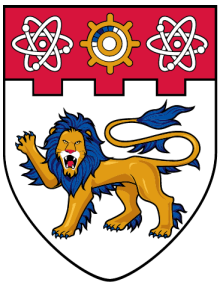


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

College of Computing and Data Science

SC2002 Assignment

Hospital Management System

Report

SCS6 GROUP 5

Name	Matriculation No.
Adam Soh Shi Jie	U2320112A
Chong Jia Chern	U2320724G
Ho Shang Ji Jason	U2322832D
Song Tingfeng	U2320053J
Tan Uei Horng	U2320729F

Table of Contents

---

1. DESIGN CONSIDERATIONS

Hospital Management System (HMS) is a Java-based console application designed with a focus on maintainability and modularity. It streamlines hospital operations by managing core functions like appointment scheduling, inventory and staff administration. The design caters to different user roles (patient, doctor, pharmacist, administrator) each with distinct responsibilities, ensuring the system is adaptable and scalable for future enhancements. The system aims to improve hospital resource utilization, enhance patient care and simplify administrative tasks all through a command-line interface for ease of use and efficiency.

1.1 Design Approach

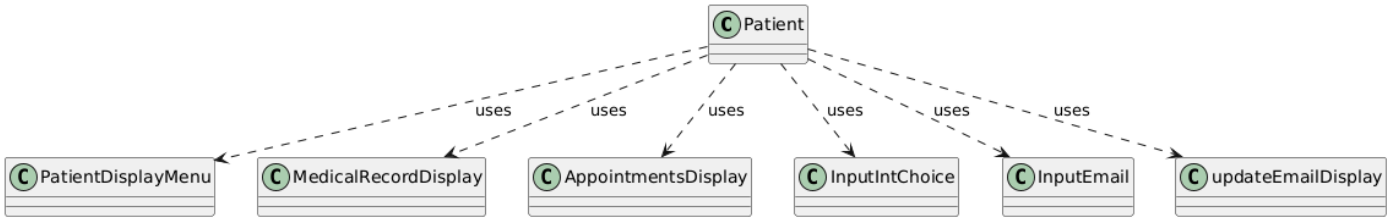
The design of our HMS application follows the course content's object-oriented design approach with a focus on clear separation of concerns to enhance maintainability and scalability. The HMS design adheres to the Model-View-Controller (MVC) architecture, creating a modular system with distinct components.

	Model	View	Controller
Purpose	Encapsulates core data and business logic	Handles user interface and presentation, offering clean interaction through interfaces	Acts as an intermediary between the view and the model, handling system logic and user input

Example Classes	***Patient, Appointment, Medication***	***DisplayInfo, InputString***	***InventoryManagement, StaffManagement***

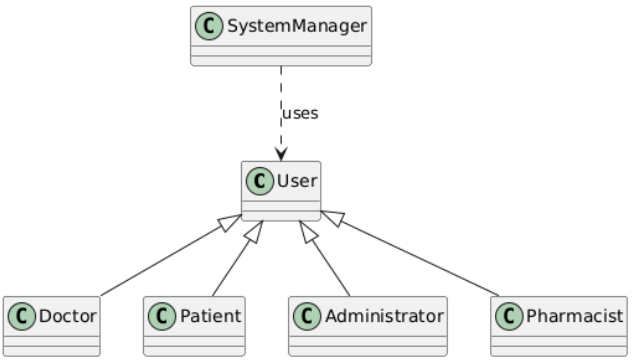
1.2 Applied Design Principles

1.2.1 Single Responsibility Principle (SRP)



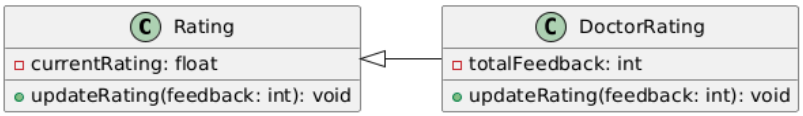
SRP states that a class should have only one reason to change, meaning it should focus on a single responsibility or functionality. Our UML diagram above demonstrates the SRP by assigning each class a specific responsibility. The ***Patient*** class delegates tasks to role-specific components like ***PatientDisplayMenu***, ***MedicalRecordDisplay***, and ***AppointmentsDisplay,*** which manage menu options, medical records, and appointments respectively. Supporting classes like ***InputChoice*** and ***UpdateEmailDisplay*** handle specific tasks like input processing and email updates. This clear separation ensures that changes to one functionality do not affect others, making the system modular, maintainable, and scalable.

