**Declaration of Original Work for SC2002/CE2002/CZ2002 Assignment**

We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

| Name | Course  (SC2002/CE2002 CZ2002) | Lab  Group 1 | Signature /Date  21/11/2024 |
| --- | --- | --- | --- |
| Tang Yeqing | SC2002 | SCSD |  |
| Pahwa Ronak | SC2002 | SCSD |  |
| Verma Shireen | SC2002 | SCSD |  |
| Ramakrishna Rohan | SC2002 | SCSD |  |

Important notes:

## **1.** **Name must EXACTLY MATCH the one printed on your Matriculation Card.**

## **2.** **Student Code of Academic Conduct includes the latest guidelines on usage of Generative AI and any other guidelines as released by NTU.**

SC2002 HMS Assignment Report

## **1. Introduction**

The **Hospital Management System (HMS)** is a command-line application designed to streamline hospital operations by incorporating features that address the needs of patients, doctors, pharmacists, and administrators. Built using the principles of Object-Oriented Programming (OOP), this system encapsulates hospital workflows into modular and maintainable components.

### **Objective**

The main goal of HMS is to enhance hospital efficiency by automating routine tasks such as patient management, appointment scheduling, staff administration, and inventory control. By centralizing these functionalities, the system provides a unified platform to manage hospital operations seamlessly.

## **2. Design Considerations**

The **Hospital Management System (HMS)** leverages Object-Oriented Programming (OOP) principles to create a modular, scalable, and maintainable architecture. The design considerations focus on addressing the functional requirements of different user roles (patients, doctors, pharmacists, and administrators) while ensuring seamless interaction with system components. The following subsections outline the key aspects of HMS’s design.

### **2.1 Object-Oriented Design**

### HMS is structured using OOP principles, ensuring modularity and reusability. Key design elements include:

#### **a. Encapsulation**

* All core entities, such as patient, staff, Inventory, and MedicalRecord, encapsulate data and expose functionalities through well-defined methods.
* Sensitive attributes, like password in the user class, are accessible only through controlled mechanisms.

#### **b. Abstraction**

* Complex operations, such as managing appointments or inventory, are abstracted through service classes (AppointmentManager, PatientApptService, PharmApptService, etc.).
* User-specific menus (patientMenu, doctorMenu, etc.) abstract detailed workflows for each role, providing a simplified interface.

#### **c. Inheritance**

* While no explicit class inheritance is observed, roles such as staff and patient are treated as distinct entities with their own data and behaviors, showcasing role-specific specialization.

#### **d. Polymorphism**

* Different menus (administratorMenu, doctorMenu, etc.) demonstrate polymorphism by providing unique implementations of their DisplayMenu methods, while adhering to a common design.

### **2.2 Data Management**

HMS manages its data dynamically and persistently:

#### **a. Dynamic Data Handling**

* In-memory lists (patientList, staffList) store loaded data during runtime.
* Real-time modifications (e.g., adding new patients or updating inventory) are performed using these data structures.

#### **b. Persistent Storage**

* CSV files are used for persistent storage of patients, staff, and inventory.
* Classes like patientIN and staffIN handle data import/export, ensuring that updates during runtime are reflected in persistent files.

### **2.3 Validation and Error Handling**

Robust validation and error handling mechanisms are integrated to ensure data integrity and system reliability:

* **Input Validation:** Methods like updateContactInfo in the patient class validate email and phone formats using regular expressions.

Assumptions - patient can only add an **email** or a **Singaporean number (8-digit number starting with 8 or 9)**

* **Appointment Slot Constraints:** Appointment slots have the following constraints based on our assumptions:
* Hospital working hours: 9am to 6 pm
* Closed on Sundays
* Appointments can only be set 1 week in advance
* Appointments for the past cannot be set

*The screenshots of these have been provided under 5. Test cases*

* **Error Handling:** File I/O operations (e.g., loading CSV data in patientIN) handle exceptions to avoid runtime crashes.

### 

### **2.4 User Interface and Experience**

HMS uses a command-line interface (CLI) for user interaction. The design ensures:

* **Clarity:** Menus and options are clearly displayed for each role.
* **Ease of Use:** Step-by-step prompts guide users through workflows.
* **Error Messages:** Informative messages are provided for invalid inputs or failed operations, enhancing user experience.

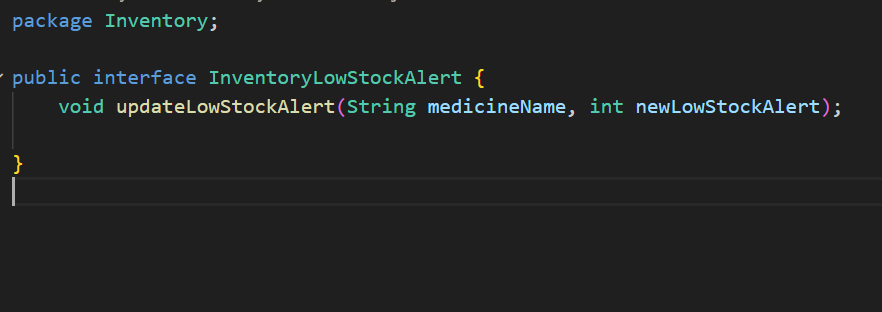
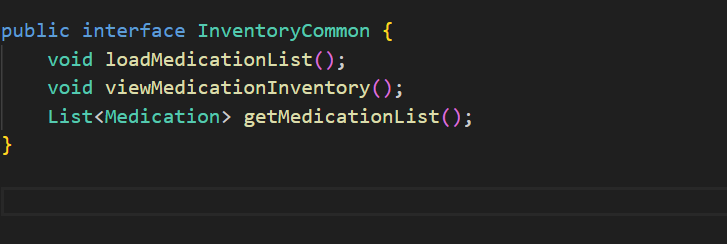
### **2.5 Design Principles**

### **a. Single Responsibility Principle (SRP):**

The **Single Responsibility Principle** states **a class should have only one reason to change.** PharmApptService,AdministratorApptService,DoctorApptService classes focused on tasks relevant to a specific role (pharmacist, administrator, doctor), ensuring modularity and clarity.This reduces overlap and unnecessary dependencies between classes.

