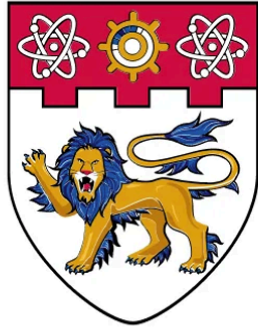


SC2002 : OBJECT ORIENTED DESIGN AND PROGRAMMING
AY 24/25 SEMESTER 1 GROUP ASSIGNMENT








**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

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 honoured 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)	Lab Group	Signature /Date
Chong Ting Hao	SC2002	3	 18/11/24
Ng Zhengbin Claven	SC2002	3	 18/11/24
Stanley Benjamin Yukon	SC2002	3	 18/11/24
Ang Jin He Darrel	SC2002	3	 18/11/24
Lim Kang You	SC2002	3	 18/11/24

1. Design Considerations

1.1 Approach

HMS (Hospital Management System) is a java Command Line Interface(CLI) application. It is made to automate hospital operations such as patient and staff management, appointment scheduling and medicine supply tracking. It aims to facilitate an efficient and effective management of hospital resources via the streamlining of administrative procedures.

We utilised Model-View-Control(MVC) architecture in the design of our classes. Users interact with the application through menus (Boundary/View class). It requests data from managers, which acts as Control/Logic classes, implementing logic and performing error checking and exception handling to ensure application works as intended. The data is fetched from Entity/Model classes, which store the data.

1.2 Assumptions

1. No deletion of Medical Records
2. No deletion of past or cancelled Appointments
3. No deletion of past Prescriptions
4. No deletion of past or cancelled Replenishment Requests

This is to ensure that there is history and receipts of everything.

1.3 SOLID Design Principles

1.3.1 Single Responsibility Principle (SRP)

The single responsibility principle ensures that a class is never overloaded with responsibilities. Each class should do at most one operation and serve one purpose. An example of this might be how our AdministratorUI is further split into AdministratorManageStaffUI and itself split into AdministratorManageStaffUpdateUI. As putting everything into a single UI would overload the AdministratorUI with too many responsibilities.

1.3.2 Open-Close Principle (OCP)

The open-close principle states that a module should be closed to modification but open to extension. It means that we should be able to add new functionalities to the program without modifying previous classes. Our AbstractAuthMenu is an example of this principle as we are able to further add more authenticated UIs should there be more roles added to the HMS.

```
abstract class AbstractAuthMenu implements IDisplayOptions, ILaunchAuthMenu {
```

1.3.3 Liskov Substitution Principle (LSP)

The Liskov Substitution Principle states that Derived Classes should be substitutable for their Base Classes. In other words, Derived Classes that inherit from a Base Class should be capable of carrying out all the functionalities of their Base Class given the same pre-conditions. For example, in our viewScheduledAppointments method in PatientManager, we can substitute the User class for any of its Derived classes (in this case we are passing a patient)and it would still work.

```
/**
 * Displays all scheduled and pending appointments for a patient.
 *
 * @param p The patient User object whose appointments should be shown
 */
public void viewScheduledAppointments(User p) {
    List<Appointment> scheduledAppointments = db.getAllAppointments()
        .stream()
        .filter(a -> a.getPatientId() != null) // Check null first
        .filter(a -> a.getPatientId().equals(p.getId())) // Then check equality
        .filter(a -> a.getAppointmentStatus().equals(Appointment.AppointmentStatus.SCHEDULED)
            || a.getAppointmentStatus().equals(Appointment.AppointmentStatus.PENDING))
        .collect(Collectors.toList());

    if (scheduledAppointments.isEmpty()) {
        System.out.println(x:"You have no scheduled appointments");
    }
    else {
        System.out.println(x:"----YOUR SCHEDULED APPOINTMENT SLOTS-----");
        int count = 1;
        for (Appointment a : scheduledAppointments) {
            System.out.println("SLOT " + count++ + "");
            a.print();
        }
    }
}
```

