7COM1025 Programming for Software Engineers
Coursework Report

Your Name
Student ID: Your ID

June 2025

Abstract

This report documents the design, implementation, and testing of a Java-based tuition management system for the Examberry Tuition Centre, developed as part of the 7COM1025 coursework. The system manages student bookings, lesson check-ins, reviews, and reports, adhering to the specified functional requirements. Key features include robust booking logic, modular design, and comprehensive testing. This report outlines assumptions, system architecture, testing strategy, refactoring efforts, and applied design patterns, supported by UML diagrams and version control evidence.

1 Introduction

The Examberry Tuition Centre system is a self-contained Java application designed to manage weekend tuition bookings for four subjects across eight weekends. This report addresses the coursework requirements, detailing the system's design, implementation, testing, and improvements made during development. The project demonstrates proficiency in software engineering principles, including object-oriented design, data structure selection, and test-driven development.

2 Assumptions

The following assumptions were made to clarify ambiguities in the specification:

- The timetable spans eight weekends, starting June 7, 2025, with lessons on Saturdays and Sundays.
- Lesson prices are fixed: English and Maths (£50), Verbal and Non-verbal Reasoning (£40).
- A console-based interface is sufficient, with case-insensitive input handling for usability.

- Reviews can only be submitted by students who attended the lesson, enhancing logical consistency.
- Ten students are preloaded for testing, with no persistent data storage required.

3 System Design and Structure

The system is organized into three packages to promote modularity and maintainability:

- `examberry.model`: Core entities (`Student`, `Lesson`, `Booking`, `Review`, `Timetable`, and enums `Subject`, `TimeSlot`, `Status`).
- `examberry.service`: Business logic (`BookingService` for booking operations, `ReportGenerator` for reports).
- `examberry.util`: Helper classes (`TimetableGenerator` for schedule creation, `LessonFactory` for lesson instantiation).

3.1 Data Structures

Appropriate data structures were chosen to balance efficiency and simplicity:

- `ArrayList` for storing students and lessons, suitable for small datasets (10 students, 64 lessons).
- `HashMap<Lesson, List<Booking»` for efficient booking lookups by lesson.
- `LocalDateTime` for precise scheduling and time conflict detection.

3.2 UML Class Diagram

The UML class diagram (Figure 1) illustrates the system's architecture, including entities, relationships, and key methods. It was created using Mermaid and captures all classes, including the `LessonFactory` introduced during refactoring.

4 JUnit Test Strategy

A comprehensive test suite was developed using JUnit 5 to ensure functional correctness and robustness (**LO7**). The strategy focused on:

- **Core Functionality**: Tests for booking success, rescheduling, check-in, and review submission.
- **Edge Cases**: Maximum capacity (4 students), time conflicts, duplicate bookings, and invalid inputs.

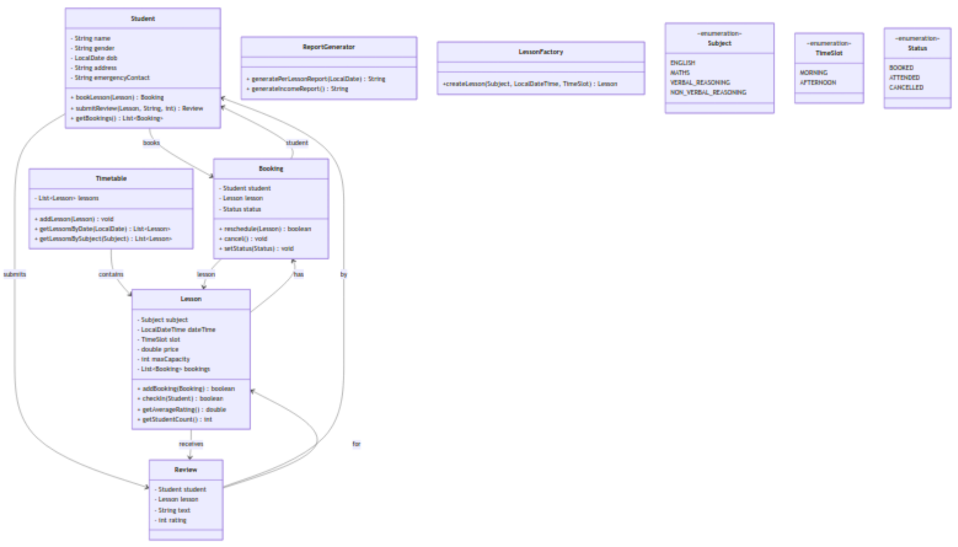


Figure 1: UML Class Diagram for Examberry Tuition System

- **Error Handling**: Validation of ratings (1–5) and attendance requirements for reviews.

