# SC2002 Object Oriented Design & Programming

**Group Final Project**

**Hospital Management System (HMS)**

**GROUP 2**

**SCEB**

## 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 | Signature /Date |
|---|---|---|---|
| Kow Zi Ting | SC2002 | SCEB | [illegible] 16/11/2024 |
| Law Yu Chen | SC2002 | SCEB | [illegible] 19/11/2024 |
| Manasi Singh | SC2002 | SCEB | [illegible] 19/11/2024 |
| Odilia Kathleen Gunawan | SC2002 | SCEB | Odilia Kathleen 15/11/2024 |
| Timothy Lim Wei En (Lin Wei'En) | SC2002 | SCEB | Tim 18/11/2024 |

**Table of Contents**

## 1. Design Considerations

The Hospital Management System (HMS) is built to ease the access of information for hospital staff and patients alike.

### 1.1 Approach

Users of the system are first classified into 2 categories: Staff or Patient. The Staff class includes Doctor, Pharmacist, and Administrator. Furthermore, both Staff and Patient inherits from a User class. In addition, we made use of an Appointment class to store appointment details and patient's medical history. Lastly, we created the Medicine class to keep track of the list of medicine and its inventory status. StaffInv, PatientInv and MedicineInv include the usage of a Singleton pattern.

### 1.2 Assumption

Some of the assumptions made when creating our system:

- Doctors are available for appointments during work hours unless specified otherwise
- Each appointment slot lasts an hour
- Doctors only work 7 hours a day, starting at 9 am.
- Appointments are only available to be booked a week (7 days) in advance.

### 1.3 Design Principles

#### 1.3.1 Usage of OO Concepts and Principles

Abstraction

First, we did abstraction to ensure that there was no redundancy within classes and all methods and attributes were fully relevant to the required functions of the HMS. The details of patients and staff members provided in Patient_List and Staff_List guided us in our initial declaration of attributes for our role classes. Further attributes and methods were then added as the classes required.

For example, Doctor contains the Schedule attribute to store Patients' appointments as well as keep track of each doctor's availability.

Encapsulation

Encapsulation is implemented in the HMS across the User and Staff hierarchies and their subclasses, Administrator, Doctor, Patient and Pharmacist. All attributes in User and Staff are declared private. Hence, attributes in their subclasses to be declared private as well, since they utilise the base class constructors.

This ensures that the internal state of our classes is not directly exposed to the external interface. Instead, access and modifications to the attributes of created objects are moderated through our accessors and mutators. This is demonstrated through the way setAvailability() in the Doctor class checks for valid date and time inputs before allowing changes to be made to the schedule. In addition, the fact that our classes interact with our collections StaffInv, PatientInv and MedicineInv through singletons demonstrates encapsulation as well, since the data in the collections cannot be directly accessed or modified.

Inheritance

In the HMS, inheritance is demonstrated through our hierarchy of classes starting from User and Staff. We identified that all users shared certain commonalities: userID, password and role. Further, all users would have to log in to the system and change their password on the first login. This thus formed the attributes and methods of our base class, User.

We then adopted the same process to create Staff as a child of User and parent of Doctor, Pharmacist and Administrator, as we realised that members of the staff information shared commonalities absent in Patient, like age as reflected in the provided Staff_List. This would also make it possible to subsume them under the same StaffInv.

Polymorphism

Our system made use of polymorphism in the form of method overriding. In Doctor, Pharmacist, and Administrator, the changeName, changeAge, changeGender, and changeRole methods override the same method of their parent class, the Staff class. This allows the respective methods in Administrator, Doctor and Pharmacist to be dynamically bound to the function calls in HospitalApp.

### 1.3.2 Design Principles

Single Responsibility Principle

The single responsibility principle ensures that one module is only responsible for a single function, allowing for an easier modification and extension in the future. We made use of this principle in our code, particularly in the implementation of StaffInv, PatientInv, and MedicineInv. These 3 classes serve only one responsibility each, to generate our ArrayList from the respective .csv files. This is to ensure that we have ease of updating our classes to generate this ArrayList without the need to change any other classes, for example in the case of a change in file extension.

Open/Closed Principle

Open for extension: Our system can be easily extended to serve different purposes. For example, child classes like Neurosurgeon or Paediatrician can be easily added to Doctor, allow us to account for doctors' specialisations. Each of these classes will be able to use the inherited methods in Doctor seamlessly, without modifying the existing Doctor class.