1.3.4 Interface Segregation Principle (ISP)

Interface segregation states that when using interfaces, each interface should be as small as possible. This is to avoid Derived classes having to provide implementation for abstract functions they don't necessarily need. Many specific interfaces are generally better than one general interface and a class should implement many interfaces as opposed to one large interface.

```
•○ IAuthChangePassword
•○ IChangePassword M
•○ ICheckMedicineExists
•○ IDisplayOptions
•○ IHashPassword
•○ ILaunchAuthMenu
•○ IViewMedicalRecord
•○ IViewMedicineInventory
•○ IViewScheduledAppointments
```

1.3.5 Dependency Injection Principle (DIP)

The dependency injection principle states that high level modules should not depend on low level ones. Both should depend on Interfaces or Abstract Functions and Classes. In our IChangePassword and IAuthChangePassword, we extend the IHashPassword interface. IChangePassword is only used on first login as it has additional dialogue whilst

IAuthChangePassword is used for all the other authenticated UIs. We could have implemented a general ChangePassword interface that both our current ChangePassword interfaces would extend and dependency injected that general interface into all our Managers and have it satisfy this principle but since we only used the first login interface in one of our Managers, we thought it was excessive.

1.4 APIE Design Principles

1.4.1 Abstraction (A)

Abstraction is the process whereby we identify the “relevant” details and “ignore” the unnecessary details of an object. This allows us to more efficiently implement an object’s essence and functions in code. An example of this is how a human being might have many attributes such as height or eye colour. However, we would only be interested in attributes relevant to the HMS such as their name and role.

```
/**
 * Represents a user in the system with basic profile information and authentication capabilities.
 * This class implements Serializable to allow for object persistence.
 */
public class User implements Serializable, IHashPassword{
    /**
     * The unique identifier for the user
     */
    private String id;
    /**
     * The user's name
     */
    private String name;
    /**
     * The password stored as a hash for the user
     */
    private String password;
    /**
     * The user's role, one of Doctor, Patient, Administrator, Pharmacist
     */
    private String role;
    /**
     * The user's gender, Male or Female
     */
    private String gender;
    /**
     * A boolean that is true if this is the user's first login, false otherwise
     */
    private boolean firstLogin;
```

1.4.2 Polymorphism (P)

Polymorphism in Java refers to the ability of objects to take on many forms. A program may invoke a method through a superclass variable, however, the actual version of the method called may depend on the actual object pointed to by the superclass reference. In other words, Polymorphism allows objects to respond to the same message or method call in multiple ways. An example of this in our code is in our HMSManager, where the login function simply returns a User but this User is passed into all the different UIs and downcasted to their specific roles.

```
@Override
public void launchAuthMenu(User u) {
    // Downcast user to patient
    Patient p = (Patient) u;

    boolean loggedIn = true;
```

1.4.3 Inheritance (I)

Inheritance in Java is a mechanism by which one class is allowed to reuse the fields and methods of another class. It quite simply is the creation of new classes based on existing ones. An example of this could be any of the Derived classes such as Doctor, Pharmacist or Administrator inheriting the attributes name and password from Base class Staff.

1.4.4 Encapsulation & Information Hiding (E)

Encapsulation means bundling related data and methods into a single unit, such as an object. Information hiding means restricting access to the internal details of an object, such as its data fields or implementation logic, from the user, other objects or other classes. An example of this in our code is how we set the visibility of attributes to private. They can be accessed via get & set methods.

```
public class Patient extends User {
    private String bloodType;
    private LocalDate dateOfBirth;
    private String email;
    private String contactNumber;
```

1.5.1 Extra Design Patterns

1.5.1 (A) Singleton Database and Scanner

The singleton pattern is a design pattern that restricts the instantiation of a class to a singular instance. The pattern is useful when exactly one object is needed to coordinate actions across a system. The Singleton does have some drawbacks however.

Firstly, it violates the Single Responsibility Principle (SRP) as a Singleton will have to manage access to itself, especially when several threads request access to its one instance.

If the Singleton takes responsibility for dispatching data to each request i.e. preventing Race-Conditions or handling read and write to deliver valid data, we could thus end up with one class that has too many responsibilities.