#### **b.Open-Closed Principle (OCP)**

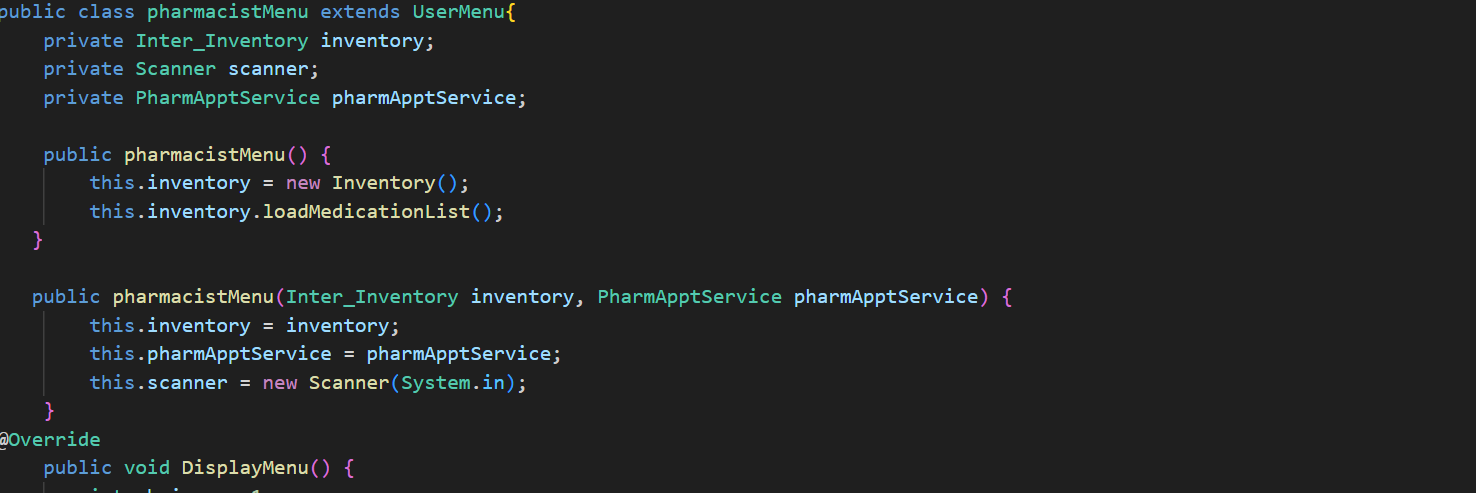
The OCP emphasizes Classes should be open for extension but closed for modification.



**Our Interface-based design** enables adding new functionalities without modifying existing code.If you want to implement a different inventory management system, you can create new class that implements Inter\_Inventory (or one of its smaller interfaces) without modifying the existing Inventory class.

#### **c**. **Liskov Substitution Principle (LSP)**

The Liskov Substitution Principle (LSP) ensures that **a subclass can replace its parent class without breaking the program**. Here's how the code demonstrates this principle:



* **Subclass Extends Parent Class Behavior**

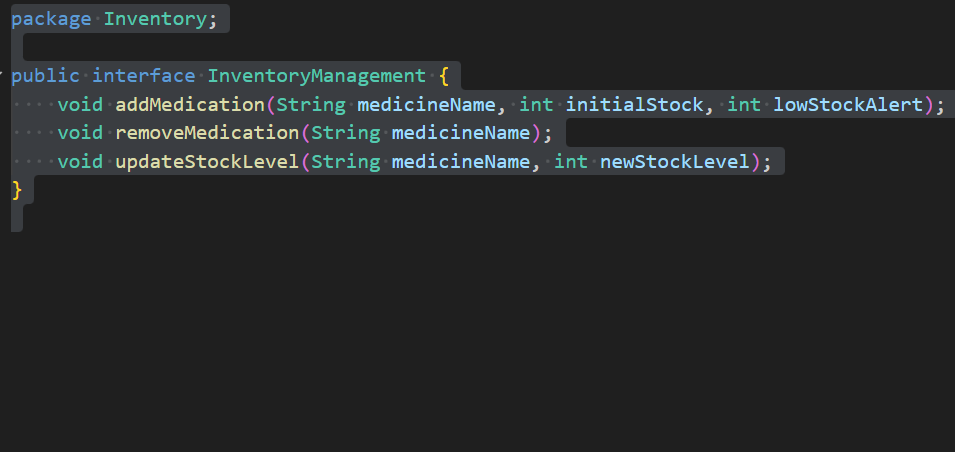
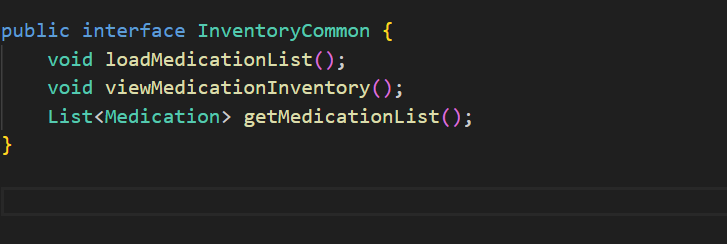
The pharmacistMenu class extends the abstract UserMenu class and implements the DisplayMenu method. This ensures that pharmacistMenu behaves consistently with UserMenu.

### **Substitutability**

The design ensures that the subclass administratorMenu can substitute the parent class UserMenu without causing any issues.This substitution maintains program correctness, a key aspect of LSP.

#### **d. Interface Segregation Principle (ISP)**

The **Interface Segregation Principle** states **clients should not be forced to depend on interfaces they do not use.**



#### Each small interface in the design handles only one aspect of inventory management.This ensures that a client or class depending on an interface is only exposed to the methods it truly needs.

