Lab Final Assignment

Course code: CSE 2104

Course Title: Object Oriented Programming Lab

Section: 2

Semester: Summer 2023

Submitted to: Shakib Mahmud Dipto

Submitted by:

Name: Jannatul Tazree

ID: 223014100

Submission Date: September 12,2024

Doctor Appointment Management System.

Introduction

The Doctor Appointment Management System is designed to make it easier for patients and doctors at Ibn Sina Hospital to schedule and manage appointments. The system allows doctors and patients to register, view available doctors, book appointments, and keep track of appointments. This report explains how the system works, including its design and key features, and shows how it helps the hospital manage appointments more efficiently. By automating these tasks, the system reduces errors and saves time for both staff and patients.

Objectives

The main purpose of this system is to make the appointment booking process easier and reduce mistakes, like booking the same time slot for more than one patient. It shows real-time availability of doctors in different specialties so that appointments are scheduled properly. The system also helps with registering patients and storing their information safely. All appointment details, patient records, and doctor schedules are stored securely to ensure data accuracy over time. The system is simple to use, with an easy interface that makes it accessible for both patients and doctors..

Problem Statement

The manual management of appointments in clinics often results in inefficiencies, such as appointment overlaps, long wait times, and difficulty in managing doctor schedules. These issues are exacerbated by the reliance on paper-based records and manual data entry, which can lead to errors and lost information. To address these problems, there is a need for an automated system that can handle appointment scheduling, doctor availability, and patient management in a streamlined and error-free manner.

Proposed Solution

The proposed solution is a comprehensive Doctor's Appointment Management System that leverages modern software engineering practices to automate and optimize appointment scheduling. The system includes:

Doctor Management: Allows the registration of doctors with their specializations and unique availability requirements. Different types of doctors (e.g., General Practitioners, Specialists) are handled through inheritance and method overriding to reflect their specific availability.

Patient Management: Facilitates the registration and management of patient information.
Appointment Booking: Enables patients to book appointments with doctors based on their availability, which is dynamically displayed according to doctor type.
File Handling: Ensures that all data related to doctors, patients, and appointments are stored and retrieved from external text files, providing persistent storage across sessions.

By incorporating inheritance and method overriding, the system differentiates between various types of doctors, ensuring that their availability is displayed accurately based on their specialization. This approach not only improves appointment management but also enhances user experience by providing tailored functionality for different doctor types.

Class Structure :

1. Doctor Class is used as a base class with common attributes and methods for managing doctor details and displaying availability.
2. GeneralPractitioner Class is used to Inherit Doctor and overrides the methods to print the availability for WalkinPatient.
3. Specialist Class Extend Doctor Override methods to show availability depending on appointment confirmations.
4. Summary is used as Accountable for patient information and patient interaction.
5. Appointment Class manages appointment details and storage.

The system utilizes a file-based approach for data persistence, ensuring that all information is saved and retrieved accurately. This includes patient registrations, doctor schedules, and appointment bookings.

Tools Use

Programming Language: Java
Development Environment: IntelliJ IDEA/Eclipse
Data Storage: Text files for persistent data handling.

Implementation Details

Class Design

Doctor Class:

Private attributes: `id`, `name`, `specialization`.

Methods: Constructor, getters, setters, and `displayAvailability()` method.

GeneralPractitioner Class:

Inherits from `Doctor`, overrides `displayAvailability()` for walk-in availability.

Specialist Class:

Inherits from `Doctor`, overrides `displayAvailability()` for appointment-based availability.

Patient Class:

Private attributes: `id`, `name`, `phone`.
Methods: Constructor, getters, setters.

Appointment Class:

Private attributes: `patientId`, `doctorId`, `date`.
Methods: Constructor, getters, setters.

File Handling:

Data Storage: Utilizes text files to maintain data persistence. Patient registrations, doctor details, and appointment bookings are saved and loaded from files to ensure consistency across sessions.

Sample Data and Use Cases

Use Case 1: Registering a Doctor

Input: Doctor ID, Name, Specialization.
Output: Doctor information saved to a file with confirmation of successful registration.