Secondly, it violates the Open-Closed Principle (OCP). For a class to be “open” it must be possible to inherit from it. As the Singleton class inhibits inheritance, it’s no longer “open”. Furthermore, if a Singleton class allows inheritance to make itself “open” for extension, it can no longer enforce the singleton pattern.

Thirdly, it violates the Dependency Injection Principle (DIP) as many objects will become dependent on its **concrete** instantiation.

1.5.1 (B) Efficient Serialise/ Deserialize

Data is maintained via Hashmaps and ArrayLists in the Singleton HMSDatabase class that are updated via the various other manager class methods. This allows for easy and efficient transfer of data from program to hard files. Serializing and Deserializing is only done upon program entry and exit respectively. This is to maintain efficiency by ensuring the computationally costly serializing and deserializing functions are called as little as possible, increasing efficiency.

1.5.2 Extra Feature 1: Password Hashing & Security

Keeping all the passwords as Strings within Objects within the ArrayList<User> *allUsers* means that hackers or corrupt administrators could have very easy access to passwords as the passwords already are in String Format. We thus implemented Password Hashing to increase security.

The login for loop within USER also does not return to main class HMS upon the entry of an invalid username as that would allow hackers to guess valid usernames via brute force guessing.

```
public User login() {
    System.out.println(x:"Enter your login id: ");
    String loginId = scanner.nextLine();
    System.out.println(x:"Enter your password: ");
    String password = scanner.nextLine();
    // Loop all users
    for (User u : db.getAllUsers()) {
        if (loginId.equals(u.getId())) {
            if (verifyPassword(password,u.getPassword())) {
                // First Login must change password
                if (u.getFirstLogin()) {
                    System.out.println(x:"This is your first time logging in");
                    changePassword(u, password);
                    u.setFirstLoginFalse();
                }
                System.out.println(x:"Successful login!");
                return u;
            }
        }
    }
    System.out.println(x:"Unsuccessful login, please try again");
    return null;
}
```

1.5.3 Extra Feature 2: Billing Patients

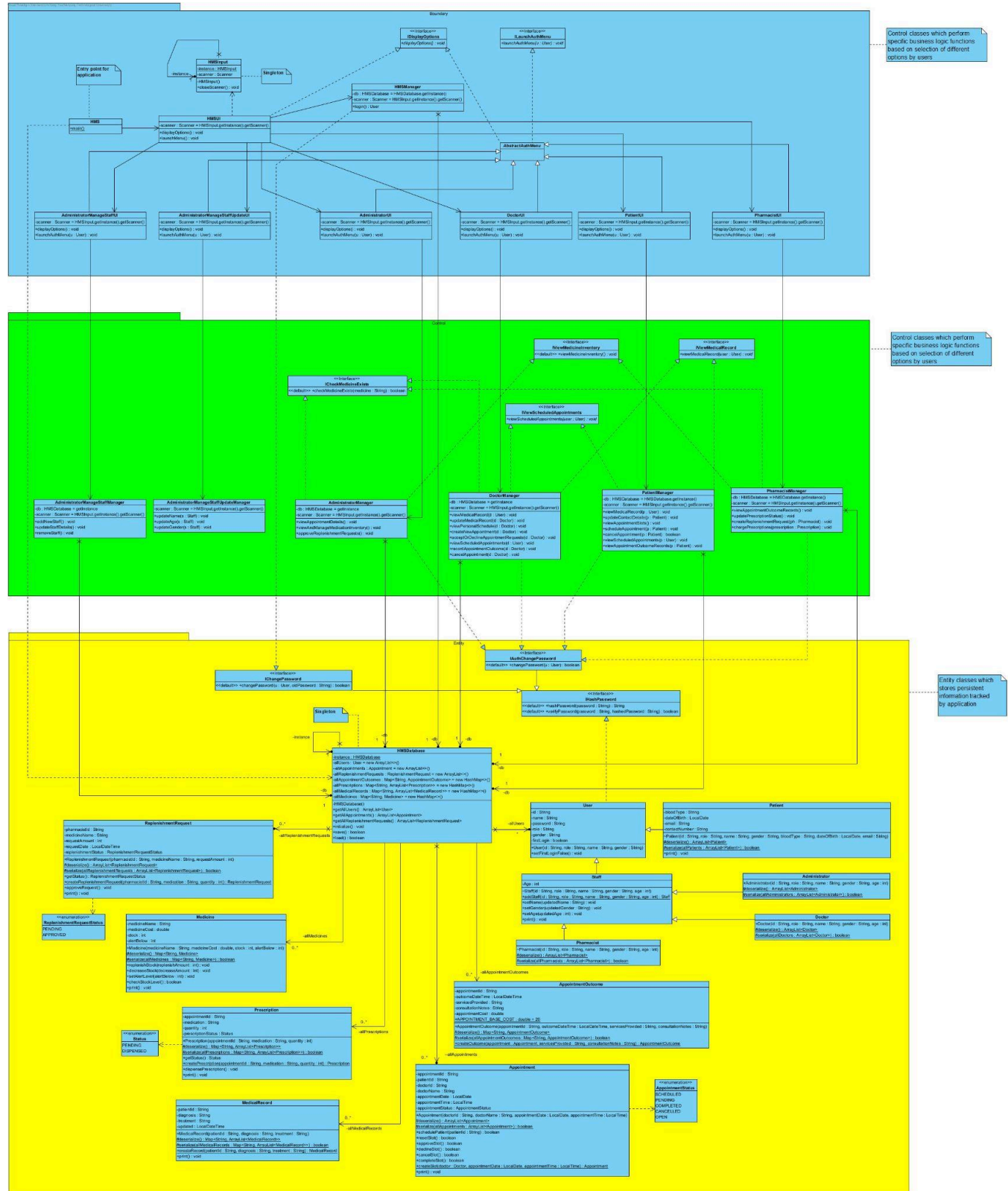
A HMS System requires a way to charge the patients with a cost amount after each Appointment has been completed. We have added methods and logic to calculate and update the cost of each appointment made by patients in the HMS System as well as different costs provided for the medication list in our HMSDatabase initialization method.

```
public class AppointmentOutcome implements Serializable {
    private String appointmentId;
    private LocalDateTime outcomeDateTime;
    private String servicesProvided;
    private String consultationNotes;
    private double appointmentCost;
    public static final double APPOINTMENT_BASE_COST = 20;
```

