**Boost Physio Clinic (BPC) Booking System - Report**

**Student Name:** Syed Khurram Ali  
**Student ID:** 23068799  
**Course Code:** 7COM1025  
**Instructor:** Dr. Hui Cheng  
**University:** University of Hertfordshire  
**Date:** April 2025  
Github Url: https://github.com/Khurram-ak/Booking\_System

**1. Introduction**

This report presents the design and implementation of the Boost Physio Clinic (BPC) Booking System. The objective of the system is to manage the scheduling of treatment appointments between patients and physiotherapists. It supports booking, cancellation, attendance tracking, and end-of-term reporting. The system utilizes a four-week timetable and provides both command-line and graphical interfaces for user interaction.

**2. Assumptions Made**

* Physiotherapists and a subset of patients are pre-registered at the start of the program.
* All data is either hardcoded or read from text files; no external databases are used.
* The four-week timetable includes fixed and non-repeating appointments.
* Each appointment is one-to-one (no overlapping or duplicate bookings).
* The application operates entirely in memory; data persistence after closing the program is not required.

**3. UML Class Diagram**

(Refer to Appendix A for the full UML diagram)

**Key Classes:**

* Physiotherapist
* Patient
* Treatment
* Appointment
* BookingSystem
* ReportGenerator

**4. System Structure and Design**

### 4.1 Class Descriptions

* **Physiotherapist**: Represents a medical professional with a unique ID, contact details, areas of expertise, and a weekly timetable.
* **Patient**: Represents a client of the clinic. Each has a unique ID, name, address, and phone number.
* **Treatment**: Defines the name, date/time, and physiotherapist associated with a therapy session.
* **Appointment**: Connects a treatment and a patient, and holds a status (booked, cancelled, attended).
* **BookingSystem**: Core controller class that manages all operations including patient registration, appointment booking, and data validation.
* **ReportGenerator**: Responsible for creating formatted reports summarizing the activities of each physiotherapist.

### 4.2 Design Patterns and Principles

* **Model-View-Controller (MVC)**: Clean separation between business logic (Model), user interface (View), and control logic (Controller).
* **Single Responsibility Principle**: Each class focuses on a specific role within the system.
* **DRY (Don't Repeat Yourself)**: Common functionality such as date parsing and appointment validation is modularized.

**5. Core Functionalities**

### 5.1 Add/Remove a Patient

* Patients can be registered or removed via a simple Swing-based form.

### 5.2 Book a Treatment Appointment

* Patients can browse appointments by area of expertise or search by physiotherapist name.
* System enforces checks for time conflicts and duplicate bookings.

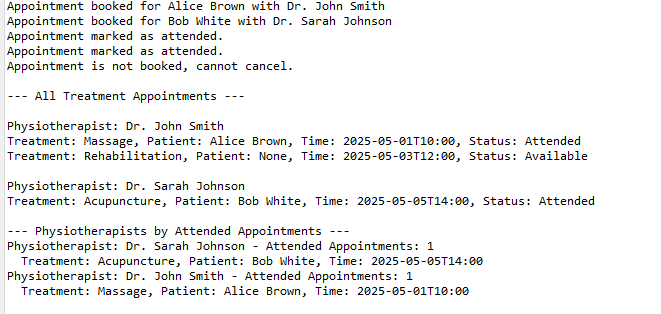
### 5.3 Change/Cancel a Booking

* Change = cancel the existing booking and book a new one.
* Booking IDs are used to track and manage appointments reliably.

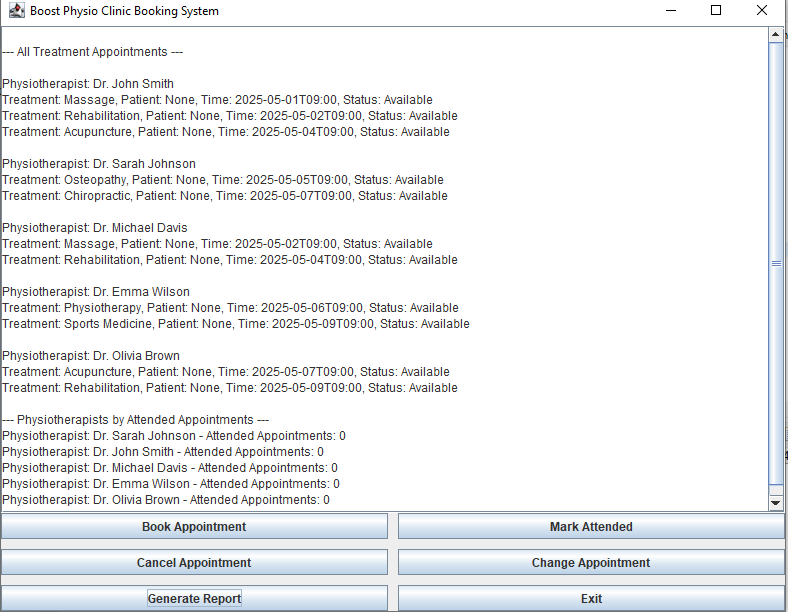
### 5.4 Attend Appointment

* Patients can mark an appointment as attended.
* Appointment status transitions from "booked" to "attended".