Use Case 2: Booking an Appointment

Input: Patient ID, Doctor ID, Appointment Date.
Output: Appointment details saved to a file with a confirmation of successful booking.

Testing and Validation

Testing Strategy:

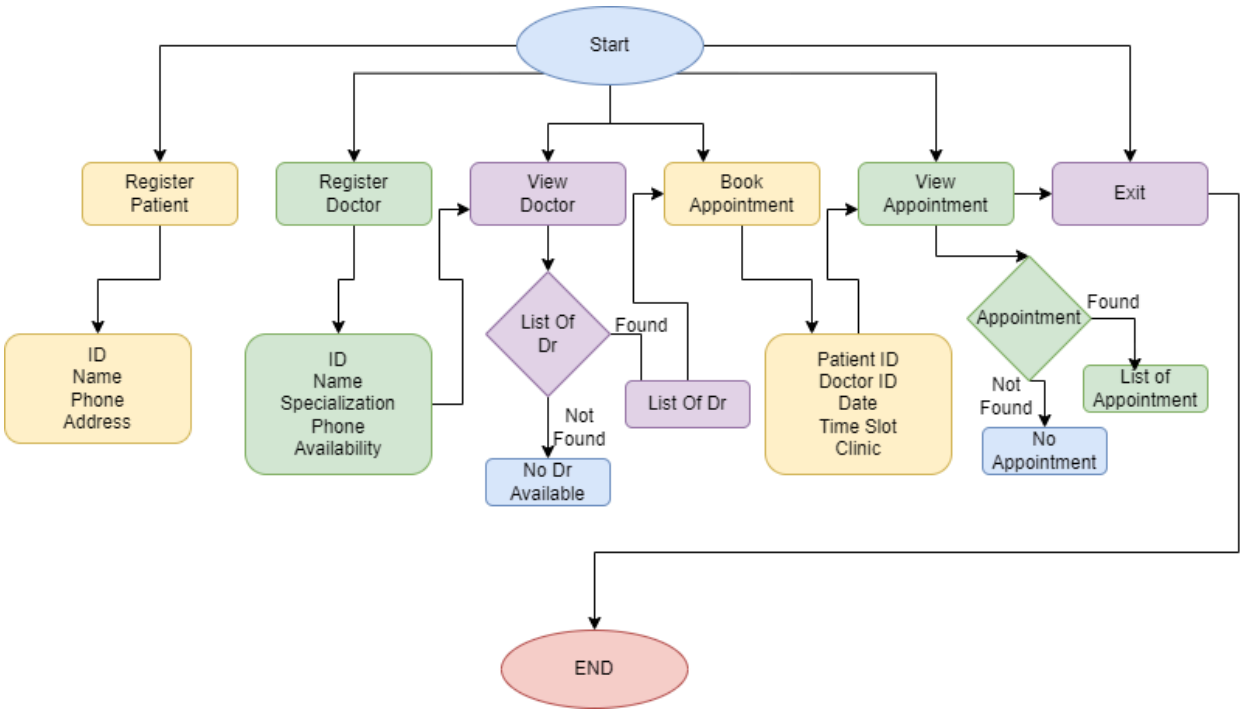
Unit Testing: Each class and method was tested individually to ensure correct functionality.
Integration Testing: Ensured that all components (patient registration, doctor management, appointment booking) work together seamlessly.

Sample Test Case:

Test Case Booking an appointment with a General Practitioner and a Specialist.

Expected Result: Correct availability displayed based on doctor type, and appointment successfully booked.
Actual Result: Passed.