2. UML Class Diagram

(Please check GitHub Link for Actual High-Resolution Class Diagram)

<https://github.com/KY-LIM21/SC2002-OOP>



3. Testing

3.1 Schedule an Appointment

```
7. View Scheduled Appointments
8. View Past Appointment Outcome Records
9. Logout
Enter your choice: 7
----YOUR SCHEDULED APPOINTMENT SLOTS----
SLOT 1
Doctor: John Smith
DoctorId: D001
PatientId: P1001
Date: 10/03/24
Time: 10:30
Status: SCHEDULED
```

Patient successfully schedules a new appointment with Dr. John Smith

```
Enter your choice: 3
----AVAILABLE APPOINTMENT SLOTS----
SLOT 1
Doctor: John Smith
DoctorId: D001
Date: 10/03/24
Time: 10:30
Status: OPEN
SLOT 2
Doctor: John Smith
DoctorId: D001
Date: 10/04/24
Time: 10:30
Status: OPEN
SLOT 3
Doctor: Emily Clarke
DoctorId: D002
Date: 10/03/24
Time: 10:30
Status: OPEN

Enter your choice: 3
----AVAILABLE APPOINTMENT SLOTS----
SLOT 1
Doctor: John Smith
DoctorId: D001
Date: 10/04/24
Time: 10:30
Status: OPEN
SLOT 2
Doctor: Emily Clarke
DoctorId: D002
Date: 10/03/24
Time: 10:30
Status: OPEN
```

The selected time slot for Dr. John Smith becomes unavailable after the patient schedules the 10.30 time slot on 10/03/24.