In this way, if a new functionality needs to be added to the inventory system, it can be introduced via a new interface without affecting existing ones.

Changes to one aspect (e.g., replenishment logic) do not impact other unrelated functionalities (e.g., stock updates)

#### **e. Dependency Inversion Principle (DIP)**

The Dependency Inversion Principle (DIP) states that **high-level modules (e.g., administratorMenu) should not depend on low-level modules (e.g., Inventory, staffIN, AppointmentManager); both should depend on abstractions (interfaces or abstract classes)**. Similarly, low-level modules should depend on these abstractions instead of directly interacting with high-level modules.

Here’s how our code adheres to the DIP:

## 

The code uses an abstract class (UserMenu) as the base for high-level and low-level modules. It defines the common functionalities, such as getUserChoice and exitMenu, as well as an abstract DisplayMenu method. By relying on this abstraction, high-level modules interact with UserMenu without knowing the details of its concrete implementations.

## **3. UML Class Diagram** (attached separately)

## 

## **4. Additional Features and Functionalities**

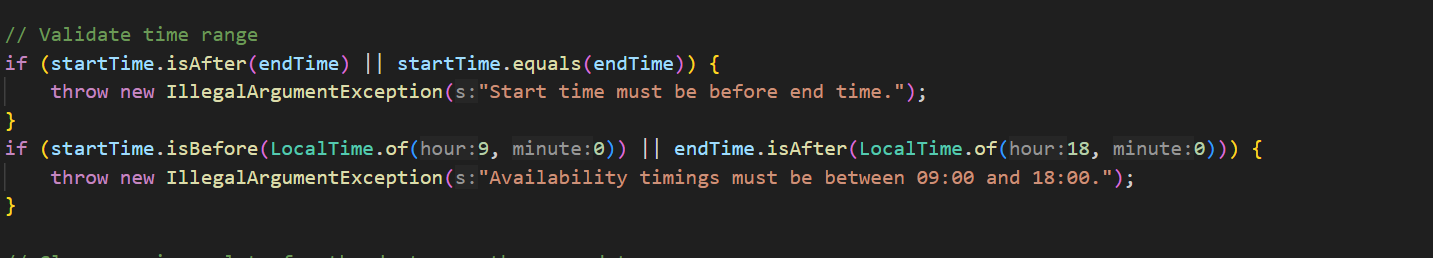
## **4.1 2FA System during User Login**

A **2FA (Two-Factor Authentication)** system is a security mechanism that requires a user to provide two distinct forms of identification to verify their identity before granting access to a system, application, or service. It enhances security by combining two different types of credentials.

* **the workflow of a 2FA system:**
  + **Login Attempt**:the user enters their username and password
  + **Verification of the Second Factor**:The system requests a second authentication factor, such as a one-time password (OTP) sent to the user's phone or email
  + **Access Granted:**
    - If both factors are verified successfully, the system grants access.
    - If either factor fails, access is denied.

**4.2 Hospital Working Hours**

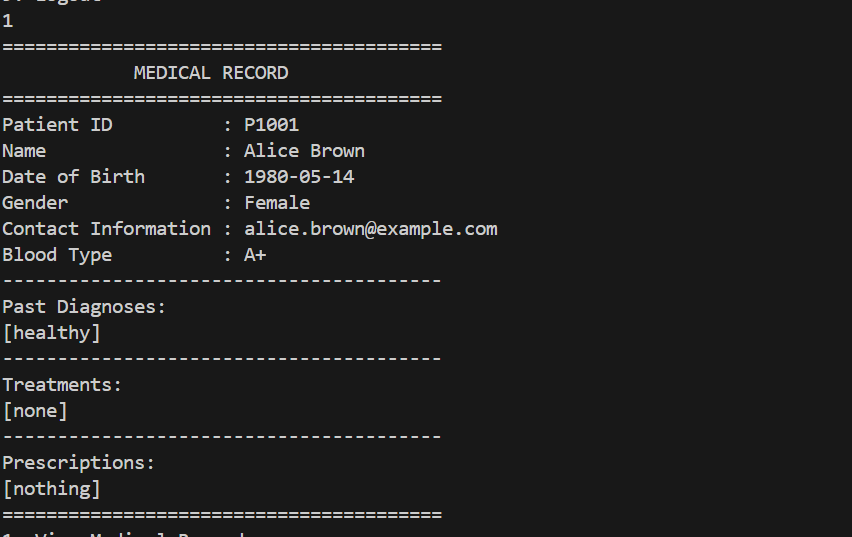
When the doctor sets availability for appointments, the system uses Java’s date and time API from the java.time package for precise handling of date and time operations. Following our previously stated assumptions, we employed:

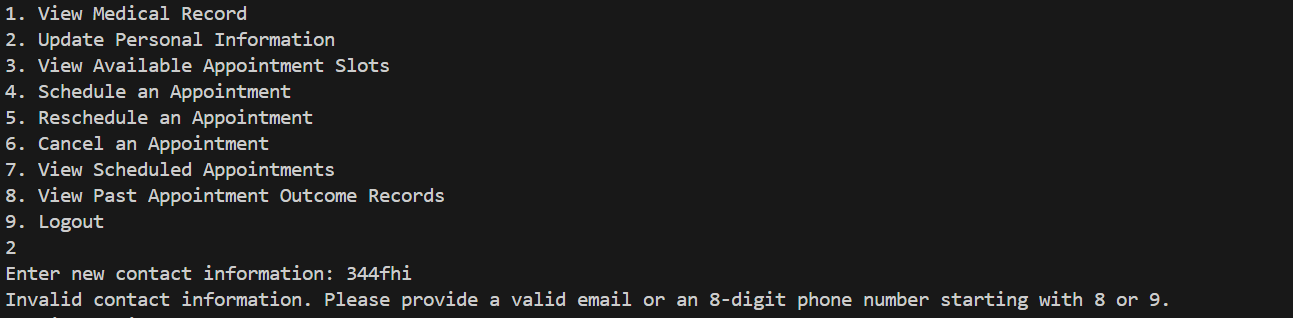
* **LocalDate** (ensuring date is not in the past or more than 1 week ahead)
* **LocalTime** (generating 30-minute intervals within the appointment range)
* **DateTimeFormatter** (parse and validate input time strings in the HH:mm format)