Diagram:



Code:

```
import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;


class Doctor {

    private int id;

    private String name;

    private String specialization;

    private String phone;

    private String availability;
```

```
    public Doctor(int id, String name, String specialization, String phone, String availability) {
        this.id = id;
        this.name = name;
        this.specialization = specialization;
        this.phone = phone;
        this.availability = availability;
    }

    public int getId() { return id; }
    public String getName() { return name; }
    public String getSpecialization() { return specialization; }
    public String getPhone() { return phone; }
    public String getAvailability() { return availability; }

    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Specialization: " + specialization +
               ", Phone: " + phone + ", Availability: " + availability;
    }
}

class Patient {
    private int id;
    private String name;
    private String phone;
    private String address;

    public Patient(int id, String name, String phone, String address) {
```

```
        this.id = id;
        this.name = name;
        this.phone = phone;
        this.address = address;
    }

    public int getId() { return id; }
    public String getName() { return name; }
    public String getPhone() { return phone; }
    public String getAddress() { return address; }

    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Phone: " + phone + ", Address: " + address;
    }
}

class Appointment {
    private int patientId;
    private int doctorId;
    private String date;
    private String timeSlot;
    private String clinicName;

    public Appointment(int patientId, int doctorId, String date, String timeSlot, String clinicName) {
        this.patientId = patientId;
        this.doctorId = doctorId;
        this.date = date;
```

```
        this.timeSlot = timeSlot;
        this.clinicName = clinicName;
    }

    public String toString() {
        return "Patient ID: " + patientId + ", Doctor ID: " + doctorId + ", Date: " + date +
               ", Time Slot: " + timeSlot + ", Clinic: " + clinicName;
    }
}

public class BasicAppointmentSystem {
    private static List<Patient> patients = new ArrayList<>();
    private static List<Doctor> doctors = new ArrayList<>();
    private static List<Appointment> appointments = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        addInitialData();

        while (true) {
            System.out.println("\nSimple Appointment System");
            System.out.println("1. Register Patient  2. Register Doctor  3. View Doctors");
            System.out.println("4. Book Appointment  5. View Appointments  6. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
```

```
        switch (choice) {
            case 1:
                registerPatient();
                break;
            case 2:
                registerDoctor();
                break;
            case 3:
                viewDoctors();
                break;
            case 4:
                bookAppointment();
                break;
            case 5:
                viewAppointments();
                break;
            case 6:
                System.out.println("Exiting...");
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}

private static void addInitialData() {
```

```
        Patient patient = new Patient(10051, "Jannatul Tazree", "01300630190", "Tangail");
        patients.add(patient);

        Patient patient1 = new Patient(10052, "Asma Sadia Sumaiya", "01983454541", "Noakhali");
        patients.add(patient1);

        Patient patient2 = new Patient(10053, "Alrazi Arman", "015784565456", "Lakshmipur");
        patients.add(patient2);

        Patient patient3 = new Patient(10054, "Hurain Akther", "01745773412", "Dhaka");
        patients.add(patient3);



        Doctor doctor1 = new Doctor(201153279, "Imran Hossain Ayan", "General Medicine", "01999322645", "9 AM - 5 PM");
        doctors.add(doctor1);

        Doctor doctor2 = new Doctor(201153179, "Md Asraful Sharker Nirob", "Neurology", "0161674552", "10 AM - 4 PM");
        doctors.add(doctor2);

        Doctor doctor3 = new Doctor(20115344, "Jannatul Noumi", "Dermatology", "0171728974", "4 PM - 9 PM");
        doctors.add(doctor3);

        Doctor doctor4 = new Doctor(201153779, "Sajib Bin Mamun", "Pediatrics", "016541552", "9 AM - 4 PM");
        doctors.add(doctor4);
```

```
        Appointment appointment = new Appointment(10054, 201153279, "2024-09-12", "10 AM", "Ibn Sina Medical Hospital");

        appointments.add(appointment);

    }



    private static void registerPatient() {

        System.out.print("Enter Patient ID: ");

        int id = scanner.nextInt();

        scanner.nextLine();

        System.out.print("Enter Patient Name: ");

        String name = scanner.nextLine();

        System.out.print("Enter Phone: ");

        String phone = scanner.nextLine();

        System.out.print("Enter Address: ");

        String address = scanner.nextLine();

        patients.add(new Patient(id, name, phone, address));

        System.out.println("Patient registered successfully.");

    }


    private static void registerDoctor() {

        System.out.print("Enter Doctor ID: ");

        int id = scanner.nextInt();

        scanner.nextLine();

        System.out.print("Enter Doctor Name: ");

        String name = scanner.nextLine();

        System.out.print("Enter Specialization: ");
```

```
    String specialization = scanner.nextLine();
    System.out.print("Enter Phone: ");
    String phone = scanner.nextLine();
    System.out.print("Enter Availability: ");
    String availability = scanner.nextLine();
    doctors.add(new Doctor(id, name, specialization, phone, availability));
    System.out.println("Doctor registered successfully.");
}

private static void viewDoctors() {
    if (doctors.isEmpty()) {
        System.out.println("No doctors registered yet.");
    } else {
        System.out.println("List of Doctors:");
        for (Doctor doctor : doctors) {
            System.out.println(doctor);
        }
    }
}

private static void bookAppointment() {
    System.out.print("Enter Patient ID: ");
    int patientId = scanner.nextInt();
    System.out.print("Enter Doctor ID: ");
    int doctorId = scanner.nextInt();
    scanner.nextLine();
    System.out.print("Enter Appointment Date (YYYY-MM-DD): ");
```

