Hospital Appointment System

Object-Oriented Programming Report

Team No.: 7

1 Introduction

Appointment scheduling is still done by hand in many hospitals using basic registration logs or paper records. This traditional approach is inefficient and often leads to problems such as double booking, missing patient information, and difficulty in tracking doctor availability, etc. As the number of doctors and patients increases, managing appointments manually becomes more complex, difficult and error-prone.

To address these challenges, this project presents a **Hospital Appointment System** developed using the Java programming language and Object-Oriented Programming (OOP) principles. The system provides a menu-driven console interface that allows hospital staff to manage doctors, patients, and appointment slots in a structured and organized manner.

In this report, we describe the system design and explain how OOP principles were applied to meet the project requirements clearly and effectively powered by screenshots after running.

2 Class Diagram

Figure 1 illustrates the Unified Modeling Language (UML) class diagram of the Hospital Appointment System. The diagram shows the main classes, their attributes and methods, and the relationships between them.

The UML diagram demonstrates that the system follows a clear object-oriented structure. At the top of the hierarchy is the abstract `Person` class, which represents a general person in the hospital system. It contains common attributes such as name, unique identifier, national ID, and contact information. The class also defines common setter and getter methods and is not instantiated directly.

Both the `Doctor` and `Patient` classes inherit from the `Person` class. The `Doctor` class extends it by adding attributes related to the doctor's specialization and available time slots. It includes methods that allow managing these slots, which are later used to generate appointment times.

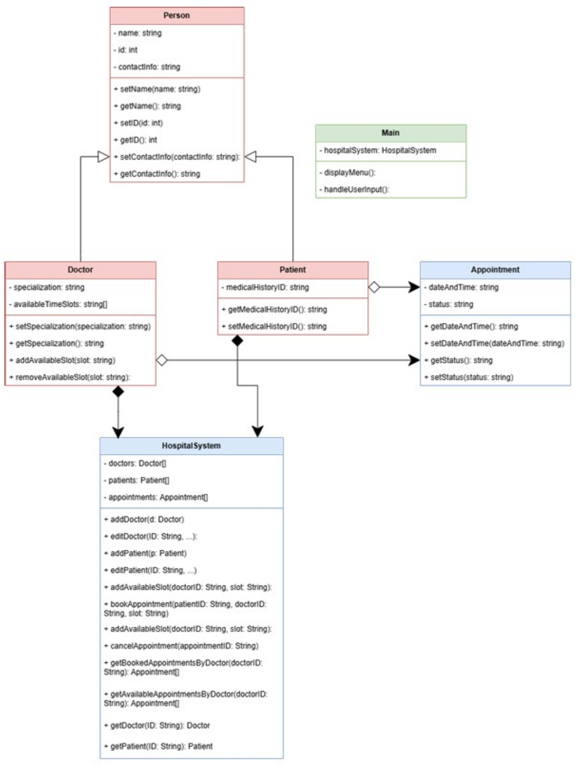

Person
- name: string
- id: int
- contactInfo: string
+ setName(name: string)
+ getName(): string
+ setID(id: int)
+ getID(): int
+ setContactInfo(contactInfo: string):
+ getContactInfo(): string
Main
- hospitalSystem: HospitalSystem
- displayMenu():
- handleUserInput():
Doctor
- specialization: string
- availableTimeSlots: string[]
+ setSpecialization(specialization: string)
+ getSpecialization(): string
+ addAvailableSlot(slot: string)
+ removeAvailableSlot(slot: string):
Patient
- medicalHistoryID: string
+ getMedicalHistoryID(): string
+ setMedicalHistoryID(): string
Appointment
- dateAndTime: string
- status: string
+ getDateAndTime(): string
+ setDateAndTime(dateAndTime: string)
+ getStatus(): string
+ setStatus(status: string)
HospitalSystem
- doctors: Doctor[]
- patients: Patient[]
- appointments: Appointment[]
+ addDoctor(d: Doctor)
+ editDoctor(ID: String, ...):
+ addPatient(p: Patient)
+ editPatient(ID: String, ...)
+ addAvailableSlot(doctorID: String, slot: String):
+ bookAppointment(patientID: String, doctorID: String, slot: String)
+ addAvailableSlot(doctorID: String, slot: String):
+ cancelAppointment(appointmentID: String)
+ getBookedAppointmentsByDoctor(doctorID: String): Appointment[]
+ getAvailableAppointmentsByDoctor(doctorID: String): Appointment[]
+ getDoctor(ID: String): Doctor
+ getPatient(ID: String): Patient

Figure 1: UML class diagram of the Hospital Appointment System

The `Patient` class extends `Person` and introduces a medical history identifier that uniquely represents each patient's medical record. This design allows patient-specific data to be managed independently while still sharing common personal information through inheritance.