1.2.2 Open/Closed Principle (OCP)



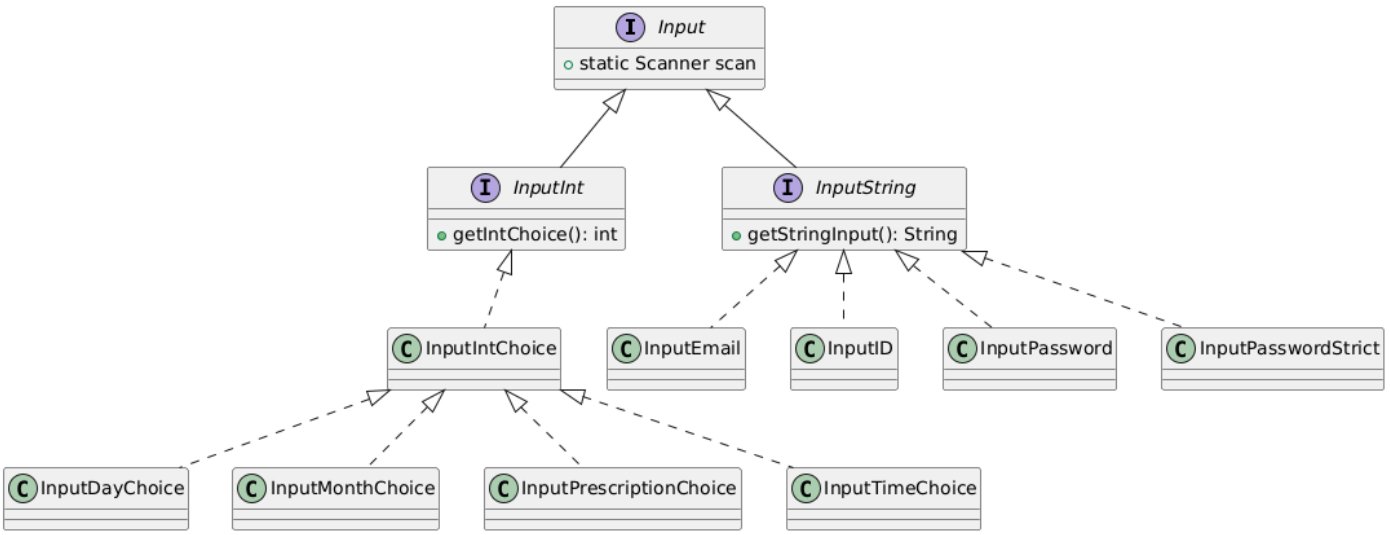
Our design above demonstrates the OCP by designing the ***User*** class to be open for extension but closed for modification. The ***User*** class acts as a base class while specialized classes like ***Doctor, Patient, Administrator***, and ***Pharmacist*** extend it to add their specific behaviors. This allows the system to accommodate new user types like ***Nurses*** by creating additional subclasses without altering the existing ***User*** class or its derived classes and System Manager. This design ensures flexibility, scalability, extendability and adherence to OCP by enabling the addition of new subclasses without risking changes to tested and stable code.

1.2.3 Liskov Substitution Principle (LSP)



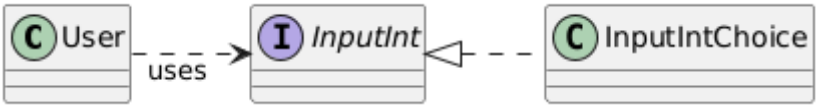
The LSP states that subclasses should be substitutable for their base classes without affecting program correctness. In the HMS, the ***Rating*** class serves as the base class, providing an ***updateRating*** method that takes an integer feedback to update the rating. The ***DoctorRating*** subclass overrides this method with the same signature, taking only the feedback argument and using it to calculate an average rating based on feedback count. The subclass adheres to the base class contract, introducing no additional behavior or requirements. This ensures that ***DoctorRating*** can replace ***Rating*** without changes to the program, exemplifying LSP.