Closed for modification: As mentioned in Encapsulation, each class's attributes are set to private and access and modification to them is strictly controlled by accessors and mutators.

Liskov Substitution Principle

In our code, our consideration of the Liskov Substitution Principle has guided us to ensure that the preconditions of overridden methods in child classes do not exceed those of the base class, and the postconditions do not fall short of those of the base class. This ensures that our User.login method can be used for Doctor, Patient, Administrator and Pharmacist objects alike.

Interface Segregation Principle

Within the Appointments class, prescriptions are available to be added and edited by the Doctor, as one would expect due to it being part of the outcome of the appointment. Pharmacists only need to work on the prescription portion of the Appointment; thus, the pharmacist only uses methods related to prescribing medication, an implementation of the Interface Segregation Principle.

Dependency Injection Principle

Even though there is tight coupling in many of our functions, especially to the main HospitalApp, loose coupling and thus the Dependency Injection Principle exists within our program.

### 1.3.3 Design Considerations

do-while Loops

A majority of our code in the HospitalApp class is governed by do-while loops. The do-while loops ensure that each menu will be displayed for user selection at least once, and error input from users can be comfortably managed by simply re-prompting for a valid selection.

Singleton pattern

The Singleton Pattern ensures that a class has only one instance and provides a global access point to it, making it ideal for managing shared resources like in MedicineInv, StaffInv and PatientInv. It offers controlled access to a single instance, reduces namespace pollution, and supports easy initialization to optimize resource usage. A properly implemented Singleton can be thread-safe and improve resource management by avoiding multiple instances competing for the same resource. However, it can lead to tight coupling and difficulties in unit testing due to its global nature.

Integer checking

For the sake of robustness, when integers are expected as input, it is necessary to ensure only integers are accepted. This prevents arbitrary errors from occurring, thus ensuring our system

### 1.3.4 Coupling and Cohesion

Coupling and cohesion are two important software design principles that affect the quality and maintainability of the code. Coupling refers to the degree of interdependency between modules/classes in a system. It is a measure of how much a module relies on the other. Cohesion refers to how focused a module is one a single purpose/function. It is the measure of how strongly the elements in a module belong together. For good design, a system needs to have low coupling and high cohesion.

An instance of low coupling in our system are key classes like Appointment and Medicine that have no dependencies on other classes. This minimizes its reliance on other classes so it can be maintained easily and changes in other classes does not cause cascaded changes in our key classes. There are classes in our system however, that do have a high number of dependencies on other classes, and this is an area we can improve on.

An instance of high cohesion in our system are classes like StaffInv, PatientInv, and MedicineInv. These are classes that bear only one responsibility, which is to generate arraylists from the csv files as mentioned above. By following the Single Resposibility Principle in our classes, we can ensure that they are focused on a single purpose, increasing cohesion of the classes.

### 1.3.5 Trade Offs

Simplicity for Complexity

For greater ease of implementation, our team has sustained the trade-off of simplicity for complexity. Instead of having only one responsibility, many of our classes assume multiple roles. While this makes each class more complex, it allows for direct synchronisation of data where the distinct responsibilities are connected and makes our code easier to understand.

## 1.4 Highlighted Features

Some notable features in our system:

- CSV file import: Initial user data are imported from a CSV file and implemented as a Singleton pattern, eliminating the need for an administrator to create and add new users to the system one-by-one. This also ensures that the entire program uses the same imported ArrayLists, with usage of the Singleton pattern, preventing information persistence throughout the changes and addition of data.
- Change of password: Users are required to change their password upon first login and are given the option to change their password anytime, not just during their first login.
- Password confirmation: When changing their password, users are required to confirm their new password to avoid mistakenly putting the wrong password.
- User input: When providing an input to the system, we make use of a number or ID to prevent mistyping of names and ease of input.
- Invalid input: When user input appears to be invalid, system will ask users to re-enter the input. If users have entered the wrong choice, they also have the option to return to the previous menu.
- Change role: In the current implementation of HMS, since there are only four roles – Administrator, Pharmacist, Doctor and Patient – with no transferable skills between them, changing between these roles would be unrealistic. However, we still decided to include the Change Role function for a different purpose. Under the HMS, should any member of Staff go on long-term leave or no-pay leave, they should be unable to access records in the HMS for confidentiality purposes (like how students on LOA should not be able to access NTULearn). Thus, administrators can change the role of these staff to

"On Leave" to make it impossible for them to access the HMS and reinstate them only when their leave period has ended.

