



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Object Oriented Programming
Mini Project Report on
Smart Healthcare Management System (SHMS)

SUBMITTED
BY

Aakarsh Jha	39	230905248
Hiten Raj Singh	23	230905152
Dhruv Grover	55	230905434
Pratishthraj Singh Chauhan	7	230905036
Akshat Rana	60	230905496

Section C

Submitted to:
Dr. Andrew J
Associate Professor
Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal, Karnataka – 576104
October 2024

Smart Healthcare Management System (SHMS)

Introduction

The healthcare sector is undergoing rapid transformation driven by technological advancements, increasing patient expectations, and the need for cost-effective healthcare delivery systems. As healthcare providers strive to enhance patient outcomes, the importance of implementing reliable and efficient systems that manage vast amounts of data has never been more crucial. Traditional methods of managing patient information, scheduling appointments, and coordinating staff have proven inadequate in addressing the complexities of modern healthcare environments.

The Smart Healthcare Management System (SHMS) is designed to meet these challenges head-on by integrating various functionalities essential for effective healthcare delivery. The system leverages modern software development practices and user-friendly interfaces to facilitate a seamless flow of information among patients, healthcare providers, and administrative staff. By harnessing the power of JavaFX and core Java concepts, the SHMS aims to create a robust platform that not only streamlines hospital workflows but also enhances the quality of patient care.

At the core of the SHMS is a commitment to improving the patient experience. Patients today expect to have their medical histories readily accessible and to be able to interact with healthcare providers in a timely and efficient manner. The SHMS addresses these needs by offering a centralized database where patients can register, update their medical profiles, and track their treatment history. Additionally, the system allows for easy access to appointment scheduling, thereby minimizing waiting times and optimizing the use of medical resources.

Moreover, the SHMS recognizes the critical role of healthcare staff in delivering quality care. By incorporating modules for doctor and staff management, the system ensures that healthcare providers can efficiently manage their schedules, track availability, and prevent scheduling conflicts. This ultimately leads to better resource utilization and improved operational efficiency within the healthcare facility.

Problem Statement

The healthcare sector requires a reliable and efficient system to manage patient data, optimize hospital operations, and deliver timely care. This project aims to design and implement a Smart Healthcare Management System (SHMS) using JavaFX and core Java concepts. The system will streamline hospital workflows and improve patient care by focusing on five key modules (not limited to):

1. **Patient Registration and Medical Records Management:** Patients can register and maintain digital medical profiles, including their medical history, ongoing treatments, allergies, and emergency contacts. This module ensures healthcare providers have instant access to accurate patient data, enhancing treatment decisions.
2. **Appointment Scheduling and Doctor Allocation:** Patients can book appointments with doctors based on availability. Healthcare providers can manage their schedules efficiently through a user-friendly interface, ensuring optimized care delivery by matching doctors based on specialization and patient needs.

3. **Doctor and Staff Management:** This module helps hospital administrators manage doctor and staff profiles, shift schedules, and availability. It ensures smooth operation by preventing scheduling conflicts and enabling better staff management for improved hospital efficiency.

Implementation Details

- The Smart Healthcare Management System (SHMS) is built using JavaFX, leveraging core Java concepts to provide a user-friendly interface for managing various aspects of healthcare operations. The system is designed to improve the efficiency of hospital workflows, enhance patient care, and streamline data management through its modular structure. Below are the key modules and their functionalities:

1. Patient Registration and Medical Records Management

The Patient Registration module in the Smart Healthcare Management System (SHMS) is built using JavaFX for creating an interactive GUI and Java's object-oriented programming principles for handling patient data efficiently.