The `Appointment` class represents individual appointment slots. It stores the appointment date and time, booking status, and the association between doctors and patients. An appointment is considered available when it is not assigned to a patient.

The `HospitalSystem` class acts as the core logic of the application. It manages collections of doctors, patients, and appointments and provides operations for adding, editing, booking, canceling, and retrieving appointments. This class enforces validation rules to ensure data consistency across the system.

Finally, the `Main` class represents the user interface layer. It provides a menu-driven console interface and interacts with the `HospitalSystem` class to handle user input and trigger system operations.

3 Object-Oriented Programming Principles

3.1 Inheritance

Inheritance is applied by defining the `Person` class as a base class and extending it in the `Doctor` and `Patient` classes. This approach allows shared attributes and behaviors to be reused while enabling each subclass to define its own specific features.

3.2 Encapsulation

Encapsulation is achieved by keeping all class attributes private and accessing them through getter and setter methods. Validation logic is applied when updating sensitive information such as contact numbers and medical history IDs.

3.3 Abstraction

Abstraction is implemented through the abstract `Person` class, which defines general characteristics while leaving role-specific behavior to subclasses.

3.4 Polymorphism

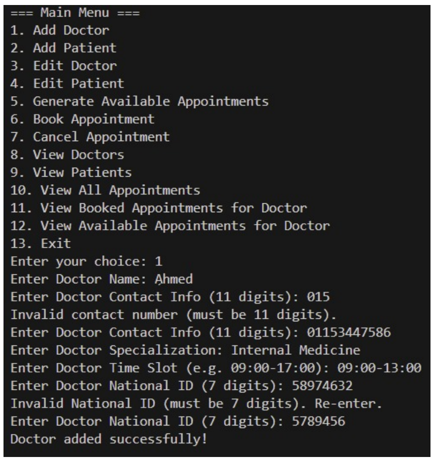
Polymorphism is demonstrated through method overriding. Methods such as `getRole()` and `toString()` behave differently depending on whether the object represents a doctor or a patient.

3.5 User Interface and Main Program

The system is operated through a menu-driven console interface implemented in the `Main` class. User input is collected using the `Scanner` class, and each menu option triggers a specific service in `HospitalSystem`. For example, options are provided to add/edit doctors and patients, generate available appointment slots, book or cancel appointments, and display booked/available appointments for a selected doctor. This design separates the user interaction (Main) from the application logic layer (HospitalSystem).

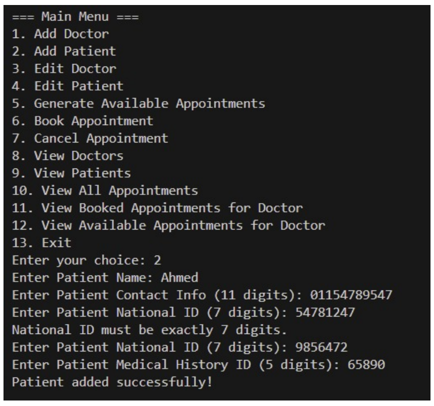
4 Example Outputs and Program Screenshots

The following screenshots show key features of the running system, including data validation, appointment generation, booking, cancellation, and verification of the updated appointment list.



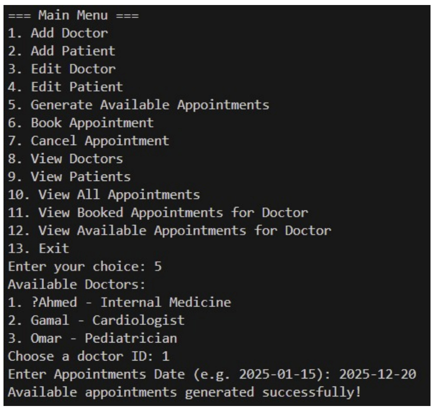
```
=== Main Menu ===
1. Add Doctor
2. Add Patient
3. Edit Doctor
4. Edit Patient
5. Generate Available Appointments
6. Book Appointment
7. Cancel Appointment
8. View Doctors
9. View Patients
10. View All Appointments
11. View Booked Appointments for Doctor
12. View Available Appointments for Doctor
13. Exit
Enter your choice: 1
Enter Doctor Name: Ahmed
Enter Doctor Contact Info (11 digits): 015
Invalid contact number (must be 11 digits).
Enter Doctor Contact Info (11 digits): 01153447586
Enter Doctor Specialization: Internal Medicine
Enter Doctor Time Slot (e.g. 09:00-17:00): 09:00-13:00
Enter Doctor National ID (7 digits): 58974632
Invalid National ID (must be 7 digits). Re-enter.
Enter Doctor National ID (7 digits): 5789456
Doctor added successfully!
```

