Hospital Management System

A JAVA-BASED APPLICATION USING OOP AND JDBC

Group Members: **Behroz Ahmed** (2412107), **Humza Adnan Shah** (2412131), **Sameer Hayat** (2412127)

Class: **BSCS 2-A**

Submitted to: **Sir Muhammad Zaid**

Submission Date: **31-5-2025**

ABSTRACT

This report presents the design and implementation of a Hospital Management System, a console-based Java application developed to manage hospital operations, including patient and doctor records, appointments, billing, and user authentication. The system leverages Object-Oriented Programming (OOP)principles—encapsulation, inheritance, polymorphism, abstraction, and composition—and uses JDBC for data persistence with a MySQL database. The project comprises 15Java files and one SQL script, meeting the specified requirements. Key features include CRUD operations, secure login, and error handling. Challenges such as date-time parsing errors and Java version compatibility were resolved through targeted fixes. This reports de tails the system architecture, implementation, OOP principles applied, challenges, and set up instructions, providing a comprehensive overview for academic evaluation.

1) INTRODUCTION

The Hospital Management System is a console-based application designed to streamline hospital operations, such as managing patient and doctor information, scheduling appointments, generating bills, and authenticating users. Developed in Java, the system adheres to OOP principles and integrates with a MySQL database using JDBC. The project fulfills the requirement of including at least 15 Java files, organized into packages for modularity and maintainability.

1.1 Objectives

- Implement a system to manage patients, doctors, appointments, and bills.
- Provide secure user authentication.
- Enable search functionality for patients.
- Persist data using MySQL via JDBC.
- Apply OOP principles: encapsulation, inheritance, polymorphism, abstraction, and composition.
- Handle errors and validate user inputs effectively

2) SYSTEM ARCHITECTURE

The system follows a layered architecture, separating concerns into five packages:

• **model**: Defines entity classes (Person, Patient, Doctor, Appointment, User, Bill).

• **controller**: Manages business logic and database operations (PatientController, DoctorController, AppointmentController, BillingController, UserController, DatabaseConnection).

• **view**: Handles user interface and interaction (ConsoleView, MenuHandler).

• **exceptions**: Defines custom exceptions (InvalidInputException, DatabaseException)

• **Hospital**: Contains the entry point (Main.java).

2.1 File Structure

The project includes **15 Java files** and **one SQL script**:

1. Person.java (abstract base class)
2. Patient.java (extends Person)
3. Doctor.java (extends Person)
4. Appointment.java
5. User.java
6. Bill.java
7. DatabaseConnection.java

8. PatientController.java
9. DoctorController.java
10. AppointmentController.java
11. BillingController.java
12. UserController.java
13. ConsoleView.java
14. MenuHandler.java
15. InvalidInputException.java
16. DatabaseException.java
17. Main.java (main class)
18. database_setup.sql (database schema)

---

3) METHODOLOGY

The project was developed using an iterative approach:

1. **Requirements Analysis**: Identified core features (patient/doctor management, appointments, billing, authentication).
2. **Database Design**: Created a MySQL schema with tables for patients, doctors, appointments, users, and bills.
3. **OOP Design**: Applied encapsulation, inheritance, polymorphism, abstraction, and composition to model entities and interactions.
4. **Implementation**: Wrote Java code for model, controller, view, and exception handling, integrating with JDBC.

5. **Testing and Debugging**: Addressed issues like date-time parsing errors and Java version compatibility.

---

4) IMPLEMENTATION DETAILS

4.1 Features

- **Patient Management**: Add, view, and search patients by ID.
- **Doctor Management**: Add and view doctors.
- **Appointment Management**: Schedule and view appointments with date time validation.
- **Billing**: Generate and view patient bills.
- **Authentication**: Secure login with username/password.
- **Error Handling**: Custom exceptions for input validation and database errors.
- **Data Persistence**: MySQL database with JDBC for CRUD operations.

4.2 Database Schema

The MySQL database (hospital_db) includes:

- patients: (id, name, contact, address, medical_history)
- doctors: (id, name, contact, address, specialization)
- appointments: (id, patient_id, doctor_id, date_time)
- users: (id, username, password, role)
- bills: (id, patient_id, amount, description)

Foreign key constraints ensure referential integrity. The database_setup.sql script initializes the database with a default admin user(admin[username]/admin123[password]).

4.3 OOP Principles:

The project demonstrates all required OOP principles:

- **Encapsulation**: Private fields with public getters/setters in model classes (Person, Patient, Doctor, etc.) protect data integrity. Example: Patient's Medical History is accessed via getMedicalHistory ().
- **Inheritance**: Patient and Doctor extend the abstract Person class, inheriting attributes (id, name, contact, address) and the abstract getDetails () method.
- **Polymorphism**: getDetails () is overridden in Patient and Doctor, allowing ConsoleView to call it polymorphically. Example: displayPatients calls Patient.getDetails().
- **Abstraction**: The Person abstract class defines the getDetails () contract, hiding implementation details from ConsoleView.
- **Composition**: MenuHandler composes ConsoleView, PatientController, DoctorController, etc., to coordinate functionality. Appointment and Bill reference Patient and Doctor via IDs (logical composition).

