Boost Physio Clinic Booking System Report

**Module:** 7COM1025 – Programming for Software Engineers  
**Student Name:****Submission Date:** 28 April 2025  
**IDE Used:** IntelliJ IDEA Community Edition  
**Language:** Java

# **Introduction:**

This report outlines the design and implementation of a Java-based appointment booking system for Boost Physio Clinic. The purpose of the system is to allow patients to book and manage appointments with physiotherapists while ensuring that conflicts such as double-bookings are prevented.

The system is built using object-oriented programming principles, aiming to be robust, scalable, and easy to maintain. It supports appointment booking, cancellation, status updates, reporting, and automated testing through JUnit.

# **Methodology:**

The system was developed using an **iterative and incremental software development methodology**. The overall development process included the following phases:

## **Requirement analysis:**

* Studied the assignment brief and extracted functional requirements:
* Booking system for physiotherapy appointments.
* Avoid time slot conflicts.
* Handle appointment status changes (Scheduled, Cancelled, Attended).
* Provide reports and listings.

## **Object-Oriented Design:** Identified and modeled real-world entities as Java classes: Patient, Physiotherapist, Treatment, Appointment, and BookingSystem. Defined relationships and interactions between objects using encapsulated fields and methods.

## **Implementation:**

* Implemented class-by-class, starting from basic data classes to core logic.
* Used IntelliJ IDEA Community Edition for writing and organizing code.
* Wrote a complete Main class to simulate clinic scenarios.

## **Testing:**

* Applied test-driven development (TDD) principles for key classes.
* Wrote unit tests using JUnit 5 to validate booking, cancellation, and attendance logic.
* Manually verified outputs using system print statements in the Main class.
* Conducted several test runs, identified bugs (e.g., attending cancelled appointments), and resolved them.
* Improved modularity and readability by separating logic into dedicated classes.

# **Features Implemented:**

**Patient & Physiotherapist Management:**

* Creation and storage of patient and physiotherapist profiles.
* Contact and address information handling.

**Appointment Booking:**

* Book an appointment by linking a patient, physiotherapist, treatment, and timeslot.
* Appointments are initially marked as Scheduled.

**Conflict Detection:**

* Prevents physiotherapists from being double-booked at the same time.
* Ensures only one appointment per physiotherapist per time slot (unless cancelled).

**Status Transitions:**

* Appointments can be marked as Cancelled or Attended.
* Cancellations prevent further modifications like attendance.

**Reusability of Cancelled Slots:**

* Cancelled slots can be rebooked.
* Ensures flexibility and efficient scheduling.

**Reports & Listings:**

* Generate a full report of all appointments with current statuses.
* List all available (i.e., not attended or cancelled) appointments.