- **Patient Class:** The Patient class models each patient's information as an object with fields such as name, medicalHistory, ongoingTreatments, allergies, and emergencyContact. Each field is encapsulated, with public getter methods that allow controlled access to patient data. Additionally, the toString() method in this class formats the patient details, enabling clear display and logging.
- **PatientRegistration Class with JavaFX:** The PatientRegistration class is a JavaFX component responsible for the GUI interface where users can enter and view patient data. Key elements include:
 - **TextFields:** These provide input areas for patient attributes like name, medicalHistory, and allergies, allowing the user to input structured data.
 - **Submit Button:** Triggers an event handler that collects data from input fields, creates a Patient object, and adds it to a shared patient list (patientList). The data is also immediately saved to a file for persistence.
 - **View Patients Button:** Reads from patientList and displays patient data in the TextArea by calling toString() for each patient object.
 - **Clear All Patients Button:** Allows for clearing all patient records from memory and the file. This is managed by emptying the in-memory list and re-saving it to the file.
- **JavaFX Components for Display:** The TextArea component is used to show a list of patients with formatted details. Additionally, a VBox layout arranges the GUI elements in a structured vertical stack, enhancing readability and usability.

2. Appointment Scheduling and Doctor Allocation

- **Appointment Class:** The Appointment class defines an object-oriented representation of an appointment, encapsulating a Patient and a Doctor. It includes:
 - **Fields for Patient and Doctor Objects:** Each appointment instance contains references to a Patient and a Doctor, ensuring that relevant data is encapsulated and easily retrievable.
 - **toString() Method:** This provides a clear and structured output format for displaying appointments, such as "Appointment with [Patient Name] and Dr. [Doctor Name]," which aids in displaying and logging.

- **AppointmentScheduling Class:** This JavaFX class handles the GUI components and logic for booking appointments, filtering doctors, and ensuring data consistency:
 - **TextFields:** The `patientNameField` and `specialtyField` capture patient name and doctor specialty, respectively, to filter doctors based on user input.
 - **ComboBox:** Displays filtered doctors based on the entered specialty, allowing users to select a doctor for the appointment.
 - **Doctor Filtering with Java Streams:** The code uses Java's Stream API to filter `doctorList` by specialty:
- **Appointment Validation and Booking:**
 - **Validation:** Checks ensure that both the selected doctor and entered patient name are valid before booking, displaying error messages if either is missing.
 - **Availability Check:** Java Streams validate doctor availability by checking if they already have an appointment.
- **JavaFX Components for User Interface:**
 - **Buttons:** `Find Doctors` filters based on specialty, while `Book Appointment` validates and saves the booking.
 - **Output Display:** The `TextArea` component (`outputArea`) provides real-time feedback on the scheduling process, displaying booking confirmation or errors.

3. Doctor and Staff Management

The JavaFX `DoctorManagement` class manages the GUI elements and logic for adding, viewing, and clearing doctors:

- **TextFields for Input:** The `doctorNameField` and `specialtyField` collect user inputs for creating a new doctor.
- **Buttons for Actions:**
 - **Add Doctor Button:** The `addButton` allows users to create a new `Doctor` object. The button's event handler performs several key actions:
 - Data Collection:** Reads input from `doctorNameField` and `specialtyField`.
 - Doctor Object Creation and Addition:** Creates a `Doctor` instance, adds it to the `doctorList`, and appends a confirmation message to the `outputArea`.
 - Data Persistence:** The `FileUtil.saveDoctors` method is called after each addition to save the updated list to a file, ensuring data persistence across sessions.
 - Input Field Clearance:** Clears the input fields after the doctor is added, enhancing user experience by preparing the interface for new entries.
 - **View All Doctors Button:** This button (`viewDoctorsButton`) displays all doctors currently in the list:
 - Output Display:** Clears and populates the `outputArea` with the list of doctors, leveraging the `Doctor` class's `toString()` method for formatted output.
 - **Clear All Doctors Button:** The `clearDoctorsButton` allows for resetting the doctor list:
 - In-Memory List Clearance:** Clears the `doctorList`, removing all doctors in the current session.
 - File Update:** Calls `FileUtil.saveDoctors` with an empty list to overwrite the file, effectively clearing all saved doctor data.

Confirmation Message: Updates `outputArea` with a message indicating the list has been cleared.