1.2.4 Interface Segregation Principle (ISP)



Our implementation effectively adheres to the ISP by ensuring that interfaces are specific and tailored to the distinct needs of each implementing class. The base ***Input*** interface is extended by more specialized interfaces ***InputInt*** and ***InputString*** which declare methods relevant only to integer and string inputs respectively. This design allows classes such as ***InputDayChoice, InputMonthChoice,*** and ***InputIntChoice*** to implement ***InputInt*** without being burdened by irrelevant string methods while classes like ***InputEmail, InputID, InputPassword,*** and ***InputPasswordStrict*** implement ***InputString*** focusing solely on string input functionalities. By segregating the interfaces in this manner, the system achieves high cohesion and low coupling, enhancing maintainability and scalability while ensuring that each class depends only on the methods it truly requires.

1.2.5 Dependency Injection Principle (DIP)



DIP dictates that high-level modules (such as the ***User*** class) should depend on abstractions, not concrete implementations. In this case, the ***User*** class does not directly rely on the concrete implementation (***InputIntChoice***) for integer input but instead depends on the ***InputInt*** interface. The ***InputIntChoice*** class implements this interface, serving as the lower-level module. This design ensures that the ***User*** class is decoupled from specific implementations, enabling flexibility and easier substitution of input handling logic in the future. By depending on the ***InputInt*** abstraction, the higher-level ***User*** module and the lower-level ***InputIntChoice*** module both conform to DIP principles.

1.3 Applied Object Oriented Concepts

1.3.1 Abstraction

```java
public interface InputInt extends Input{

    public int getIntChoice();

}
```

```java
public class InputIntChoice implements InputInt{
    private int numberOfChoice;


    public InputIntChoice(int no) {
        numberOfChoice = no;
    }

    public int getIntChoice() {

        int choice =-1;
        boolean validity = false;
        while (!validity) {
            try {
                System.out.println(x:"Please input a choice: ");
                choice = scan.nextInt();
```

In this example, we abstracted the concept of getting an integer input by defining the InputInt interface.We then implemented how by defining the InputIntChoice class, implementing the getIntChoice() method.

1.3.2 Encapsulation

```java
public class MedicalRecord {

    private String userId;
    private String email;
    private String gender;
    private String name;
    private String bloodType;
    private String dateOfBirth;
    private ArrayList<AppointmentOutcomeRecord> diagnosesTreatmentPrescription = new ArrayList<>();

    public void updateRecord(String email) {
        this.email = email;
    }
```

In this example, we perform encapsulation by making attributes private. We have setters and getters to access and change these attributes, hence hiding our attributes from the public.

1.3.1 Inheritance

```java
public class Doctor extends User {

    private ArrayList<MedicalRecord> patientMedicalRecords = new ArrayList<>();
    private Available personalSchedule;
```

In this example, our Doctor class inherits from the User base class by extending it. It overrides/has more specialized features and has the same generalized methods as User.

1.3.1 Polymorphism

```java
public class Rating {
    private float rating = 0;
    private int count = 0;

    public void updateRating(int feedback) {
        if(getCount() == 0) {
            setCount(1);
        }
        else {
            count++;
        }

        rating = feedback;
    }
```

```java
public class DoctorRating extends Rating{

    @override
    public void updateRating(int feedback) {

        if(getCount() == 0) {
            setCount(1);
        }
        else {
            setCount(getCount()+1);
        }


        setRating((getRating()+feedback)/getCount());

    }
}
```

DoctorRating extends Rating and overrides the updateRating() method to be more specialized.

The program utilizes the base Rating class to reference objects that may be instances of DoctorRating class (demonstrating upcasting).

1.4 Assumptions

- Doctor Availability
  All doctors are required to manually add their availability to the system for patients to schedule appointments. The system does not automatically assign availability based on predefined schedules.

- 24/7 Service
  The Hospital Management System operates round-the-clock, ensuring uninterrupted service for users, including patients, doctors, pharmacists, and administrators.

- Medicine Quantity
  The inventory system tracks the quantity of medicine in terms of packets rather than individual pills, simplifying stock management and replenishment processes.