- Editing of medication fields: The administrator could alter details of any medication and even add new ones if needed.

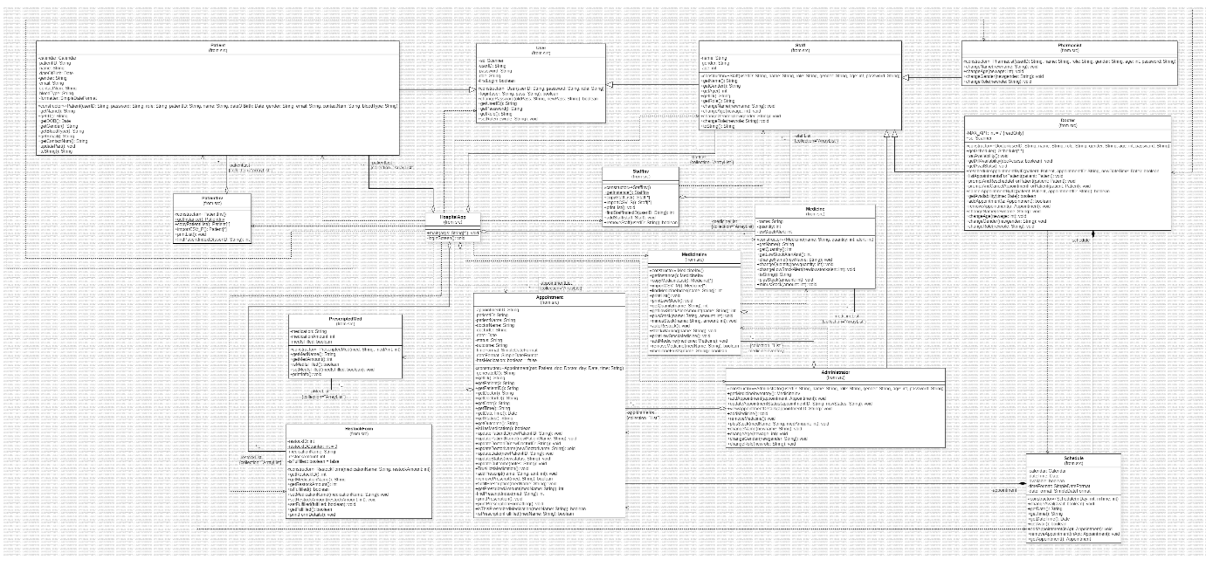
## 1.5 Proposed Expansion

There are several ways to expand the system further:

- Accounting department: Implementation of this department will allow staff to generate bills from appointments and medication. Patients are also able to access their unpaid and past bills from the system.
- Specialization: Adding specialization to doctors, allowing for a referral. Patients could also choose to make an appointment according to the type of doctors they needed.
- Appointment and Prescription printout: With the details of the appointment, its outcomes and the prescriptions, the ability to be printed has been implemented already. With that functionality, it is possible to print out all the details of the appointment or Prescription as a physical note with the addition of a printer peripheral or as a text file.

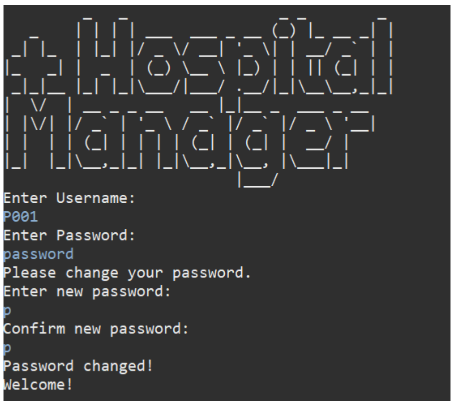
## 2. UML Class Diagram

## 2.1 Diagram

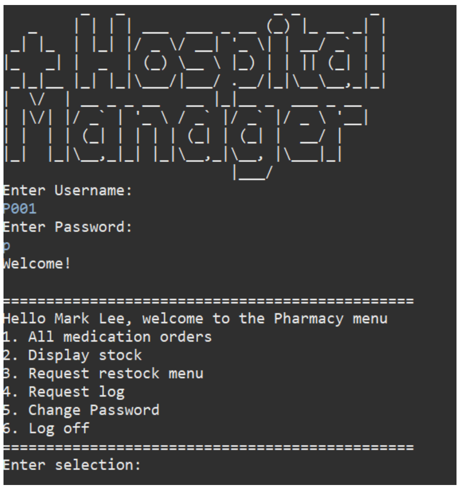


## 3. Tests

Upon first login, users will be prompted to change their password. On subsequent logins, users will be required to make use of their new password.



```
Enter Username:
P001
Enter Password:
password
Please change your password.
Enter new password:
p
Confirm new password:
p
Password changed!
Welcome!
```