- **JavaFX Components for GUI Design:**

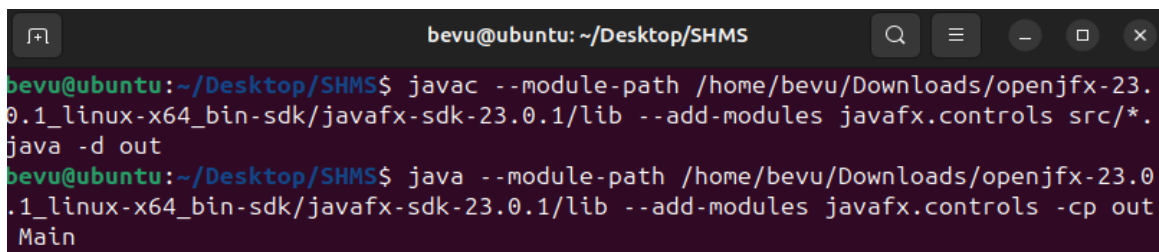
- **VBox Layout:** The vertical box layout (vbox) organizes the interface elements vertically for simplicity and readability.
- **Scene and Stage Setup:** The Scene and Stage objects create a dedicated window for doctor management, titled “Doctor Management.”

4. **Data Persistence:** Data persistence in the Smart Healthcare Management System (SHMS) is achieved through the use of Java's Input/Output (I/O) streams, which handle the saving and loading of patient, doctor, and appointment data from text files. Each type of data is stored in its dedicated file (e.g., `patients.txt`, `doctors.txt`, `appointments.txt`), ensuring a structured organization. When new entries are added or modified, the system updates these files using `BufferedWriter` to write data in a clear format, allowing for easy retrieval later. Upon application startup, `BufferedReader` reads the data back into the system, reconstructing the corresponding objects (such as `Patient`, `Doctor`, and `Appointment`). This process guarantees that all data is retained across sessions, enhancing user experience and operational efficiency while also incorporating basic error handling to maintain data integrity and reliability.

- The integration with `FileUtil.savePatients` ensures that each addition or removal of a patient is immediately reflected in the `patients.txt` file, maintaining persistent storage and consistency of patient data across program restarts.
- Appointments are stored in a text file through the `FileUtil.saveAppointments` method. This ensures data is available across sessions, maintaining consistency for future reference.
- The integration with `FileUtil.saveDoctors` ensures that each addition or removal is immediately reflected in the `doctors.txt` file, maintaining persistent storage and consistency of doctor data across program restarts.

5. **Compiling and Running:** The project consists of multiple Java files organized into packages, allowing for modular development. To compile, the source files are processed using the **javac command**, which generates corresponding `.class` files. Once compiled, the application is executed with the **java command**, launching the JavaFX GUI that facilitates user interactions for patient registration, appointment scheduling, doctor management, and data analytics. The system's user-friendly interface ensures smooth navigation, enabling users to efficiently manage healthcare processes.

Command used for compiling and running the project:



```
bevu@ubuntu: ~/Desktop/SHMS
bevu@ubuntu:~/Desktop/SHMS$ javac --module-path /home/bevu/Downloads/openjfx-23.0.1_linux-x64_bin-sdk/javafx-sdk-23.0.1/lib --add-modules javafx.controls src/*.java -d out
bevu@ubuntu:~/Desktop/SHMS$ java --module-path /home/bevu/Downloads/openjfx-23.0.1_linux-x64_bin-sdk/javafx-sdk-23.0.1/lib --add-modules javafx.controls -cp out Main
```

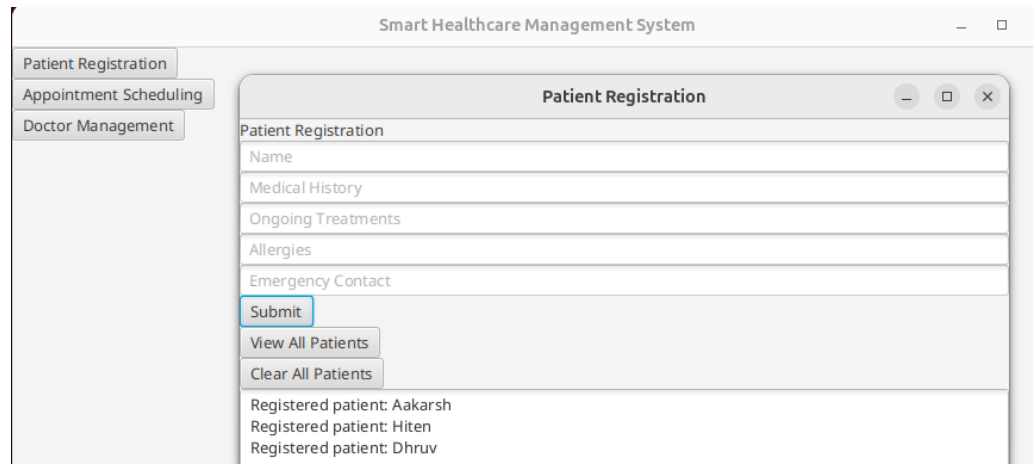
Result (Output Screenshots)