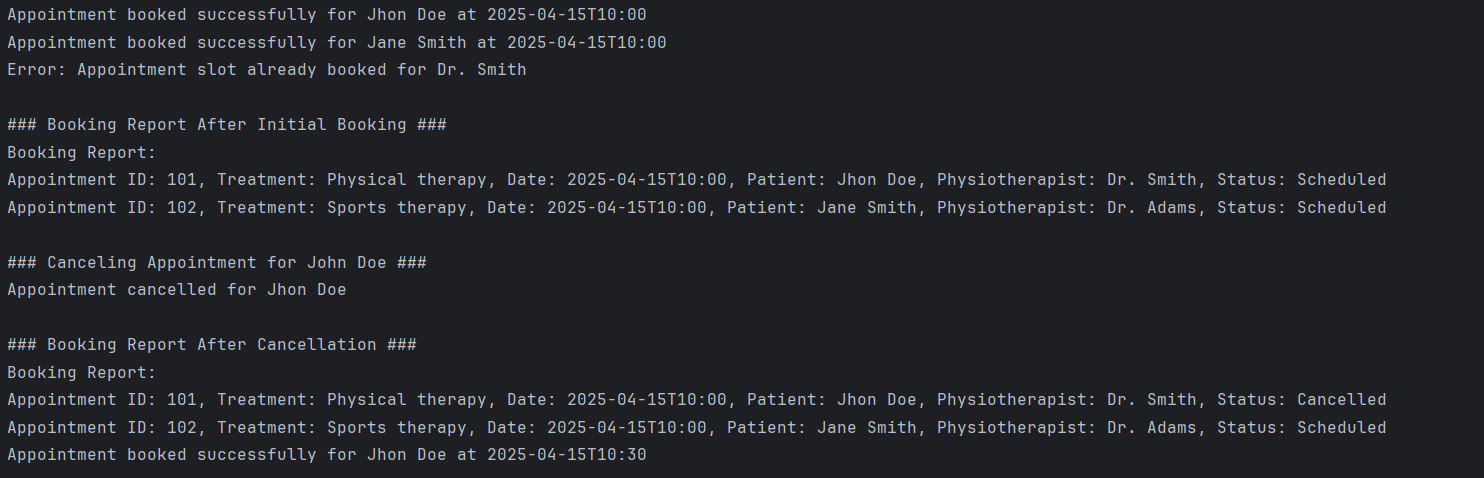
# **Code Structure:**

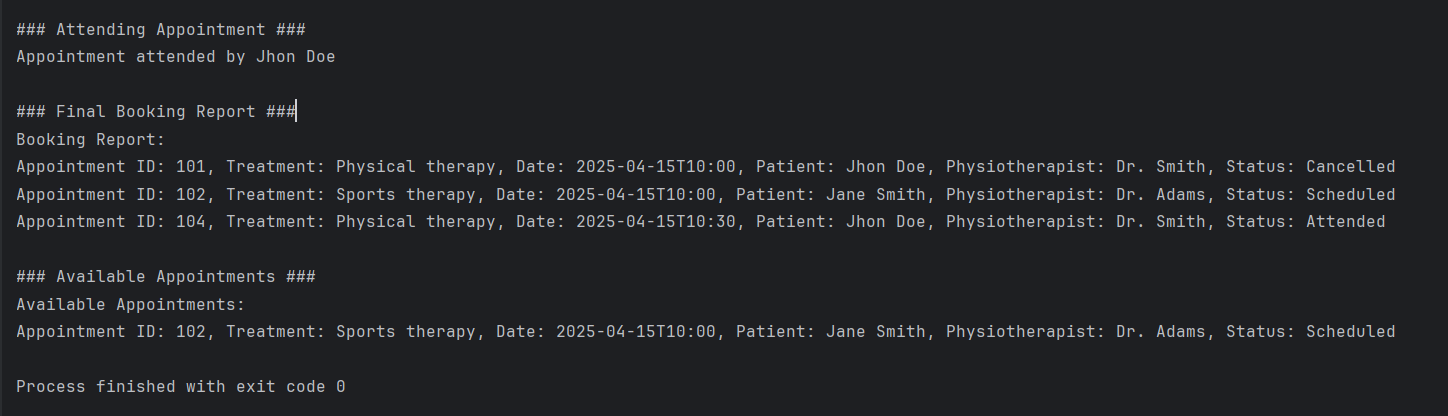
|  |  |
| --- | --- |
| **Class Name** | **Purpose** |
| Main | Entry point to run simulations and print outputs. |
| Patient | Holds patient details and appointment history. |
| Physiotherapist | Holds physiotherapist details and expertise. |
| Treatment | Encapsulates treatment information. |
| Appointment | Represents a scheduled treatment session, with status management. |
| BookingSystem | Core logic for managing appointments and ensuring valid scheduling. |
| JUnit Tests | Separate classes for testing booking logic, patient data, and attendance. |

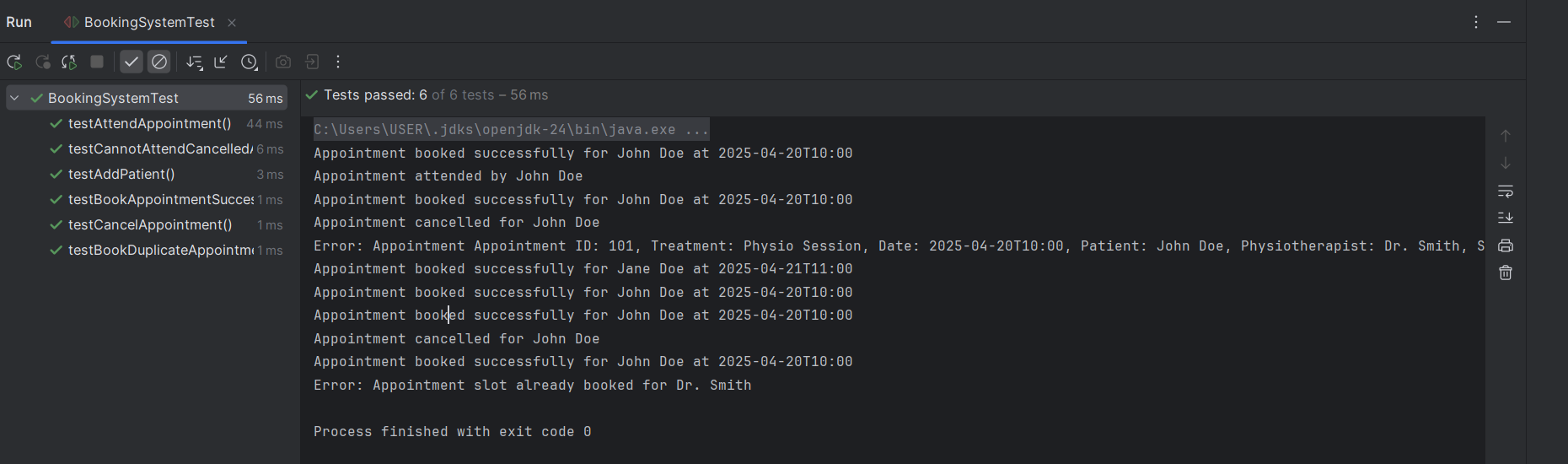
Each component interacts through well-defined methods, ensuring maintainability and modularity.