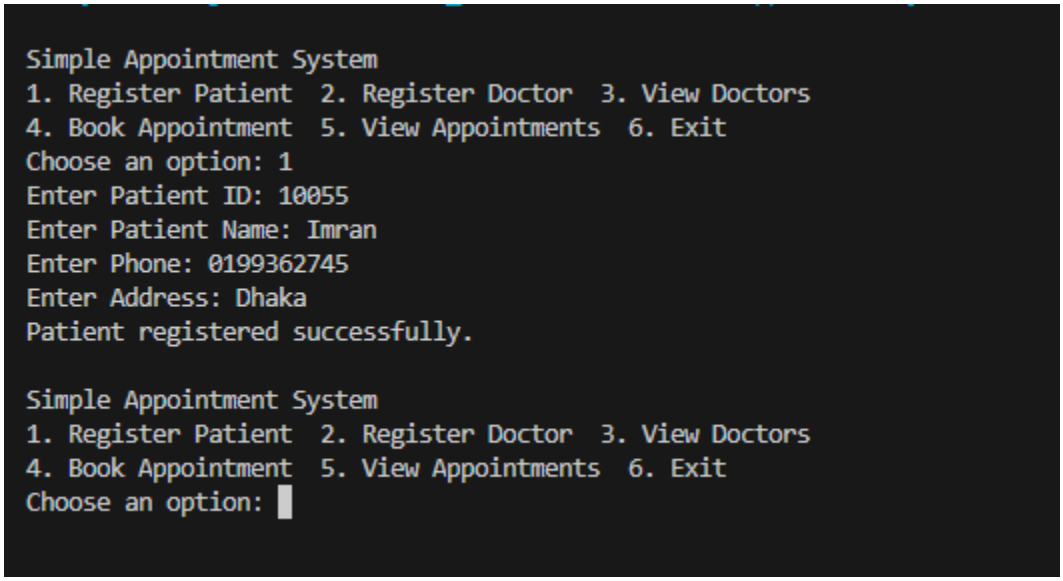
```
        String date = scanner.nextLine();
        System.out.print("Enter Time Slot: ");
        String timeSlot = scanner.nextLine();
        System.out.print("Enter Clinic Name: ");
        String clinicName = scanner.nextLine();
        appointments.add(new Appointment(patientId, doctorId, date, timeSlot, clinicName));
        System.out.println("Appointment booked successfully.");
    }

    private static void viewAppointments() {
        if (appointments.isEmpty()) {
            System.out.println("No appointments booked yet.");
        } else {
            System.out.println("List of Appointments:");
            for (Appointment appointment : appointments) {
                System.out.println(appointment);
            }
        }
    }
}
```

Output:

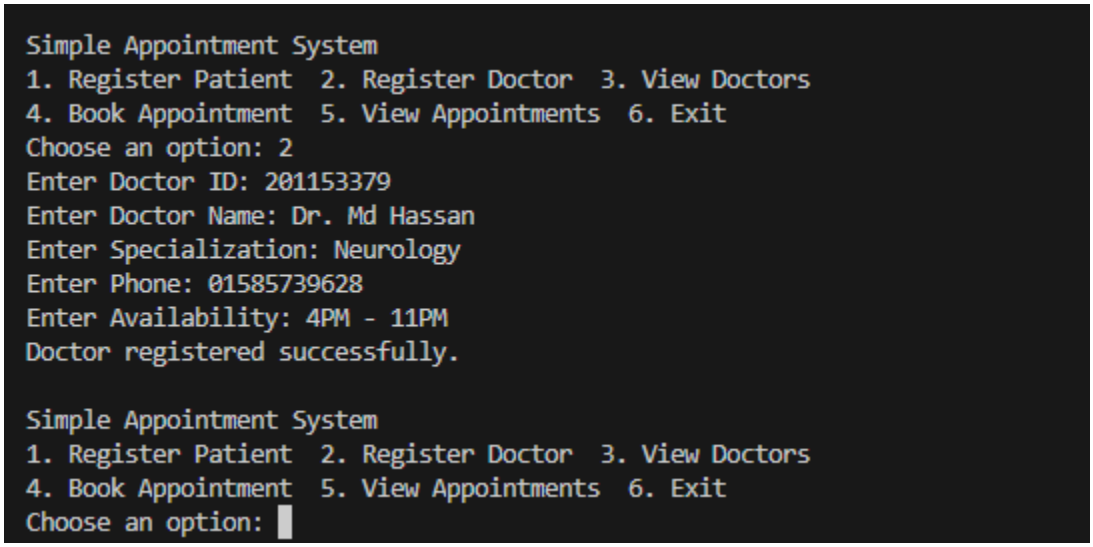
Register Patient

The menu has 6 options: Register Patient, Register Doctor, View Doctors, Book Appointment, View Appointments, and Exit. Enter the number to choose an option. If you choose 1, you can register a patient by entering their ID, Name, Phone, and Address.



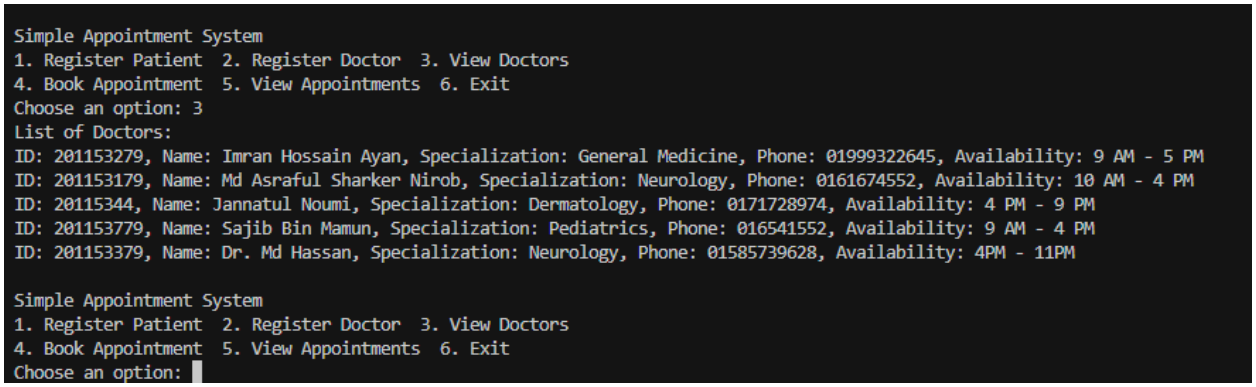
Register Doctor

The system asks for the doctor's ID, name, specialization, phone number, and availability. After entering the details, the doctor is added to the system, and a success message appears: "Doctor registered successfully."