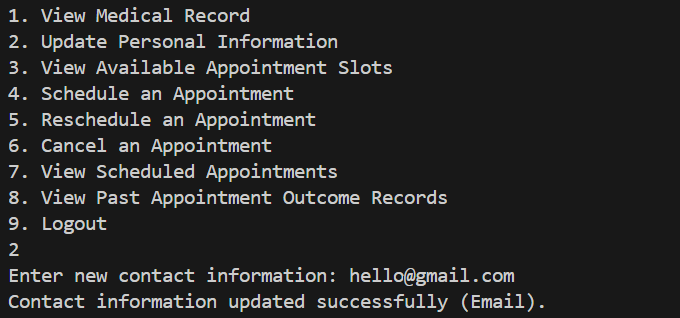
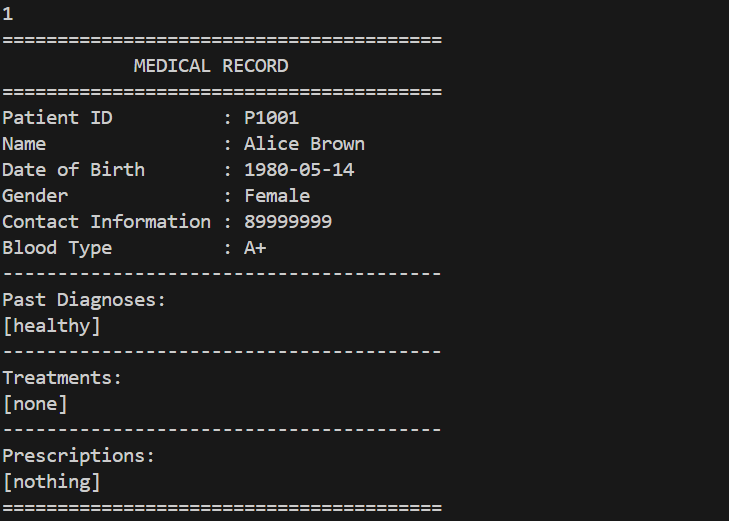
## **5. Testing and Results**

Patient

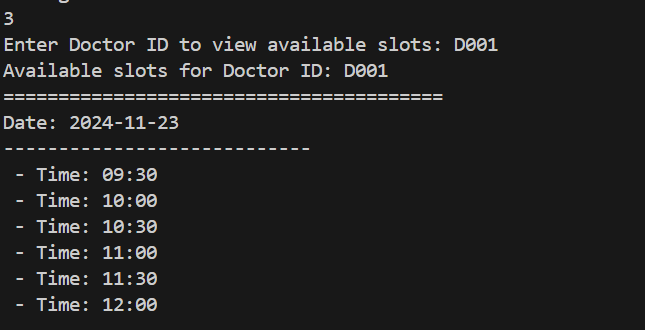
1.View Medical Record



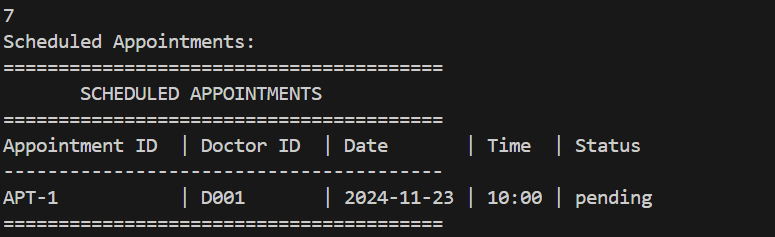
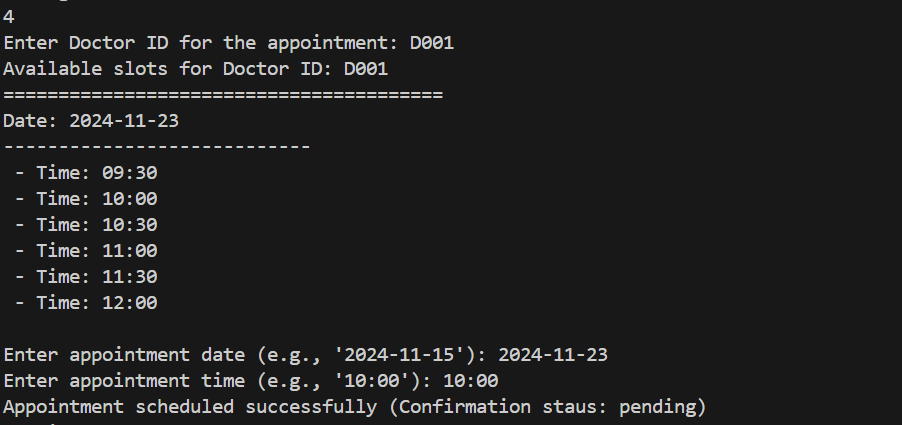
2.Update information



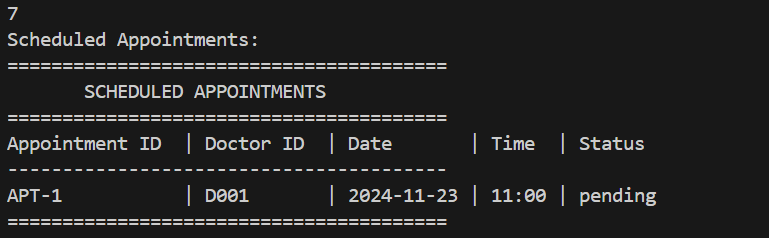
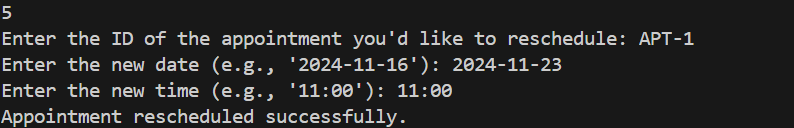
3.View available time slot