Figure 2: Adding a doctor with input validation (contact number must be 11 digits and national ID must be 7 digits).



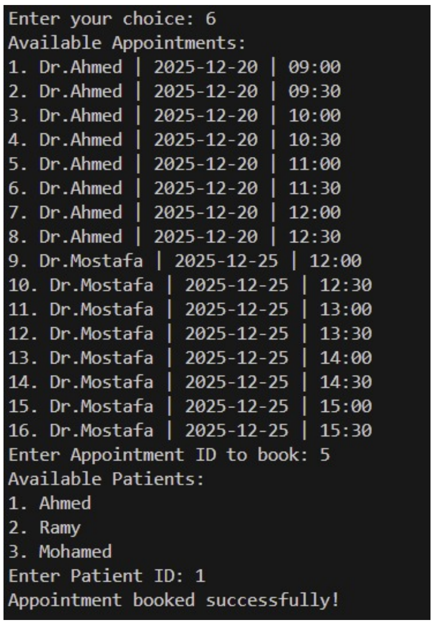
```
=== Main Menu ===
1. Add Doctor
2. Add Patient
3. Edit Doctor
4. Edit Patient
5. Generate Available Appointments
6. Book Appointment
7. Cancel Appointment
8. View Doctors
9. View Patients
10. View All Appointments
11. View Booked Appointments for Doctor
12. View Available Appointments for Doctor
13. Exit
Enter your choice: 2
Enter Patient Name: Ahmed
Enter Patient Contact Info (11 digits): 01154789547
Enter Patient National ID (7 digits): 54781247
National ID must be exactly 7 digits.
Enter Patient National ID (7 digits): 9856472
Enter Patient Medical History ID (5 digits): 65890
Patient added successfully!
```

Figure 3: Adding a patient with validation (contact number, national ID, and medical history ID constraints are enforced).



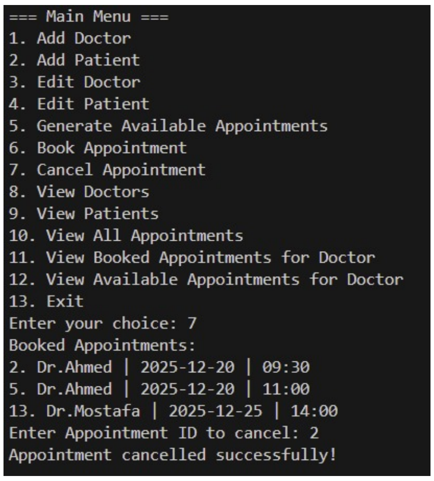
```
=== Main Menu ===
1. Add Doctor
2. Add Patient
3. Edit Doctor
4. Edit Patient
5. Generate Available Appointments
6. Book Appointment
7. Cancel Appointment
8. View Doctors
9. View Patients
10. View All Appointments
11. View Booked Appointments for Doctor
12. View Available Appointments for Doctor
13. Exit
Enter your choice: 5
Available Doctors:
1. ?Ahmed - Internal Medicine
2. Gamal - Cardiologist
3. Omar - Pediatrician
Choose a doctor ID: 1
Enter Appointments Date (e.g. 2025-01-15): 2025-12-20
Available appointments generated successfully!
```

Figure 4: Generating available appointment slots for a selected doctor and date (30-minute intervals).