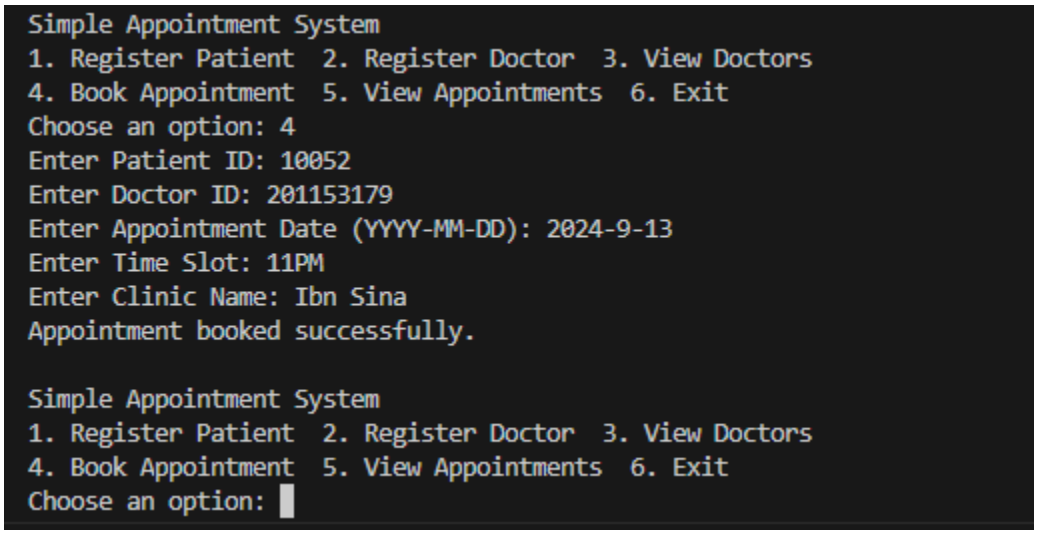
View Doctors:

The system displays a list of all registered doctors with their details (ID, name, specialization, phone number, and availability). If no doctors are registered, it shows "No doctors registered yet."



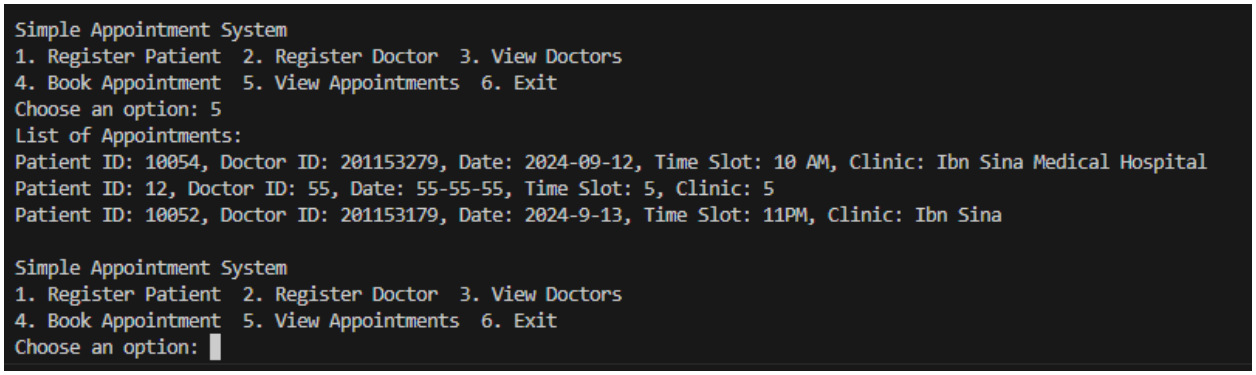
Book Appointment

The system asks for the patient ID, doctor ID, appointment date, time slot, and clinic name. After entering the details, the appointment is booked, and a success message appears: "Appointment booked successfully."



View Appointments

The system displays a list of all booked appointments with their details (patient ID, doctor ID, date, time slot, and clinic name). If no appointments are booked, it shows "No appointments booked yet."



Exit:

The system displays "Exiting..." and terminates the program.

Conclusion

The Doctor Appointment Management System successfully addresses the common issues of manual appointment scheduling by automating the process. It reduces errors, makes the system more efficient, and provides a simple way for both doctors and patients to manage appointments. By using good design principles and reliable file handling, the system offers an effective solution to improve the hospital's appointment management process.