Sample Code

Below is an example of the Person abstract class demonstrating abstraction and inheritance:

```
public abstract class Person {
    private int id;
    private String name;
    private String contact;
    private String address;

    public Person(int id, String name, String contact, String address) {
        this.id = id;
        this.name = name;
        this.contact = contact;
        this.address = address;
    }

    public abstract String getDetails();
    // Getters and setters omitted for brevity
}
```

---

5) CHALLENGES AND SOLUTIONS

Challenge 1: **DateTimeParseException**

Issue: The AppointmentController.getAllAppointments method threw a DateTimeParseException due to a mismatch between MySQL's DATETIME format (yyyy-MM-dd HH:mm:ss) and Java's expected LocalDateTime format (yyyy-MM-dd'T'HH:mm:ss).

Solution: Modified AppointmentController to use a DateTimeFormatter with pattern yyyy-MM-dd HH:mm:ss for parsing and formatting. Updated MenuHandler.handleScheduleAppointment to append :00 to user input for consistency.

Challenge 2: Java Release Version Error

Issue: Compilation failed with java: error: release version 24 not supported due to a mismatch between the project's target Java version (24) and the installed JDK.

Solution: Recommended configuring the project to use JDK 17 (Long-Term Support) or installing JDK 24. Provided steps to update JAVA_HOME and IntelliJ IDEA settings.

PREREQUISITES

- **Java**: JDK 17 (recommended) or 24.
- **MySQL**: MySQL Server 8.0 or later.
- **IDE**: IntelliJ IDEA.
- **MySQL JDBC Driver**: Version 8.0.33 or later.

RESULTS

The system successfully performs all intended functions:

- Users can log in securely.
- Patients and doctors can be added and viewed.
- Appointments can be scheduled and listed.
- Bills can be generated and retrieved by patient ID.
- Patient search by ID is supported.

Sample Outputs:

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 1
Enter username:
admin
Enter password:
admin123
Login successful!
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 2
Enter Patient ID:
5
Enter Name:
Sir Zaid
Enter Contact:
0472346394678
Enter Address:
Gulshan
Enter Medical History:
None
Patient added successfully.
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 3

=== Patient List ===
Patient ID: 5, Name: Sir Zaid, Contact: 0472346394678, Address: Gulshan, Medical History: None
Patient ID: 123, Name: Behroz, Contact: 0463746327846, Address: Gulshan, Medical History: None
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 4
Enter Patient ID:
5
Patient ID: 5, Name: Sir Zaid, Contact: 0472346394678, Address: Gulshan, Medical History: None
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 5
Enter Doctor ID:
10
Enter Name:
Sir Asim Riaz
Enter Contact:
03435235667
Enter Address:
AsimRiaz@gmail.com
Enter Specialization:
Bacho ko Darana
Doctor added successfully.
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 6

=== Doctor List ===
Doctor ID: 1, Name: sameer, Contact: 0243423523465, Address: gulshan, Specialization: mba
Doctor ID: 10, Name: Sir Asim Riaz, Contact: 03435235667, Address: AsimRiaz@gmail.com, Specialization: Bacho ko Darana
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 7
Enter Appointment ID:
11
Enter Patient ID:
5
Enter Doctor ID:
10
Enter Date and Time (yyyy-MM-dd HH:mm):
2025-05-05 11:09
Appointment scheduled successfully.
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 8

=== Appointment List ===
Appointment ID: 2, Patient ID: 123, Doctor ID: 1, Date: 2025-09-09T11:11
Appointment ID: 11, Patient ID: 5, Doctor ID: 10, Date: 2025-05-05T11:09
```

```
=== Hospital Management System ==
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 9
Enter Bill ID:
100
Enter Patient ID:
5
Enter Amount:
50
Enter Description:
regular checkup
Bill generated successfully.
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 10
Enter Patient ID:
5

=== Bill List ===
Bill ID: 100, Patient ID: 5, Amount: $50.0, Description: regular checkup
```

```
=== Hospital Management System ===
1. Login
2. Add Patient
3. View Patients
4. Search Patient
5. Add Doctor
6. View Doctors
7. Schedule Appointment
8. View Appointments
9. Generate Bill
10. View Bills
11. Exit
Choose an option: 11
Exiting system...
```

Limitations and Future Work

Limitations

- Console-based UI limits user experience.
- Passwords stored in plain text (security risk).
- Limited validation (e.g., no checks for duplicate IDs).
- No update or delete operations for entities.

Future Enhancements

- Implement a GUI using JavaFX or Swing.
- Add password hashing for secure authentication.
- Enhance input validation (e.g., unique IDs, format checks).
- Support update and delete operations.
- Introduce a service layer for complex business logic.

Conclusion

The Hospital Management System is a robust Java application that meets all project requirements, including the use of 15 Java files, OOP principles, and JDBC integration. It provides essential hospital management functionality with a modular, extensible design. Challenges were addressed through targeted fixes, ensuring reliability. The system serves as a strong foundation for further enhancements, demonstrating proficiency in Java programming and OOP concepts.