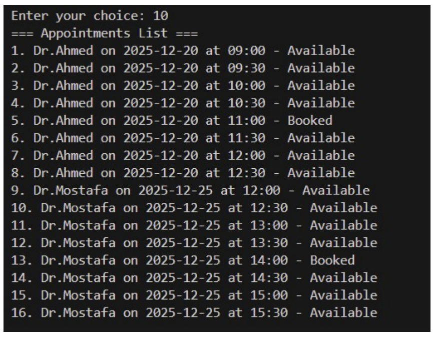
```
Enter your choice: 6
Available Appointments:
1. Dr.Ahmed | 2025-12-20 | 09:00
2. Dr.Ahmed | 2025-12-20 | 09:30
3. Dr.Ahmed | 2025-12-20 | 10:00
4. Dr.Ahmed | 2025-12-20 | 10:30
5. Dr.Ahmed | 2025-12-20 | 11:00
6. Dr.Ahmed | 2025-12-20 | 11:30
7. Dr.Ahmed | 2025-12-20 | 12:00
8. Dr.Ahmed | 2025-12-20 | 12:30
9. Dr.Mostafa | 2025-12-25 | 12:00
10. Dr.Mostafa | 2025-12-25 | 12:30
11. Dr.Mostafa | 2025-12-25 | 13:00
12. Dr.Mostafa | 2025-12-25 | 13:30
13. Dr.Mostafa | 2025-12-25 | 14:00
14. Dr.Mostafa | 2025-12-25 | 14:30
15. Dr.Mostafa | 2025-12-25 | 15:00
16. Dr.Mostafa | 2025-12-25 | 15:30
Enter Appointment ID to book: 5
Available Patients:
1. Ahmed
2. Ramy
3. Mohamed
Enter Patient ID: 1
Appointment booked successfully!
```

Figure 5: Booking an appointment by appointment ID after listing available slots and selecting a patient.



```
=== Main Menu ===
1. Add Doctor
2. Add Patient
3. Edit Doctor
4. Edit Patient
5. Generate Available Appointments
6. Book Appointment
7. Cancel Appointment
8. View Doctors
9. View Patients
10. View All Appointments
11. View Booked Appointments for Doctor
12. View Available Appointments for Doctor
13. Exit
Enter your choice: 7
Booked Appointments:
2. Dr.Ahmed | 2025-12-20 | 09:30
5. Dr.Ahmed | 2025-12-20 | 11:00
13. Dr.Mostafa | 2025-12-25 | 14:00
Enter Appointment ID to cancel: 2
Appointment cancelled successfully!
```

Figure 6: Canceling a booked appointment by appointment ID and confirming successful cancellation.



```
Enter your choice: 10
=== Appointments List ===
1. Dr.Ahmed on 2025-12-20 at 09:00 - Available
2. Dr.Ahmed on 2025-12-20 at 09:30 - Available
3. Dr.Ahmed on 2025-12-20 at 10:00 - Available
4. Dr.Ahmed on 2025-12-20 at 10:30 - Available
5. Dr.Ahmed on 2025-12-20 at 11:00 - Booked
6. Dr.Ahmed on 2025-12-20 at 11:30 - Available
7. Dr.Ahmed on 2025-12-20 at 12:00 - Available
8. Dr.Ahmed on 2025-12-20 at 12:30 - Available
9. Dr.Mostafa on 2025-12-25 at 12:00 - Available
10. Dr.Mostafa on 2025-12-25 at 12:30 - Available
11. Dr.Mostafa on 2025-12-25 at 13:00 - Available
12. Dr.Mostafa on 2025-12-25 at 13:30 - Available
13. Dr.Mostafa on 2025-12-25 at 14:00 - Booked
14. Dr.Mostafa on 2025-12-25 at 14:30 - Available
15. Dr.Mostafa on 2025-12-25 at 15:00 - Available
16. Dr.Mostafa on 2025-12-25 at 15:30 - Available
```

Figure 7: Appointments list after cancellation showing the canceled slot as available again.

5 Conclusion

This report detailed the design and implementation of the **Hospital Appointment System**, a robust solution that addresses the inefficiencies associated with manual hospital scheduling. By rigorously applying Object-Oriented Programming principles—specifically **Inheritance** through the abstract `Person` class, **Encapsulation** using private attributes and validated setter methods, and **Polymorphism** via method overriding—the system achieves a high level of modularity, maintainability, and data integrity.

The console-based interface successfully demonstrates core functionalities, including validated record creation, automated appointment slot generation, and reliable appointment booking and cancellation management. Overall, this project serves as a foundational proof of concept, presenting a clear object-oriented architecture that is suitable for future scalability and potential integration into a modern clinical environment.