### 5.5 Generate and Print Report

* Final report lists all appointments per physiotherapist with treatment names, patients, and status.
* Physiotherapists are sorted by the number of attended appointments in descending order.
* **Console Output:**
* 

**GUI:**



**6. JUnit Testing**

JUnit 5 was used to test the core functionalities of the Boost Physio Clinic Booking System. Unit tests were designed to cover both typical use cases and edge cases, ensuring system stability, data integrity, and correct business logic execution.

**Key Methods Tested**

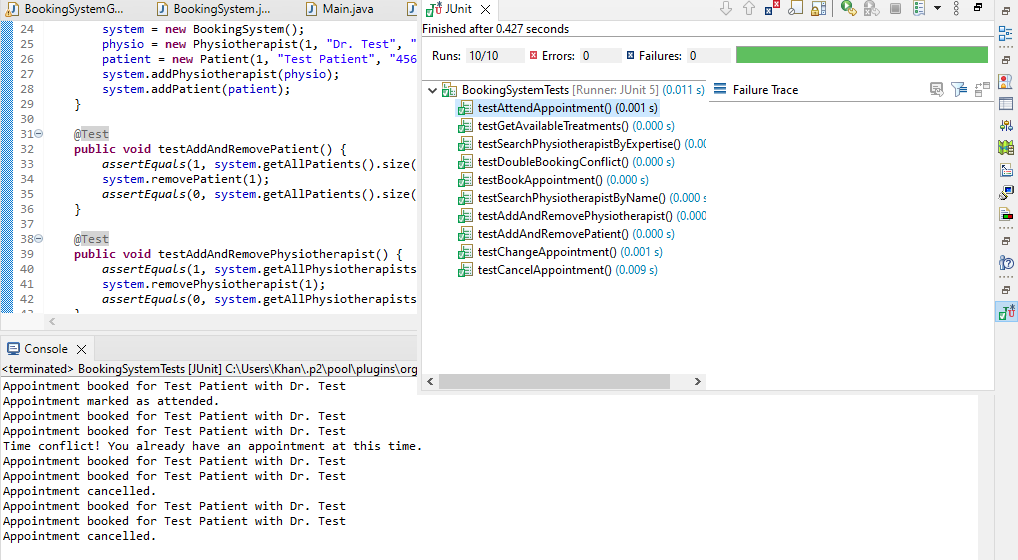
1. **Add and Remove Patients**
   * Ensures patients can be added and removed from the system.
   * Verifies the list size before and after removal.
2. **Add and Remove Physiotherapists**
   * Confirms that physiotherapists are correctly added or removed.
   * Used to test internal list consistency.
3. **Search Functionalities**
   * **By Expertise**: Verifies correct filtering of physiotherapists based on their areas of expertise.
   * **By Name**: Confirms the search returns accurate results and handles unmatched names gracefully.
4. **Book Appointment**
   * Tests successful booking when a treatment slot is available.
   * Ensures the appointment status is updated to “Booked” and patient is assigned.
5. **Double Booking / Time Conflict**
   * Simulates overlapping appointments.
   * Verifies that the system correctly blocks duplicate or conflicting bookings.
6. **Cancel Appointment**
   * Ensures a booked appointment can be canceled.
   * Updates status to “Cancelled” and detaches the patient from the treatment.
7. **Attend Appointment**
   * Changes the status of a booked appointment to “Attended”.
   * Ensures correctness in tracking attendance.
8. **Change Appointment**
   * Cancels the existing appointment and books a new one in a different timeslot.
   * Ensures time conflict resolution and updates to patient references.
9. **Get Available Treatments**
   * Returns all treatments with no assigned patient (i.e., available to book).
   * Verifies that only unbooked treatments are listed.

**Testing Tools & Coverage**

* **Framework Used**: JUnit 5
* **Assertions**: assertTrue, assertFalse, assertEquals, assertNull, and collection size checks.
* **Edge Cases Covered**:
  + Booking at the same time twice (conflict handling).
  + Searching non-existent physiotherapist.
  + Reassigning patients between treatments.
* **Test Independence**: Each test runs in isolation with a fresh setup (@BeforeEach) to ensure no dependency between tests.