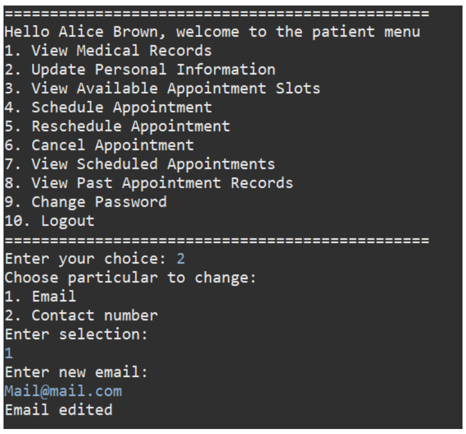


```
Enter Username:
P001
Enter Password:
p
Welcome!

============================================
Hello Mark Lee, welcome to the Pharmacy menu
1. All medication orders
2. Display stock
3. Request restock menu
4. Request log
5. Change Password
6. Log off
============================================
Enter selection:
```

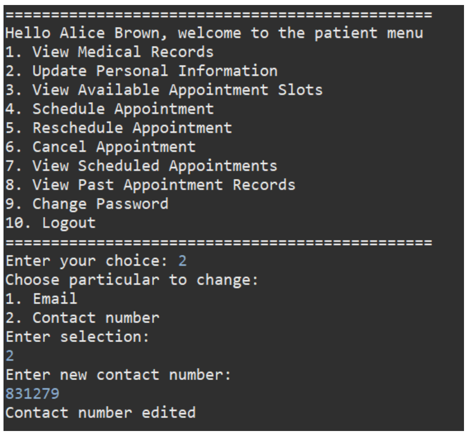
### 3.1 Patient Actions

#### 3.1.1 Update Personal Information

Patients can also change their contact information through the second selection in the menu. They can alter their email address and contact number.

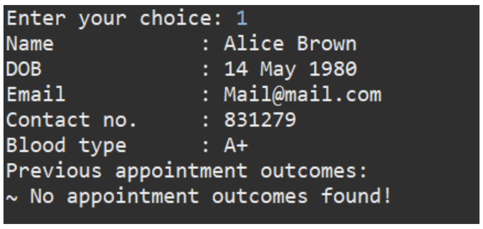


```
==============================================
Hello Alice Brown, welcome to the patient menu
1. View Medical Records
2. Update Personal Information
3. View Available Appointment Slots
4. Schedule Appointment
5. Reschedule Appointment
6. Cancel Appointment
7. View Scheduled Appointments
8. View Past Appointment Records
9. Change Password
10. Logout
==============================================
Enter your choice: 2
Choose particular to change:
1. Email
2. Contact number
Enter selection:
1
Enter new email:
Mail@mail.com
Email edited
```



```
==============================================
Hello Alice Brown, welcome to the patient menu
1. View Medical Records
2. Update Personal Information
3. View Available Appointment Slots
4. Schedule Appointment
5. Reschedule Appointment
6. Cancel Appointment
7. View Scheduled Appointments
8. View Past Appointment Records
9. Change Password
10. Logout
==============================================
Enter your choice: 2
Choose particular to change:
1. Email
2. Contact number
Enter selection:
2
Enter new contact number:
831279
Contact number edited
```

The change is also reflected if patient choose to view their records.