3.2 Cancel an Appointment

```
6. Cancel an Appointment
7. View Scheduled Appointments
8. View Past Appointment Outcome Records
9. Logout
Enter your choice: 6
Which doctor did you schedule your appointment with? (id):
D001
Which date did you schedule your appointment on? (dd/MM/yy):
10/03/24
Which timeslot did schedule your appointment on? (HH:mm):
10:30
Appointment slot successfully cancelled
```

Patient successfully cancel an existing appointment

```
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. Logout
Enter your choice: 3
----AVAILABLE APPOINTMENT SLOTS----
SLOT 1
Doctor: John Smith
DoctorId: D001
Date: 10/03/24
Time: 10:30
Status: OPEN
```


Status of the cancelled time slot becomes “OPEN” again and is reflected in the available appointment slots

3.3 Update 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. Logout
Enter your choice: 2
For which patient do you want to update records? (patient id):
P1001
Diagnosis:
Lung Cancer
Treatment:
Chemotherapy
Medical record successfully updated
```

Doctor successfully updates patient’s medical record

```
----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. Logout
Enter your choice: 1
----YOUR MEDICAL RECORD----
Id: P1001
Role: Patient
Name: Alice Brown
Gender: Female
Blood Type: A+
DOB: 14/05/80
Email: alice@example.com
Contact: 12345678
Updated: 17/11/24 14:44:59
Diagnosis: Lung Cancer
Treatment: Chemotherapy
```

The updated patient’s medical record is reflected back on their side

3.4 Accept or Decline Appointment Requests

```
5. Accept or Decline Appointment Requests
6. View Upcoming Appointments
7. Record Appointment Outcome
8. Logout
Enter your choice: 5
----YOUR PENDING APPOINTMENT SLOTS----
SLOT 1
Doctor: John Smith
DoctorId: D001
PatientId: P1001
Date: 10/03/24
Time: 10:30
Status: PENDING
Accept appointment? (y/n)
y
Appointment accepted
```

Doctor successfully accepts an appointment request from a patient

```
6. View Upcoming Appointments
7. Record Appointment Outcome
8. Logout
Enter your choice: 6
----YOUR UPCOMING APPOINTMENT SLOTS----
Doctor: John Smith
DoctorId: D001
PatientId: P1001
Date: 10/03/24
Time: 10:30
Status: SCHEDULED
```

Appointment status for patient reflects “SCHEDULED” once Doctor accepted the request

3.5 Record Appointment Outcome

```
7. Record Appointment Outcome
8. Logout
Enter your choice: 7
Which date is the appointment on? (dd/mm/yy):
10/03/24
Which timeslot is the appointment on? (H:mm):
10:30

What are the services provided?:
Diagnosis
What are the prescribed medications? (type 'n' to finish):
Painkillers
Medicine does not exist in the inventory.
This medication does not exist
What are the prescribed medications? (type 'n' to finish):
Paracetamol
What is the quantity of Paracetamol?
10
What are the prescribed medications? (type 'n' to finish):
n
Are there any consultation notes?:
Chemotherapy Recommended
Appointment outcome created
```

Doctor successfully record the outcome of a completed appointment

```
8. View Past Appointment Outcome Records
9. Logout
Enter your choice: 8
----YOUR APPOINTMENT OUTCOMES----
2024-11-17T14:49:47.958817900
Services Provided: Diagnosis
Prescriptions:
Medication: Paracetamol
Quantity: 10
Status: DISPENSED
Cost of Appointment: 22.0
Consultation Notes: Chemotherapy Recommended
```

The outcome of the appointment is updated on the patient's side

3.6 Dispensing Medicine to Patients

```
----PHARMACIST MENU----
1. View Appointment Outcome Records
2. Update Prescription Status
3. View Medication Inventory
4. Submit Replenishment Request
5. Logout
Enter your choice: 2

----PENDING PRESCRIPTIONS----

Prescription 1:
Medication: Paracetamol
Quantity: 10
Status: PENDING
Enter prescription number to dispense (1-1, 0 to quit): 1
Prescription dispensed
There are no pending prescriptions
```

Pharmacist approves a prescription by a doctor

```
8. View Past Appointment Outcome Records
9. Logout
Enter your choice: 8
----YOUR APPOINTMENT OUTCOMES----
2024-11-17T14:49:47.958817900
Services Provided: Diagnosis
Prescriptions:
Medication: Paracetamol
Quantity: 10
Status: DISPENSED
Cost of Appointment: 22.0
Consultation Notes: Chemotherapy Recommended
```

Prescription status is updated and reflected in the patient's records

3.7: Medicine Replenishment