#### Result

All test cases executed successfully, validating key system features under expected and edge-case scenarios. This provides a solid confidence level for correctness, especially for booking operations and appointment state transitions.



**7. Refactoring and Improvements**

* Refactored logic into smaller methods to improve readability.
* Introduced a utility class for date and time validation.
* Separated UI logic from backend for maintainability.
* Applied exception handling for user inputs and invalid operations.

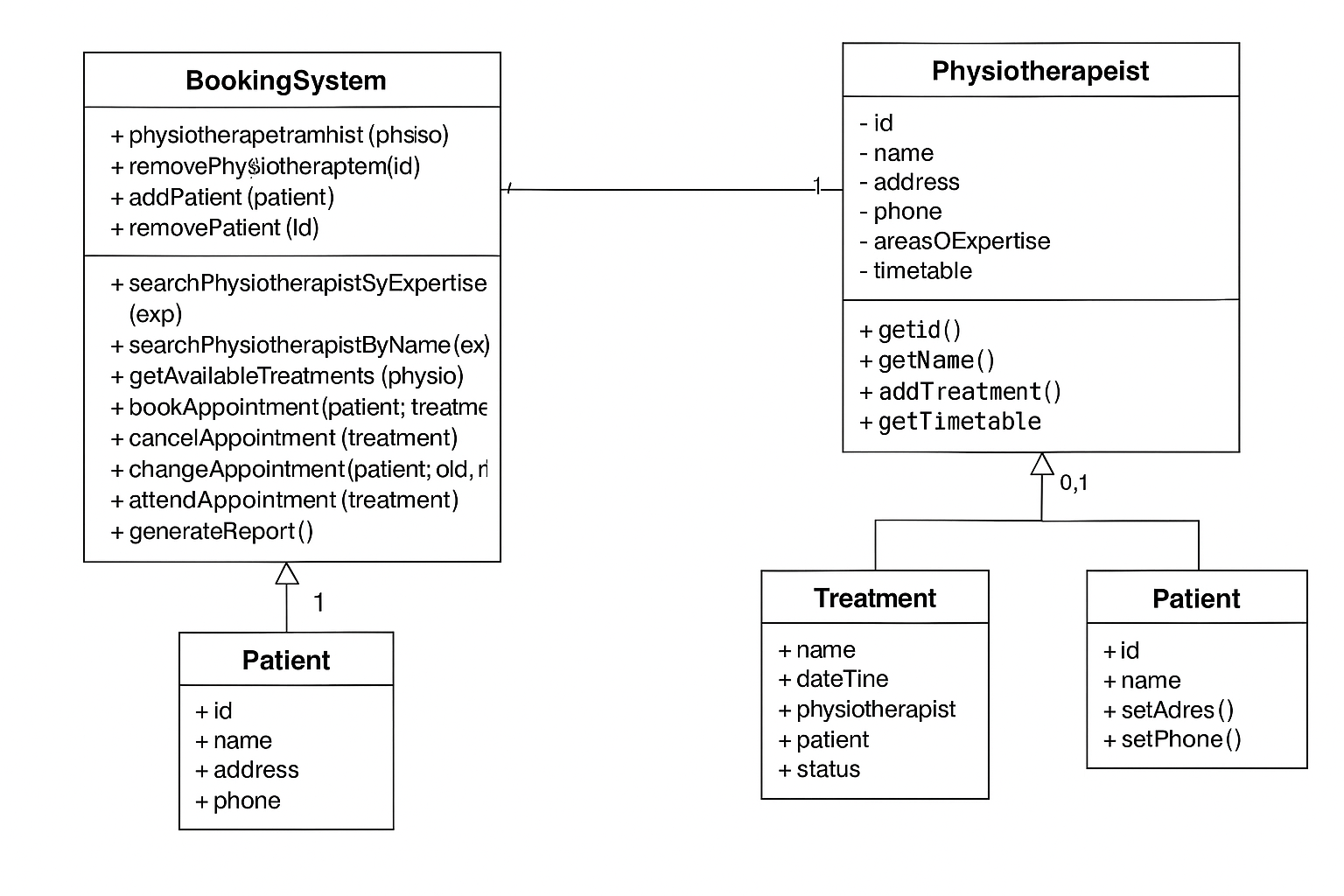
**8. Version Control Usage**

Version control was handled using Git. Major commits included:

* Initial project setup
* Core class implementations
* GUI integration
* JUnit tests
* Report generation logic

(Refer to Appendix B for screenshots and commit logs.)

**Appendix A: UML Class Diagram**



**Appendix B: Version Control Snapshots**

**Note: I pushed the first commit mistakenly by my company github account but then I have changed the credentials and pushed it again with my own account.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Appendix C: JUnit Test Output**