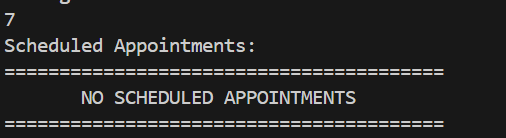
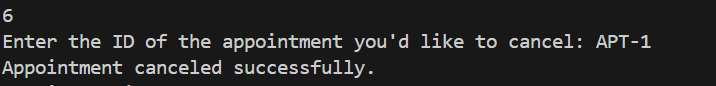
4.Schedule Appointment



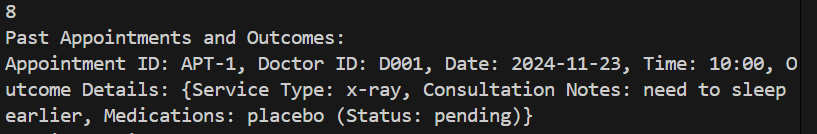
5.Reschedule an appointment



6.Cancel an Appointment

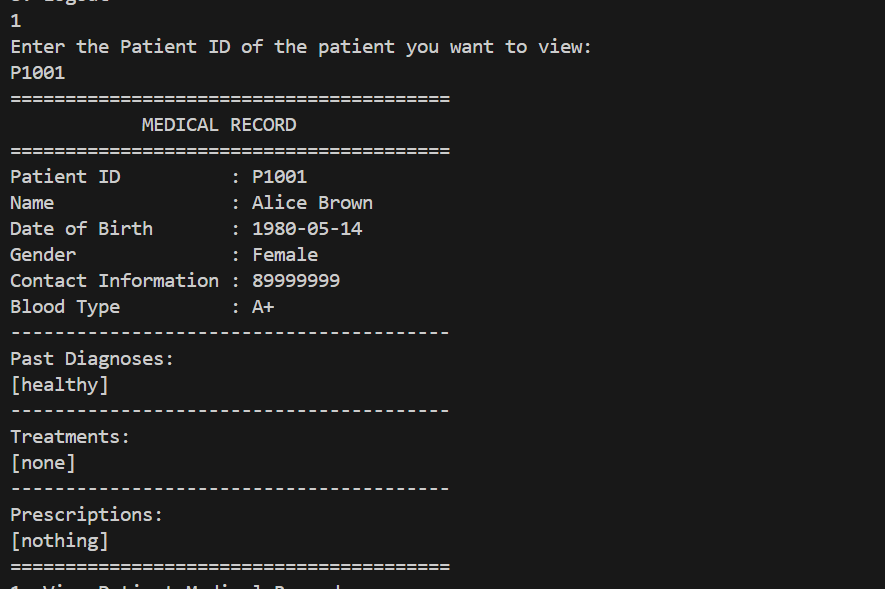


7.Show before

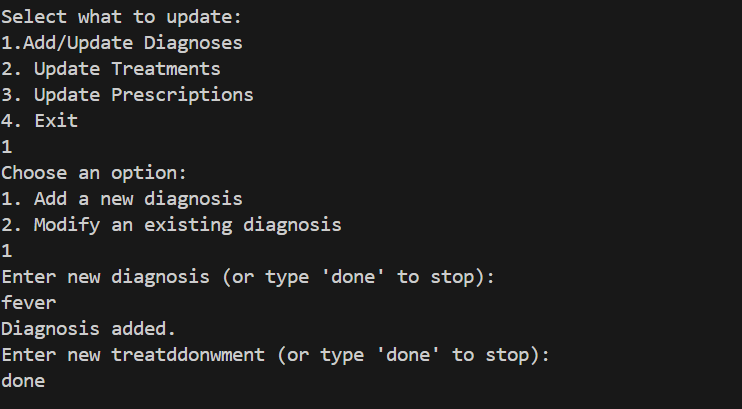
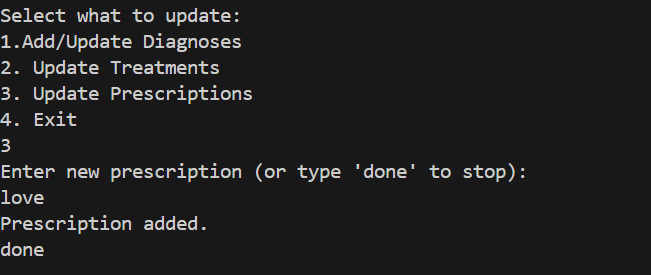
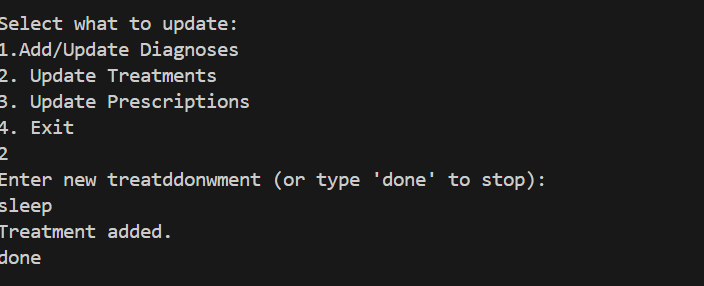
8.