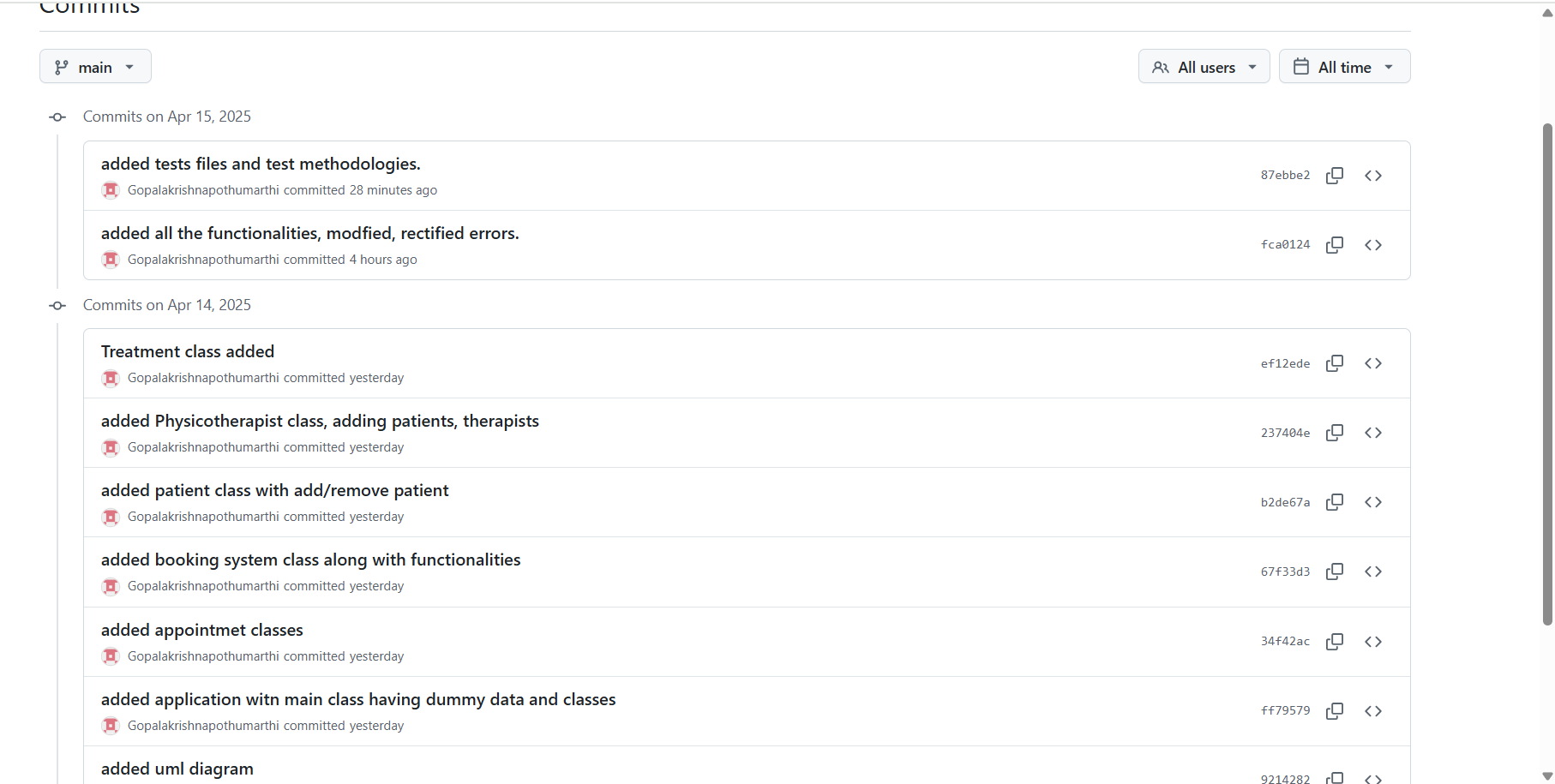
**Screenshots of Tests and Demo Output:**