- Date & Time
  Date of use is considered on that day itself. (Example: Today's is 18th Nov 2024)
  Meaning that when a Doctor sets his/her availability date, if he sets any day before 18th Nov, it is considered in the year of 2025, while if he sets any data after 18th Nov (Before and including 31st Dec), it is still considered 2024.

1.5 Additional Features

1. **Password validation**
   a. Requirements
      i. At least 1 uppercase letter and 1 lowercase letter.
      ii. At least 1 digit and 1 special character.
      iii. Minimum length of 8 characters.
   b. Implemented in inputPasswordStrict.java
   c. Enhances data security by ensuring strong passwords.

2. **Datetime validation**
   a. Functionality
      i. Verifies that the input follows a valid datetime format (month, day, and time)
      ii. Ensures the provided date exists (e.g., rejects invalid dates like 31st February)

b. Implemented in DayChecker.java
c. Improves system robustness by preventing invalid date inputs.

3. **Inbox**
    a. Features
        i. Each user has an inbox to check and clear messages
        ii. Automatic notifications for users when the following actions occur,
            - Patients schedule, reschedule, or cancel an appointment.
            - Doctors accept or reject appointment requests.
            - Pharmacists make replenishment requests.
    b. Simulates real-life applications, keeping users informed about important actions.

4. **Rating**
    a. Functionality
        i. Patients can rate doctors (out of 5) after completing an appointment.
        ii. Admins can view doctors' ratings for performance reviews.
    b. Implemented in DoctorRating and UpdateRating.java
    c. Facilitates feedback collection and supports employee performance evaluation.

5. **UI enhancement**
    a. Pretty printing
        i. Boxes for user menus and inboxes
        ii. Borders for information display
    b. Display HMS title in ASCII art

6. **Password Invisibility**
    a. Hides password input while typing for added privacy and security.
    b. Implemented in SessionManager.java

2. DETAILED UML CLASS DIAGRAM

*< refer **hms_classdiagram.svg** for high-resolution clear viewing >*

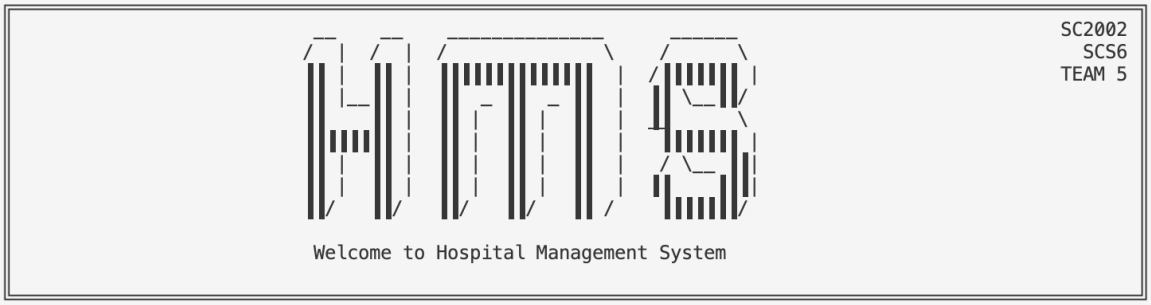
3. TESTING

3.1 Login Page

3.1.1 Welcome Screen

Upon starting the program, the **system** displays a welcome screen.

```
System starting...
Loading data from CSV files...
```



3.1.2 Enter Hospital ID and Password

The **system** prompts the user to enter their hospital ID and default password. After that, the user is asked to change to a strong password based on the password validation requirements.

To begin with, we will log into a doctor's account.

```
Enter hospital ID: D001
ID match found!

Enter password: password
Login successful!

Please change your password for security.
Enter new password: Password1!
Password changed successfully!
```