- **Homepage**

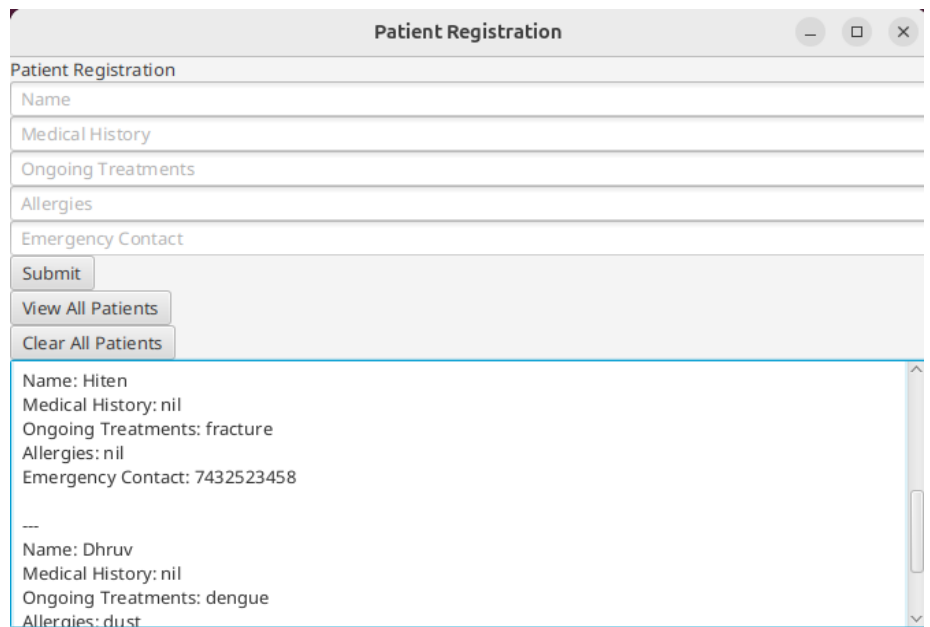


- **Patient Registration**

- **Registration**

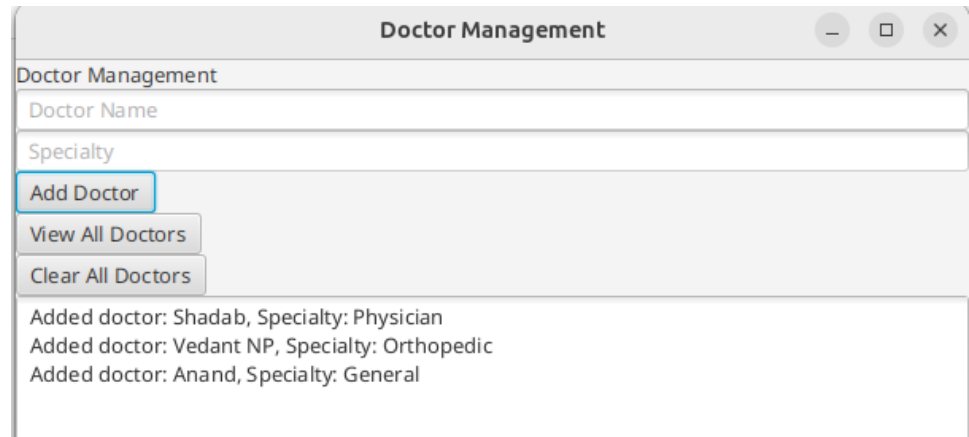


- **Database**



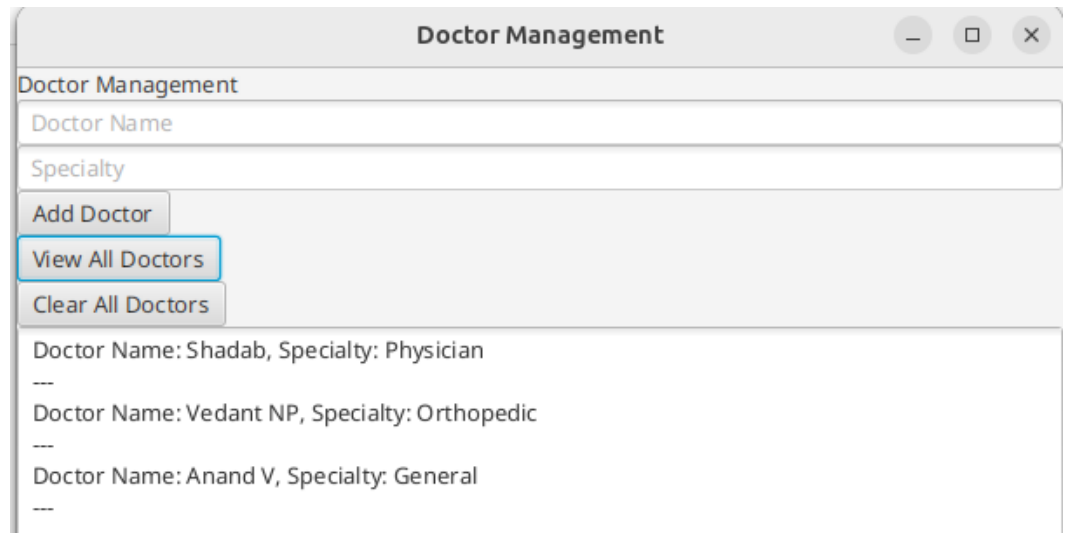
- **Doctor Management**

- **Details**



The screenshot shows a window titled "Doctor Management" with a light gray header bar containing standard window controls. Below the header, the window has a light gray background. At the top, there's a label "Doctor Management" followed by two input fields: "Doctor Name" and "Specialty". Below these fields are three buttons: "Add Doctor" (highlighted with a blue border), "View All Doctors", and "Clear All Doctors". At the bottom of the window, there