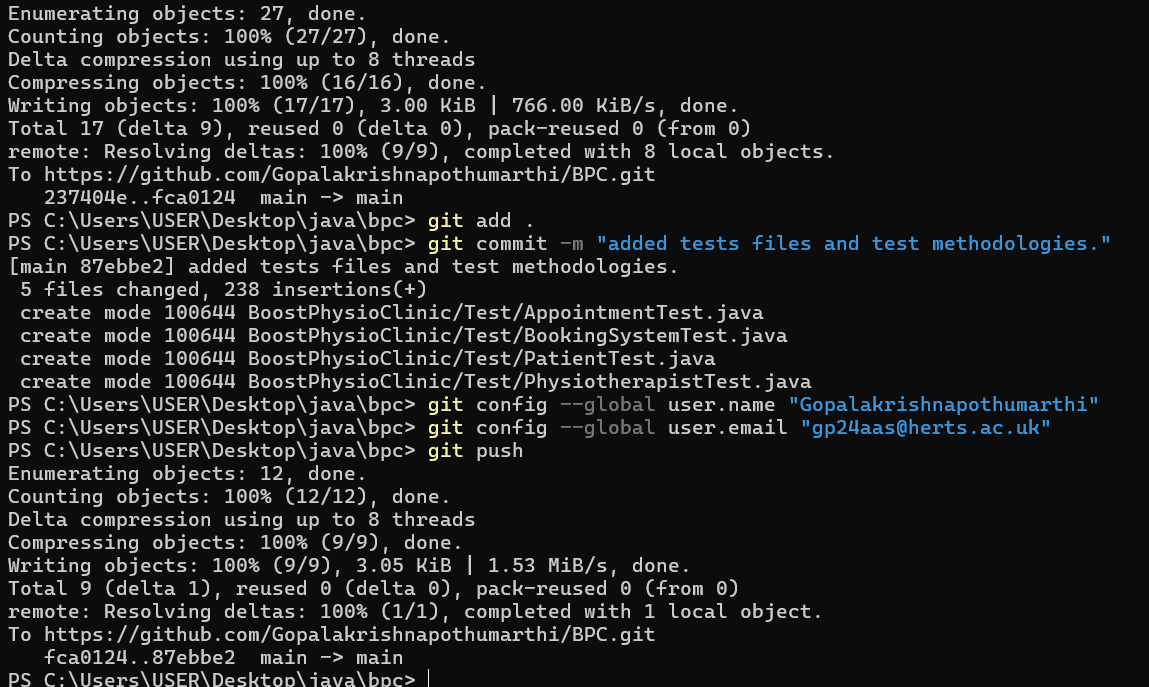
**Console Output Samples:**





**IntelliJ JUnit Output:**  


**Git Log:**  




# **Challenges Faced & Solutions**

**Managing Appointment Conflicts:**

* **Challenge:** Preventing multiple appointments from being booked for the same physiotherapist at the same time was tricky, especially with multiple patients and treatments involved.
* **Solution**: Implemented a conflict check in the BookingSystem.bookAppointment() method that compares appointment date, time, and physiotherapist, and disallows booking if the slot is already taken and not canceled.

**Handling Appointment Status Transitions**

* **Challenge**: Ensuring appointments could not be attended after being canceled, or canceled once already attended, required precise status management.
* **Solution**: Created attend() and cancel() methods in the Appointment class that restrict transitions based on current status (Scheduled, Attended, Cancelled), using conditional logic to enforce valid state changes.

**Integrating Patient and Appointment Classes**

* **Challenge**: Keeping the Patient's appointment list in sync with the central BookingSystem was difficult when appointments were created or canceled dynamically.
* **Solution**: Made sure bookAppointment() in BookingSystem also updated the patient's appointment list through the patient.bookAppointment() method.

**Displaying Human-Readable Reports**

* **Challenge**: Printing clear and detailed reports that included treatment, patient, physio, and status details without overwhelming the output.
* **Solution**: Implemented a getDetails() method in Appointment to neatly format all key details into a single string, improving readability of the booking report.

# **Conclusion:**

The Boost Physio Clinic Booking System successfully delivers a reliable and maintainable Java-based solution for managing physiotherapy appointments. By applying object-oriented design principles and following a structured development methodology, the system ensures key features such as conflict-free booking, appointment tracking, and flexible status management.

Through this project, I gained hands-on experience in designing modular systems, implementing core business logic, and writing unit tests to validate functionality. The challenge of simulating real-world constraints like time slot conflicts and dynamic appointment status transitions was met with effective coding strategies and careful planning.

This implementation not only fulfills the core requirements of the assignment but also lays the groundwork for future expansion, such as integrating a graphical user interface, connecting to a database for persistent storage, or adding multi-user access with authentication. Overall, the project demonstrates a practical application of software engineering principles in a healthcare context.