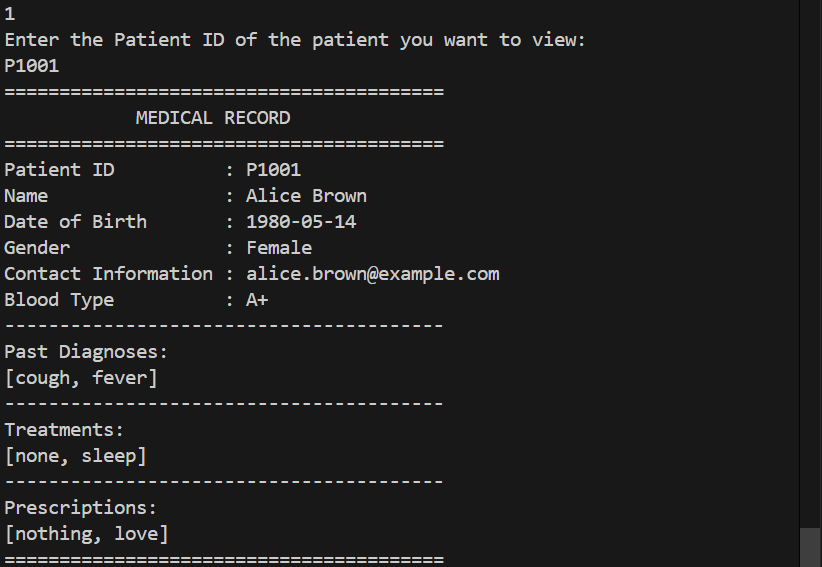
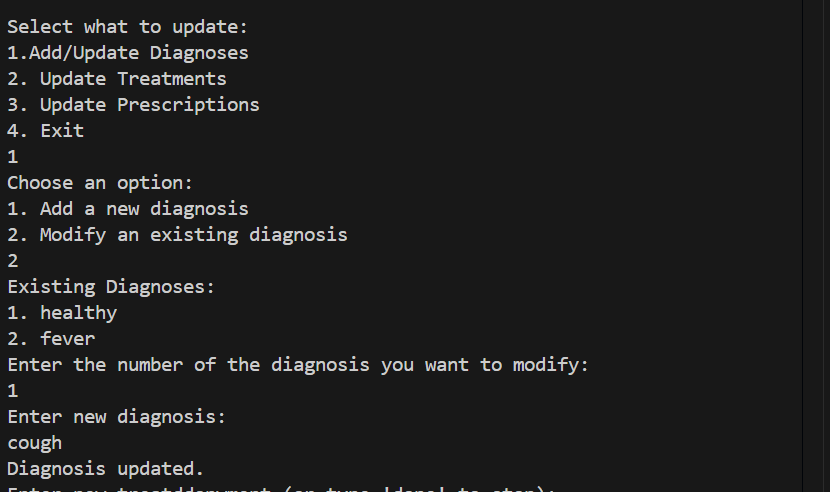
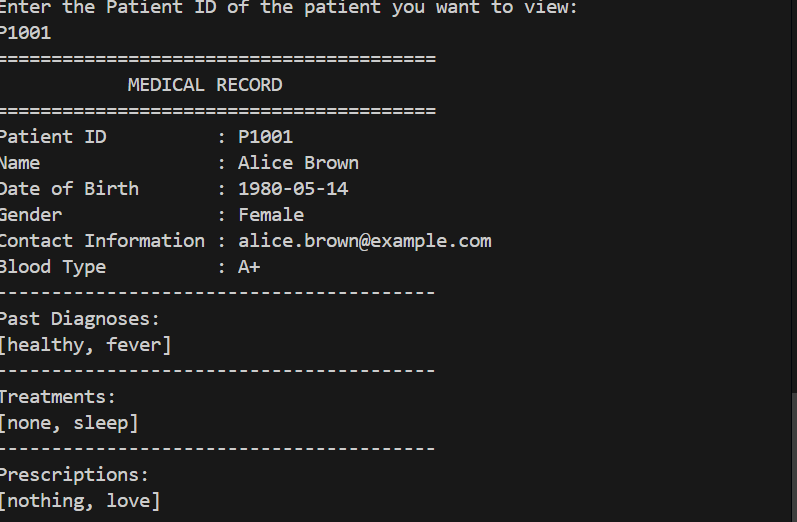
**Doctor Actions:**

**9.view patient medical record**

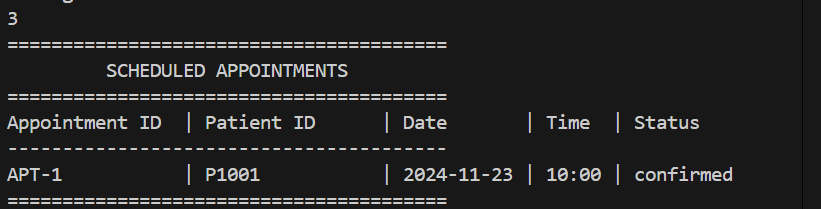
****

**10.Update Medical Records: Reflects new diagnoses and prescriptions.**

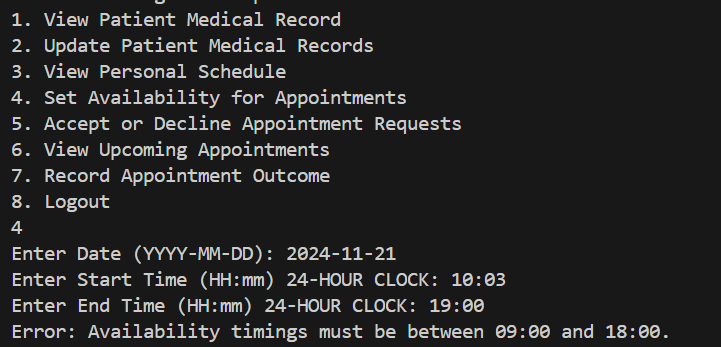
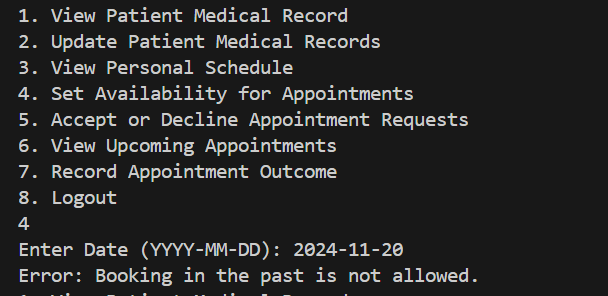