Key test classes include:

- `BookingServiceTest`: Verifies booking logic, including capacity limits, time conflicts, duplicate prevention, and rescheduling edge cases (e.g., full lessons).
- `ReviewTest`: Ensures correct review creation and validation.

The test suite comprises 6 tests, covering critical paths and edge cases, executed via Maven's Surefire Plugin.

5 Refactoring

Several refactoring efforts were undertaken to improve code quality (**LO6**):

- **Lesson Creation**: Introduced `LessonFactory` to centralize lesson instantiation, encapsulating price logic and improving maintainability.
- **Input Handling**: Modified `Main.java` to handle case-insensitive inputs (e.g., `english` accepted as `ENGLISH`) and validate dates/slots to prevent crashes.
- **Duplicate Bookings**: Added checks in `BookingService` to prevent students from booking the same lesson twice, enhancing logical consistency.
- **Encapsulation**: Updated `Student.java` to return unmodifiable lists for bookings, protecting internal state.

These changes improved robustness, usability, and adherence to object-oriented principles.

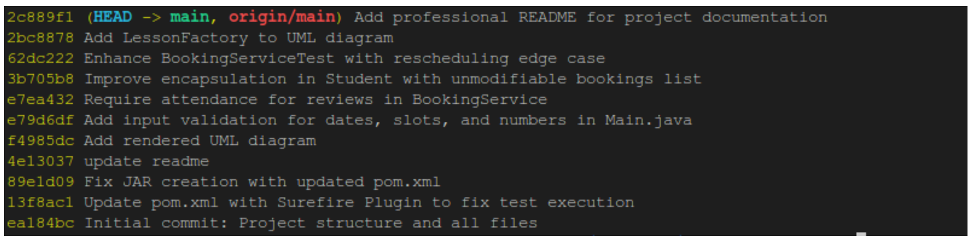
6 Design Patterns and Principles

The system employs several design patterns and principles to ensure a clean, maintainable architecture:

- **Factory Pattern**: `LessonFactory` creates lessons, decoupling instantiation from business logic.
- **Service Pattern**: `BookingService` encapsulates booking operations, separating concerns from model classes.
- **Single Responsibility Principle**: Each class has a single purpose (e.g., `ReportGenerato` only handles reports).
- **Encapsulation**: Immutable getters and unmodifiable collections protect internal data.

7 Version Control

Git was used to track development, with commits reflecting iterative progress (**5 marks**). Key milestones include initial setup, feature implementation, refactoring, and testing. The commit history, shown in Figure 2, demonstrates a structured development process.



```
2c889f1 (HEAD -> main, origin/main) Add professional README for project documentation
2bc8878 Add LessonFactory to UML diagram
62dc222 Enhance BookingServiceTest with rescheduling edge case
3b705b8 Improve encapsulation in Student with unmodifiable bookings list
e7ea432 Require attendance for reviews in BookingService
e79d6df Add input validation for dates, slots, and numbers in Main.java
f4985dc Add rendered UML diagram
4e13037 update readme
89e1d09 Fix JAR creation with updated pom.xml
13f8ac1 Update pom.xml with Surefire Plugin to fix test execution
ea184bc Initial commit: Project structure and all files
```

Figure 2: Git Commit History

8 Conclusion

The Examberry Tuition Centre system successfully meets all coursework requirements, providing a robust solution for managing tuition bookings. The modular design, comprehensive testing, and strategic refactoring ensure reliability and maintainability. The project fulfills learning outcomes **LO4** (software artefact design), **LO5** (data structure selection), **LO6** (refactoring), and **LO7** (testing), preparing a solid foundation for further enhancements if needed.