3.2 Schedule Appointment

3.2.1 Doctor set availability

Upon successful login, the system displays a menu for the doctor. The **doctor** sets his availability for appointments by entering the month, day and time.

```
WELCOME, John Smith!

+=============================================+
|                Doctor Menu                  |
+=============================================+
| 1) View Patient Medical Records             |
| 2) Update Patient Medical Records           |
| 3) View Personal Schedule                   |
| 4) Set Availability for Appointments        |
| 5) Accept or Decline Appointment Requests   |
| 6) View Upcoming Appointments               |
| 7) Record Appointment Outcome               |
| 8) View Inbox                               |
| 9) Logout                                   |
+=============================================+

Please input a choice:
4
```

```
                Select Month
-----------------------------------------------
Please select a Month:
1) Jan
2) Feb
3) Mar
4) Apr
5) May
6) Jun
7) Jul
8) Aug
9) Sep
10) Oct
11) Nov
12) Dec
-----------------------------------------------

Please input a choice:
11
```

```
                  Select Day
-----------------------------------------------
Please select a number 1-31:
-----------------------------------------------

Please input a choice:
15


                  Select Time
-----------------------------------------------
Please select timing:
1) 8AM
2) 9AM
3) 10AM
4) 11AM
5) 12PM
6) 1PM
7) 2PM
8) 3PM
9) 4PM
10) 5PM
-----------------------------------------------

Please input a choice:
3
```

3.2.2 Patient select appointment date (include reschedule & cancel appointment)

After switching accounts, the **patient** schedules an appointment by choosing a doctor and an available slot.

```
WELCOME, Alice Brown!

+=============================================+
|                Patient Menu                 |
+=============================================+
| 1) View Medical Record                      |
| 2) Update Personal Information              |
| 3) View Available Appointment Slots         |
| 4) Schedule an Appointment                  |
| 5) Reschedule an Appointment                |
| 6) Cancel an Appointment                    |
| 7) View Scheduled Appointments              |
| 8) View Past Appointment Outcome Records    |
| 9) View Inbox                               |
| 10) Rate a Doctor                           |
| 11) Logout                                  |
+=============================================+

Please input a choice:
4
```

```
                Available Date
-------------------------------------------------

1)  Doctor: John Smith
    Date: 15 NOV
    Time: 10AM

2)  Doctor: Emily Clarke
    Date: 16 NOV
    Time: 3PM

-------------------------------------------------

Enter 0 to exit scheduling

Please enter your choice: 1
```

The **patient** can also reschedule her appointment to another available timeslot, or cancel the appointment.

```
Please input a choice:
5
```

```
       Appointment
-------------------------------------------------

1)  Doctor: John Smith
    Patient: Alice Brown
    Date: 15 NOV
    Time: 10AM
    Status: Pending

-------------------------------------------------

Please select an appointment to cancel -> Please input a choice:
1


                Available Date
-------------------------------------------------

1)  Doctor: John Smith
    Date: 15 NOV
    Time: 10AM

2)  Doctor: Emily Clarke
    Date: 16 NOV
    Time: 3PM

-------------------------------------------------

Enter 0 to exit scheduling

Please enter your choice: 2
```

```
Please input a choice:
6
```

```
       Appointment
-------------------------------------------------

1)  Doctor: Emily Clarke
    Patient: Alice Brown
    Date: 16 NOV
    Time: 3PM
    Status: Pending

-------------------------------------------------

Please select an appointment to cancel -> Please input a choice:
1
```

3.2.3 Doctor accept or decline appointment request

The **doctor** can then accept or reject the appointment request from the patient.

```
WELCOME, John Smith!

+=============================================+
|                Doctor Menu                  |
+=============================================+
| 1) View Patient Medical Records             |
| 2) Update Patient Medical Records           |
| 3) View Personal Schedule                   |
| 4) Set Availability for Appointments        |
| 5) Accept or Decline Appointment Requests   |
| 6) View Upcoming Appointments               |
| 7) Record Appointment Outcome               |
| 8) View Inbox                               |
| 9) Logout                                   |
+=============================================+

Please input a choice:
5
```

```
       Appointment
-------------------------------------------------

1)  Doctor: John Smith
    Patient: Alice Brown
    Date: 15 NOV
    Time: 10AM
    Status: Pending

-------------------------------------------------

1 to Accept || 2 to Decline
Please input a choice:
1
```

3.2.4 View Upcoming Appointments

After accepting the appointment, the **patient** or **doctor** can view details of all upcoming appointments.

```
Please input a choice:
6

        Appointment
------------------------------------------------

1)  Doctor: John Smith
    Patient: Alice Brown
    Date: 15 NOV
    Time: 10AM
    Status: Accepted

------------------------------------------------
```