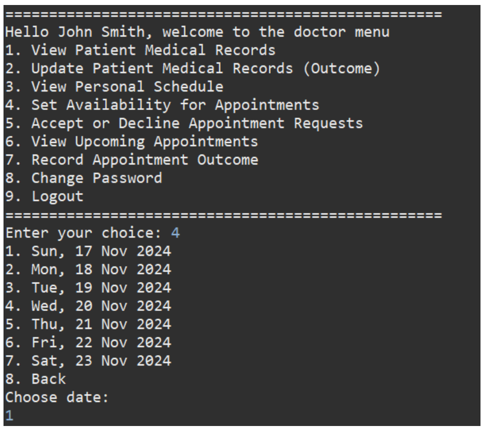


```
Enter your choice: 1
Name            : Alice Brown
DOB             : 14 May 1980
Email           : Mail@mail.com
Contact no.     : 831279
Blood type      : A+
Previous appointment outcomes:
~ No appointment outcomes found!
```

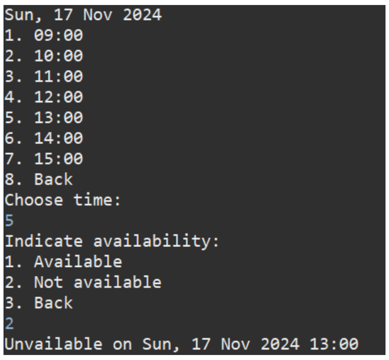
## 3.2 Doctor Actions

### 3.2.1 Set Availability for Appointments

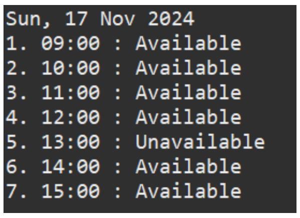
Doctors can set their availability through option 4. They are required to choose the designated date, then time, before indicating availability. The changes are reflected in their personal schedule.



```
==================================================
Hello John Smith, welcome to the doctor menu
1. View Patient Medical Records
2. Update Patient Medical Records (Outcome)
3. View Personal Schedule
4. Set Availability for Appointments
5. Accept or Decline Appointment Requests
6. View Upcoming Appointments
7. Record Appointment Outcome
8. Change Password
9. Logout
==================================================
Enter your choice: 4
1. Sun, 17 Nov 2024
2. Mon, 18 Nov 2024
3. Tue, 19 Nov 2024
4. Wed, 20 Nov 2024
5. Thu, 21 Nov 2024
6. Fri, 22 Nov 2024
7. Sat, 23 Nov 2024
8. Back
Choose date:
1
```



```
Sun, 17 Nov 2024
1. 09:00
2. 10:00
3. 11:00
4. 12:00
5. 13:00
6. 14:00
7. 15:00
8. Back
Choose time:
5
Indicate availability:
1. Available
2. Not available
3. Back
2
Unvailable on Sun, 17 Nov 2024 13:00
```

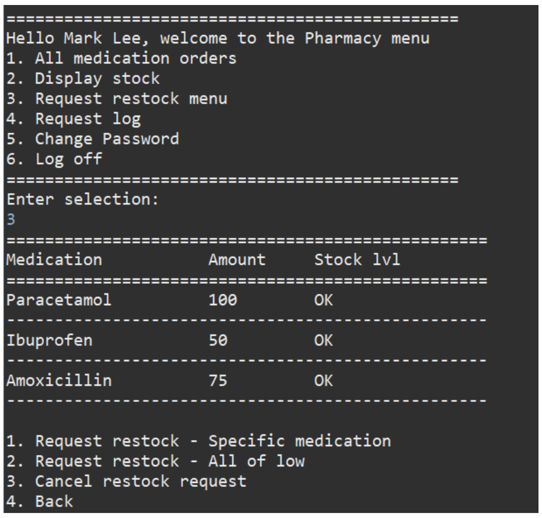


```
Sun, 17 Nov 2024
1. 09:00 : Available
2. 10:00 : Available
3. 11:00 : Available
4. 12:00 : Available
5. 13:00 : Unavailable
6. 14:00 : Available
7. 15:00 : Available
```

