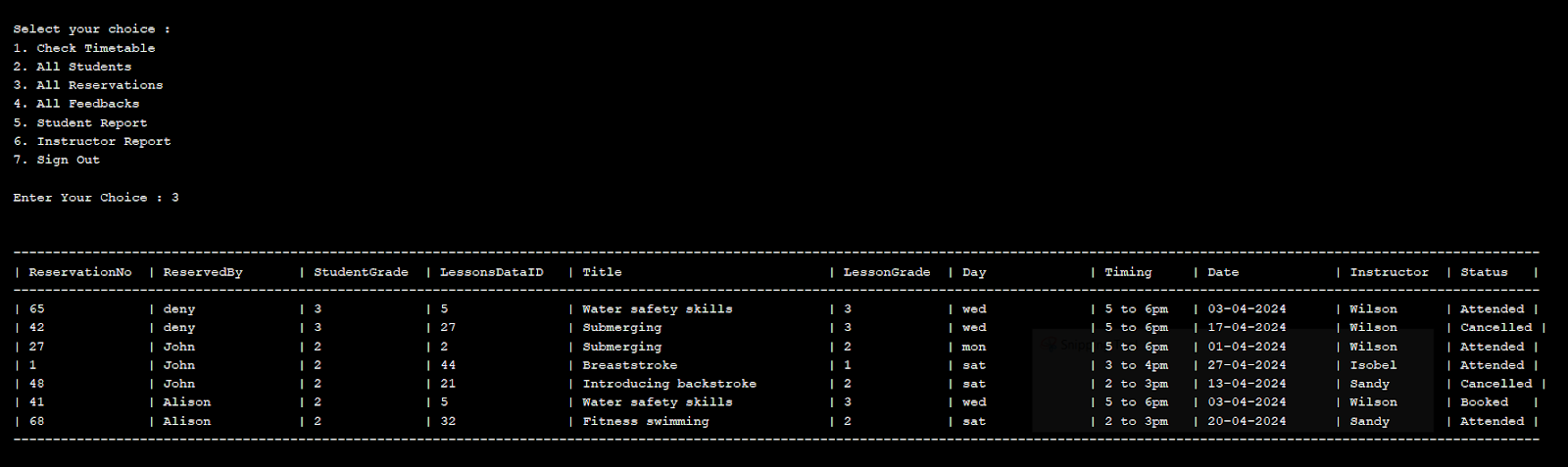
Attend Reservation 🡪



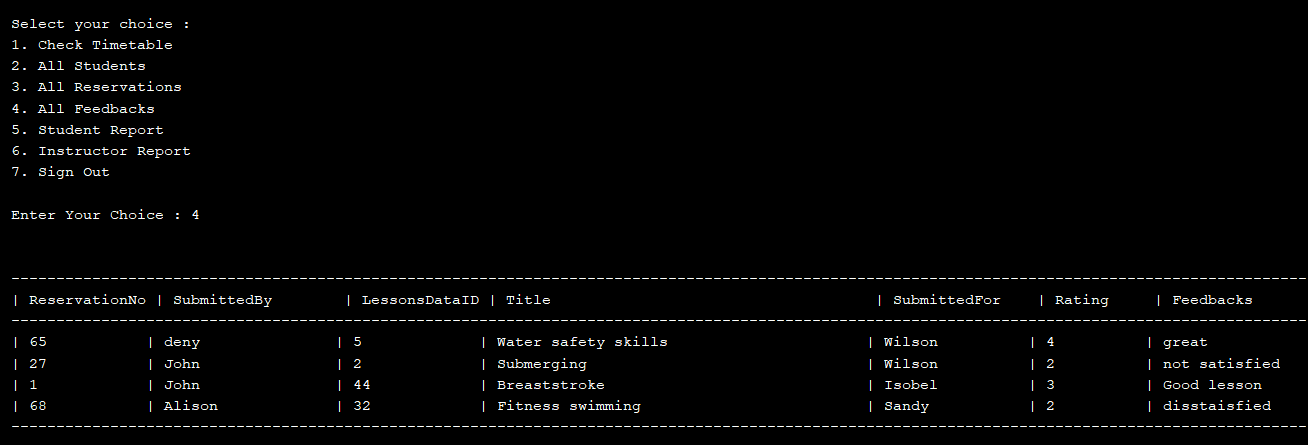
All Students  🡪



All reservations 🡪



All Feedbacks 🡪



# REFERENCES

* Almogahed, A. and Omar, M. (2021). Refactoring Techniques for Improving Software Quality: Practitioners’ Perspectives. *Journal of Information and Communication Technology*, 20(No.4), pp.511–539. doi:https://doi.org/10.32890/jict2021.20.4.3.
* McDonough, J.E. (2017). Command Design Pattern. *Object-Oriented Design with ABAP*, pp.255–264. doi:https://doi.org/10.1007/978-1-4842-2838-8\_20.
* Toure, F., Badri, M. and Lamontagne, L. (2014). A metrics suite for JUnit test code: a multiple case study on open source software. *Journal of Software Engineering Research and Development*, 2(1). doi:https://doi.org/10.1186/s40411-014-0014-6.
* Visual Paradigm (2019). *UML Class Diagram Tutorial*. [online] Visual-paradigm.com. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.