3.3 Appointment Result

3.3.1 Doctor Record Appointment Outcome

The **doctor** completes the appointment and records the appointment outcome by filling in details such as the diagnosis, treatment and consultation notes. After that, he can choose to prescribe medication for the patient by selecting medication from the inventory.

```
Please input a choice:
7

        Appointment
------------------------------------------------

1)  Doctor: John Smith
    Patient: Alice Brown
    Date: 15 NOV
    Time: 10AM
    Status: Accepted

------------------------------------------------

Please select an appointment to complete:
Please input a choice:
1
```

```
------------------------------------------------
1)Prescribe Medication
2)Finish
------------------------------------------------

Please input a choice:
1

                Medication
------------------------------------------
1) Paracetamol
2) Ibuprofen
3) Amoxicillin
------------------------------------------

Please input a choice:
1
```

```
                Medication
------------------------------------------------
1) Paracetamol
2) Ibuprofen
3) Amoxicillin
------------------------------------------------

Please input a choice:
2
```

```
Please enter your diagnoses:
fever

Please enter your treatment:
1-day hospitalisation, rest and drink more fluids

Please enter your consultation notes:
temperature 41^C, exposed to family members, recently travelled overseas, prescribe paracetamol and ibuprofen
```

3.3.2 Pharmacist Prescribe Medicine

After logging in, the **pharmacist** can prescribe the medicine required by the doctor.

```
WELCOME, Mark Lee!

+==========================================+
|           Pharmacist Menu                |
+==========================================+
| 1) View Appointment Outcome Record       |
| 2) Update Prescription Status            |
| 3) View Medication Inventory             |
| 4) Submit Replenishment Request          |
| 5) View Inbox                            |
| 6) Logout                                |
+==========================================+

Please input a choice:
2
```

```
       Update Prescription Status
------------------------------------------------

Medicine Name: Paracetamol
Medicine Status: PENDING


Medicine Name: Ibuprofen
Medicine Status: PENDING

------------------------------------------------
```

```
Do you wish to update all prescription status?
1. Yes
2. No
3. Quit

Please input a choice:
1
```

```
Paracetamol Prescribed. Stocks Remaining: 99

Medicine Name: Paracetamol
Medicine Status: DISPENSED

Ibuprofen Prescribed. Stocks Remaining: 49

Medicine Name: Ibuprofen
Medicine Status: DISPENSED

Next patient record. . .

Actions completed!
Quitting Prescription Status. . .
```

3.3.3 View Medical Record

The **patient** can view her medical record, which displays her personal and medical information. As the appointment has been completed, the diagnoses, treatments and prescriptions section is filled.

```
Please input a choice:
1


          Alice Brown Medical Record
-----------------------------------------------
P1001
Alice Brown
alice.brown@example.com
1980-05-14
Female
A+

** Past Diagnoses, Treatments and Prescriptions **

Date: 15 NOV
Diagnoses: fever
Treatment: 1-day hospitalisation, rest and drink more fluids
Consultation Notes: temperature 41^C, exposed to family members, recently travelled overseas, prescribe paracetamol and ibuprofen
Prescriptions: Paracetamol(DISPENSED) Ibuprofen(DISPENSED)

-----------------------------------------------
```

3.4 Manage Inventory

3.4.1 Submit Replenishment Request

The **pharmacist** can submit a replenishment request for any medicine in the inventory, which is necessary when the stock is running low.

```
Please input a choice:
4


Replenishment Request:

        Current Inventory Information
-----------------------------------------------

Medicine Name: Paracetamol
Stock Available: 99
Stock Status: SUFFICIENT
Request Status: NULL

-----------------------------------------------
```

```
Submit replenishment request?
1. Yes
2. No

Please input a choice:
1


Medicine Name: Paracetamol
Stock Available: 99
Stock Status: SUFFICIENT
Request Status: PENDING

Submission sent successfully! Waiting approval from administrator.
Next stock. . .
```