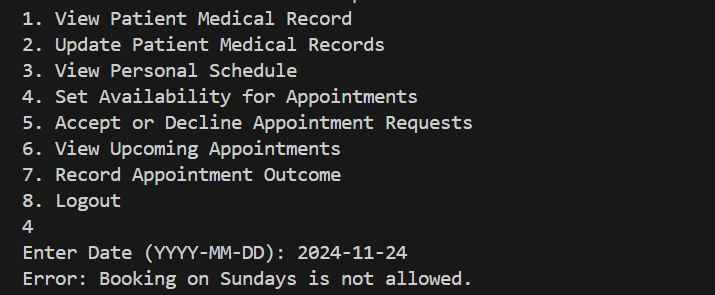
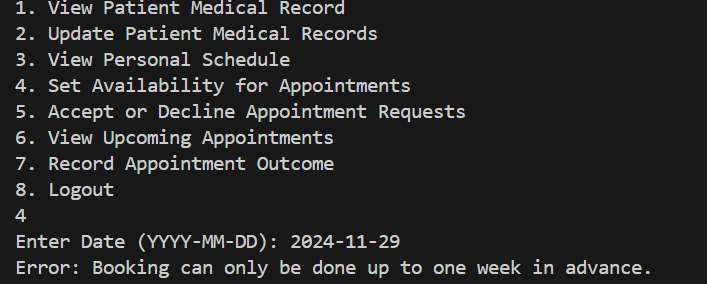


**11.: View Personal Schedule**

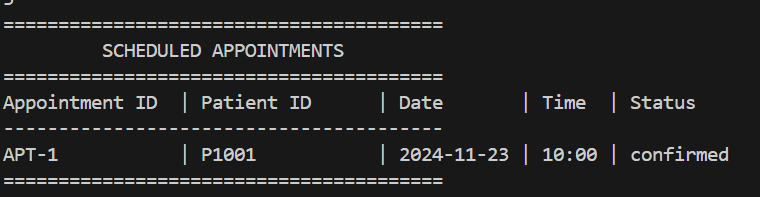
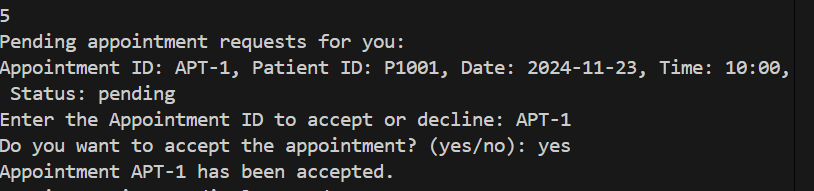


**12.Setting Availability for Appointments based on Constraints**

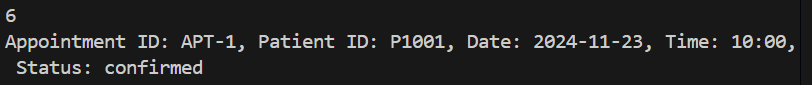




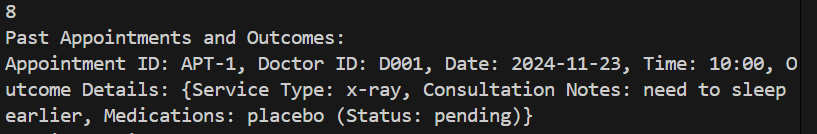
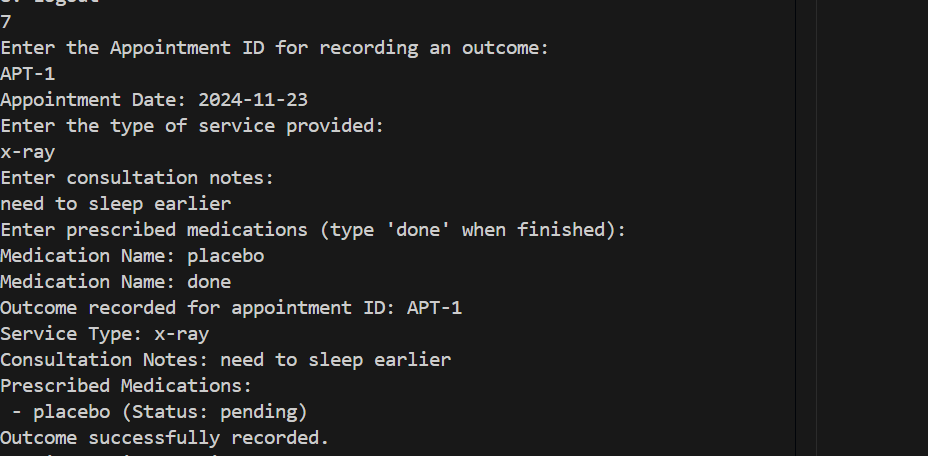
**13.accept/decline appointment**