```
4. Submit Replenishment Request
5. Logout
Enter your choice: 4
Which medicine do you want to submit a request for?:
Paracetamol
This medicine does not have low stock level
```

Replenishment requests not allowed for medicines with a healthy stock level

```
3. View Medication Inventory
4. Submit Replenishment Request
5. Logout
Enter your choice: 3

---MEDICATION INVENTORY---
Medicine Name: Paracetamol
Stock: 90
Stock Level: Low
Medicine Name: Ibuprofen
Stock: 50
Stock Level: Healthy
Medicine Name: Amoxicillin
Stock: 75
Stock Level: Healthy
```

Test will be conducted on Paracetamol, low stock level = 90

```
---PHARMACIST MENU---
1. View Appointment Outcome Records
2. Update Prescription Status
3. View Medication Inventory
4. Submit Replenishment Request
5. Logout
Enter your choice: 4
Which medicine do you want to submit a request for?:
Paracetamol
How many units of Paracetamol do you want to replenish?:
10
```

Replenishment requests can now be submitted by the pharmacist since stock level is low

```
Successful login
---ADMINISTRATOR MENU---
1. View and Manage Hospital Staff
2. View Appointments Details
3. View and Manage Medication Inventory
4. Approve Replenishment Requests
5. Logout
Enter your choice: 4

---PENDING REQUESTS---

Request 1:
Medication: Paracetamol
Request Amount: 10
Request Date: 17/11/24 15:42:39
Request By: P001
Status: PENDING
Enter request number to approve (1-1, 0 to quit): 1
Request approved
Stock for Paracetamol has been replenished with 10 units
There are no pending requests
```

Administrator approves the replenishment request submitted by the Pharmacist

```
3. View and Manage Medication Inventory
4. Approve Replenishment Requests
5. Logout
Enter your choice: 3

---MEDICATION INVENTORY---
Medicine Name: Paracetamol
Stock: 100
Stock Level: Healthy
Medicine Name: Ibuprofen
Stock: 50
Stock Level: Healthy
Medicine Name: Amoxicillin
Stock: 75
Stock Level: Healthy
```

The stock for Paracetamol is now at 100 and the stock level is healthy as reflected

4. Reflection & Future Improvements

4.1.1 Lessons Learnt

When we were starting out this project, we ran into many problems especially since many data management modules and common serialization and persistence methods (JSON, SQL) that we were familiar with were not available to us. Furthermore, ensuring data persistence and interweaving it with OOP concepts was a challenging task because it required bridging the gap between object-oriented design principles and relational database concepts that we were more familiar with, especially with our Appointment entity. However, we eventually decided to also use our knowledge on relational concepts and settled on using a mock Database class that handled all the data structures we needed with information retrieval being done through its methods which our other Manager classes can call. Furthermore, we also optimized some of our data structures with Maps to ensure better performance, especially with looking up certain data as well as employing java streams when we absolutely need to loop through Lists, enhancing the readability of our data handling code as well as having the benefit of being parallelizable allowing our HMS to scale.

4.1.2 Future Improvement : A More Principled Approach to Data Handling

Our codebase definitely can be improved on, one specific future improvement could be on the way we are handling data. Instead of a huge HMSDatabase class, we could instead split it up into many specialised different entity Databases that would each manage data for their own entities and have the HMSDatabase class only responsible for data serialisation at the end of the HMS session. This would satisfy the Single Responsibility Principle as each specialised database would only carry the data management for its respective entity. It would also follow the Open/Closed Principle as new data management classes could be implemented from a master database interface, allowing for the addition of new databases for new roles.(i.e Nurse Databank)

Another way we can improve is to further segregate the methods in each of our RoleManagers (i.e. DoctorManager) by creating more Single Responsibility interfaces. Although we have already created interfaces for re-used or similar methods, further doing this for even currently non-reused methods would allow each function to be individually implemented by future new RoleManagers via multiple interfacing should there be more extensions made to the HMS. (i.e. add new staff roles).