3.4.2 View Inventory

Meanwhile, the **administrator** can view and manage the medication inventory, which will display all the medicine name, stock available, stock status and replenishment request status.

```
WELCOME, Sarah Lee!

+=============================================+
|                Admin Menu                   |
+=============================================+
| 1) View and Manage Staff                    |
| 2) View Appointment Details                 |
| 3) View and Manage Medication Inventory     |
| 4) Approve Replenishment Requests           |
| 5) Shut Down                                |
| 6) View Inbox                               |
| 7) Log Out                                  |
+=============================================+

Please input a choice:
3


                Inventory Menu
-----------------------------------------------
1) Add Stocks
2) Update Stocks
3) Remove Stocks
4) Update Alert Line
5) Approve Replenishment Requests
6) Display Inventory
7) Exit
-----------------------------------------------

Please input a choice:
6
```

```
        Current Inventory Information
-----------------------------------------------

Medicine Name: Paracetamol
Stock Available: 99
Stock Status: SUFFICIENT
Request Status: PENDING


Medicine Name: Ibuprofen
Stock Available: 49
Stock Status: SUFFICIENT
Request Status: NULL


Medicine Name: Amoxicillin
Stock Available: 75
Stock Status: SUFFICIENT
Request Status: NULL

-----------------------------------------------
```

3.4.3 Approve Replenishment Request

The **administrator** can also approve replenishment requests from the pharmacist.

```
Please input a choice:
4
```

```
        Current Inventory Information
-----------------------------------------------

Medicine Name: Paracetamol
Stock Available: 99
Stock Status: SUFFICIENT
Request Status: APPROVED


Medicine Name: Ibuprofen
Stock Available: 49
Stock Status: SUFFICIENT
Request Status: NULL


Medicine Name: Amoxicillin
Stock Available: 75
Stock Status: SUFFICIENT
Request Status: NULL

-----------------------------------------------
```

4. REFLECTION

This HMS project presented several challenges, each requiring thoughtful strategies to overcome. One of the primary difficulties encountered was related to robustness issues in the code. To address this, extensive use of try-catch blocks, Regex, and logical enhancements were implemented to improve stability and error handling. Another significant challenge was the tendency for some classes to take on too many

responsibilities, such as a "Patient" class printing its own menu, which violated the Single Responsibility Principle (SRP) and resulted in messy code. This was resolved by applying SOLID principles and incorporating software engineering design patterns like the facade and factory patterns, ensuring better modularity and cleaner architecture.

Logical errors were another obstacle, necessitating a considerable amount of time spent understanding the root causes of these errors and debugging the code thoroughly. Finally, creating a complex yet visually appealing user interface (UI) for the HMS posed its own set of challenges. Overcoming this required extensive learning, experimentation, and iteration to enhance both the design and functionality of the interface. Through these challenges, valuable knowledge was gained ranging from robust error handling and design principles to debugging techniques and UI development. These experiences have highlighted areas for further improvement, such as refining problem-solving approaches, deepening the understanding of design patterns and enhancing UI/UX skills.

4.1 Future Enhancement

To further enhance the system in the future, several key improvements can be made. First, incorporating an actual date-time system would address limitations in the current design, which relies on assumptions about time. Additionally, introducing a patient registration feature would fill a significant gap, as the current system only allows admins to add new staff, leaving no way to register patients. Extending the user class to include a Nurse class would enable the addition of new features tailored to specific roles, improving functionality and scalability. Transforming the application into a Graphic User Interface (GUI) would significantly enhance user experience and accessibility. Finally, implementing additional software engineering design principles, such as the observer pattern would further improve the system's modularity, flexibility, and maintainability.

4.2 What We Learnt

Throughout this course and assignment, we gained a comprehensive understanding of various programming concepts and skills. We deepened our knowledge of Object-Oriented Programming (OOP), mastering core principles such as abstraction, encapsulation, inheritance, and polymorphism. Additionally, we became proficient in Java and C++ programming languages, building a strong foundation for software development. We also learned the importance of adhering to SOLID design principles when designing programs, which helped us create more modular, maintainable, and scalable systems. Exception and error handling became a crucial focus during the E-learning week, where we explored strategies for managing errors effectively. Moreover, we enhanced our ability to design and interpret UML class diagrams, gaining insights into different class types and their relationships, which significantly improved our software modeling and design skills. These lessons have equipped us with both the theoretical understanding and practical experience needed for effective software development.