**14.view upcoming appointment**

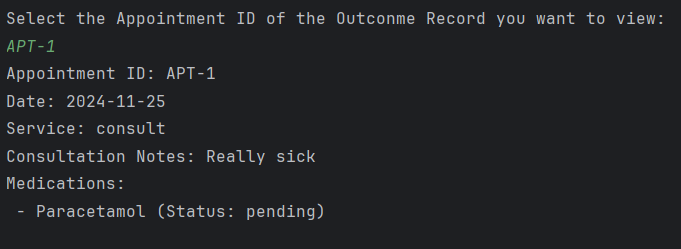


**15.: Record Appointment Outcome**

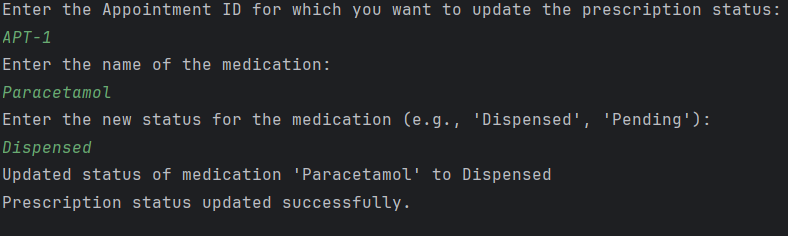
****

**Pharmacist Actions:**

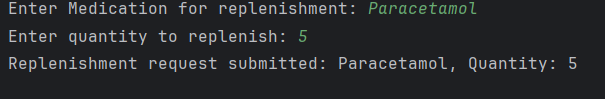
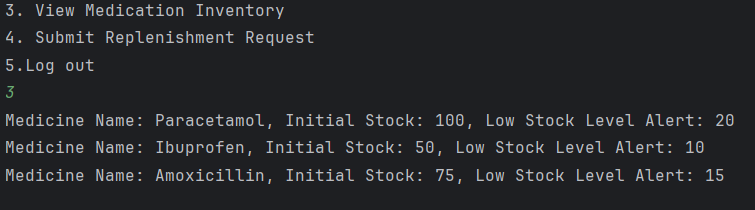
**1. View appointment record:**



**2. Update Prescription Status: Correctly marks prescriptions as "Dispensed."**

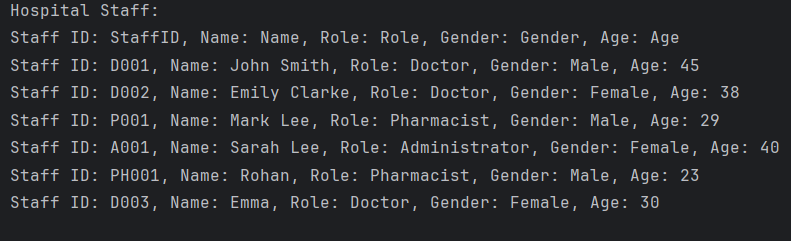
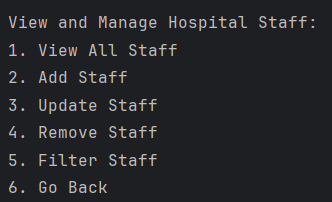


**3. View medication inventory: 4.Submit replenishment request:**

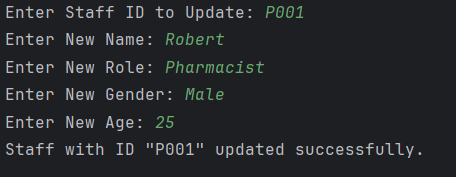
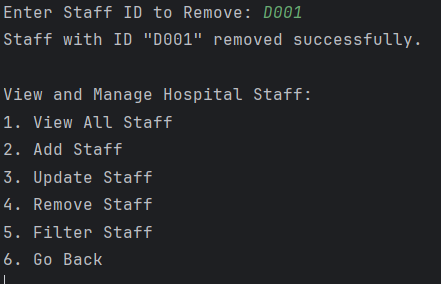
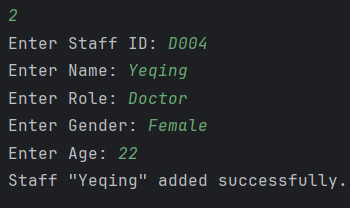


**Administrator Actions:**

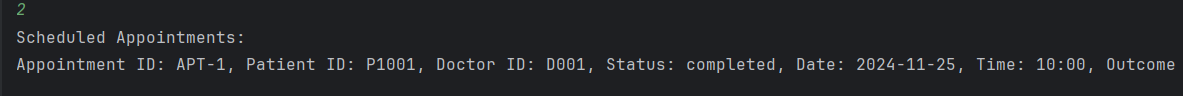
**1. View and manage hospital staff:**



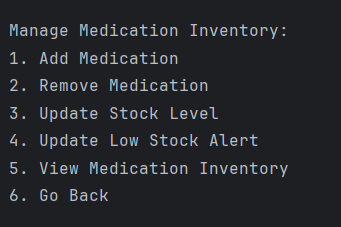
**2. Adding, removing and updating staff list:**



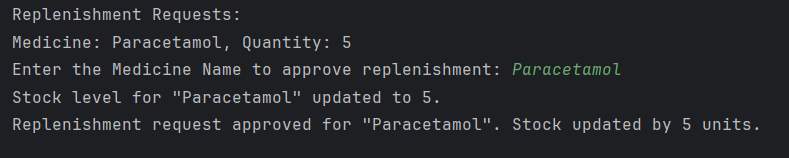
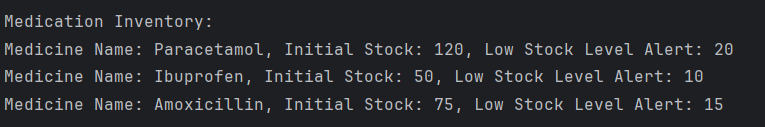
**3. View appointment details:**



**4. View and manage medication inventory:**



**5. Updated Medication Inventory: 6.Approving replenishment requests:**



### **Results**

* Test results confirm functionality adherence to requirements, with minor issues resolved during debugging.

## **6. Reflection**

### **Difficulties Encountered**

* Initial challenges with CSV parsing and data persistence.
* Ensuring role-specific access and menu transitions.

### **Solutions**

* Implemented modular IN classes for data handling.
* Designed a robust user system for authentication.

### **Lessons Learned**

* The importance of OOP principles in large-scale applications.
* Strategies for managing state and transitions in a CLI application.

### **Improvement Suggestions**

* Implementing unit tests for better reliability.
* Extending functionality to include billing and reports.