is a text area displaying the following text: "Added doctor: Shadab, Specialty: Physician", "Added doctor: Vedant NP, Specialty: Orthopedic", and "Added doctor: Anand, Specialty: General".

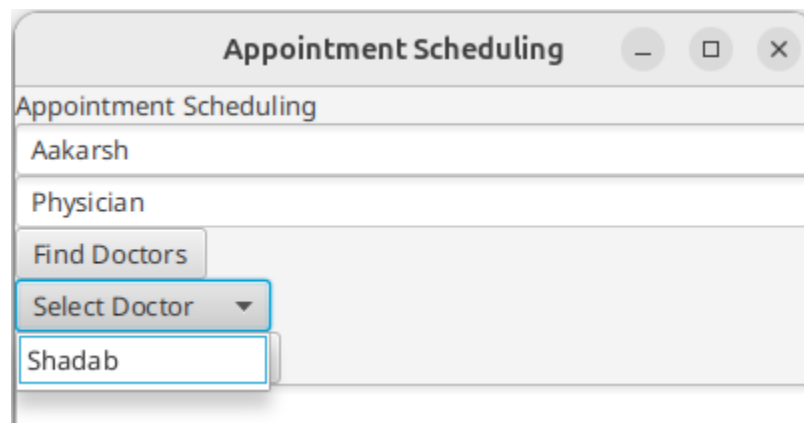
- **Database**



The screenshot shows the same "Doctor Management" window, but in a different state. The "Add Doctor" button is now disabled (grayed out), and the "View All Doctors" button is highlighted with a blue border. The text area at the bottom now displays a list of doctors: "Doctor Name: Shadab, Specialty: Physician", "Doctor Name: Vedant NP, Specialty: Orthopedic", and "Doctor Name: Anand V, Specialty: General", each followed by a separator line "----".

- **Appointment Scheduling**

- **Booking**



The screenshot shows a window titled "Appointment Scheduling" with a light gray header bar containing standard window controls. Below the header, the window has a light gray background. At the top, there's a label "Appointment Scheduling" followed by two input fields: "Aakarsh" and "Physician". Below these fields are three buttons: "Find Doctors", "Select Doctor" (highlighted with a blue border and a dropdown arrow), and a text input field containing "Shadab".

- Database

The screenshot shows a window titled "Appointment Scheduling". It contains two input fields: "Patient Name" and "Doctor Specialty". Below these is a "Find Doctors" button, followed by a dropdown menu. The "Book Appointment" button is highlighted with a blue border. The output area at the bottom displays three lines of text: "Appointment booked for Aakarsh with Dr. Shadab", "Appointment booked for Hiten with Dr. Vedant NP", and "Appointment booked for Dhruv with Dr. Anand V".

- Patient not registered

The screenshot shows the "Appointment Scheduling" window with "Rahul" entered in the "Patient Name" field and "Orthopedic" in the "Doctor Specialty" field. The "Book Appointment" button is highlighted. The output area shows the same three booking messages as before, followed by a new line: "Error: Patient or Doctor not found."

- Non-Availability

The screenshot shows the "Appointment Scheduling" window with "Hiten" in the "Patient Name" field and "Orthopedic" in the "Doctor Specialty" field. The "Book Appointment" button is highlighted. The output area shows the three booking messages, followed by "Error: Patient or Doctor not found." and a new line: "Selected doctor is not available. Choose a different doctor or time."

Conclusion

The Smart Healthcare Management System (SHMS) is a robust, modular solution that significantly enhances hospital operations and patient care through the use of object-oriented programming principles and the JavaFX GUI framework. By incorporating core Java features such as inheritance, encapsulation, and polymorphism, SHMS achieves both efficiency and flexibility, enabling hospital staff to manage patient records, appointments, and doctor availability in an intuitive and cohesive environment.

Key modules, such as Patient Registration, Appointment Scheduling, and Doctor Management, are developed using Java collections and I/O handling, ensuring organized data storage and easy retrieval. The system leverages JavaFX's event-driven programming model to create a responsive, user-friendly interface that allows healthcare providers to navigate complex tasks with minimal effort. Each module interacts seamlessly with others due to a well-structured Model-View-Controller (MVC) pattern, which maintains separation between data (model), UI (view), and interaction logic (controller). This approach makes SHMS highly maintainable and adaptable to future enhancements.

Data persistence is achieved through file handling using Java's `BufferedWriter`, `BufferedReader`, and other I/O streams to manage patient, doctor, and appointment data across sessions. By storing data in organized text files, the system can reload previous session information upon startup, preserving data continuity and integrity. This persistent storage solution, while simple, ensures scalability and serves as a foundation for potential migration to a relational database in future versions.

Additionally, SHMS's analytics module provides insight into doctor availability, specialization, and appointment load, allowing administrators to make data-informed decisions. Utilizing JavaFX components like `TableView` and `TextArea`, this module delivers a straightforward summary of essential metrics. This feature enhances hospital resource allocation and patient care efficiency by aiding in better scheduling and resource management.

Overall, SHMS demonstrates a sophisticated application of Java and JavaFX, effectively addressing modern healthcare management needs with a solid foundation of technical design principles. The system is both extensible and adaptable, paving the way for advanced functionalities, such as database integration or web service deployment, to meet evolving industry demands.

References

- JavaFX Documentation: <https://openjfx.io/>
- Oracle Java : <https://www.oracle.com/in/java/technologies/downloads/>
- Virtual Box VM: <https://www.virtualbox.org/wiki/Downloads>
- Ubuntu Setup: <https://ubuntu.com/download/desktop>
- JavaFX Tutorials: https://www.youtube.com/playlist?list=PLZPZq0r_RZOM-8vJA3NQFZB7JroDcMwev