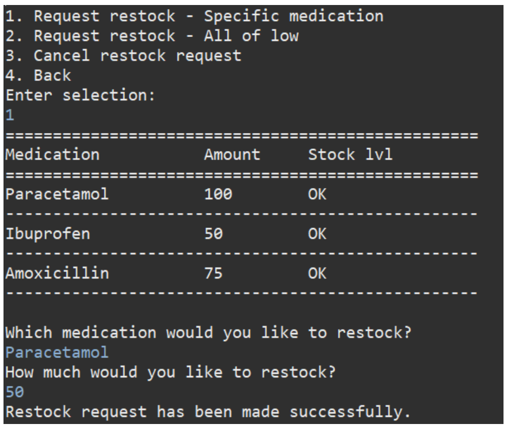
## 3.3 Pharmacist Actions

### 3.3.1 Submit Replenishment Request

If the pharmacist noticed that there is a medication that is low on stock, they can request a restock through the 3rd menu option. They are then prompted to choose between requesting restock for all medications that are low in stock or selecting a certain medication, then the amount to restock. After which, the request will be submitted pending administrator's approval.



```
==============================================
Hello Mark Lee, welcome to the Pharmacy menu
1. All medication orders
2. Display stock
3. Request restock menu
4. Request log
5. Change Password
6. Log off
==============================================
Enter selection:
3
=================================================
Medication          Amount     Stock lvl
=================================================
Paracetamol         100        OK
-------------------------------------------------
Ibuprofen           50         OK
-------------------------------------------------
Amoxicillin         75         OK
-------------------------------------------------

1. Request restock - Specific medication
2. Request restock - All of low
3. Cancel restock request
4. Back
```

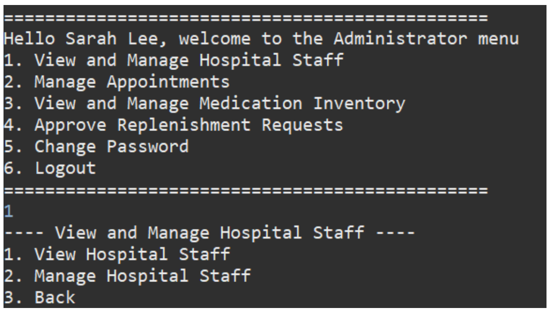


```
1. Request restock - Specific medication
2. Request restock - All of low
3. Cancel restock request
4. Back
Enter selection:
1
=================================================
Medication          Amount     Stock lvl
=================================================
Paracetamol         100        OK
-------------------------------------------------
Ibuprofen           50         OK
-------------------------------------------------
Amoxicillin         75         OK
-------------------------------------------------

Which medication would you like to restock?
Paracetamol
How much would you like to restock?
50
Restock request has been made successfully.
```

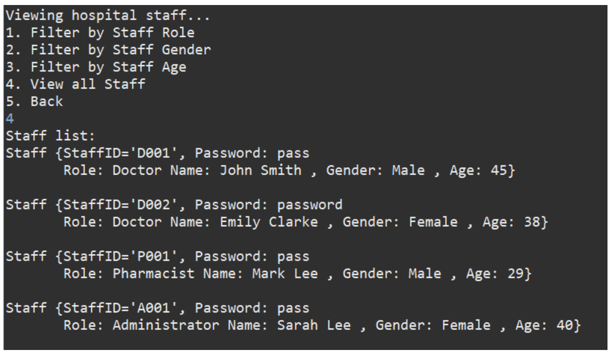
## 3.4 Administrator actions

### 3.4.1 View and Manage Staff

Administrators can view and manage hospital staff through the first option in the menu. If they choose to view staff, there also have the option to filter staff through certain attributes or view all staff.

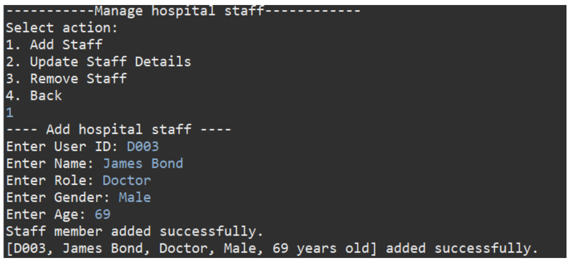


```
==============================================
Hello Sarah Lee, welcome to the Administrator menu
1. View and Manage Hospital Staff
2. Manage Appointments
3. View and Manage Medication Inventory
4. Approve Replenishment Requests
5. Change Password
6. Logout
==============================================
1
---- View and Manage Hospital Staff ----
1. View Hospital Staff
2. Manage Hospital Staff
3. Back
```

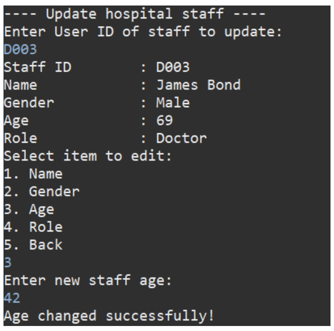


```
Viewing hospital staff...
1. Filter by Staff Role
2. Filter by Staff Gender
3. Filter by Staff Age
4. View all Staff
5. Back
4
Staff list:
Staff {StaffID='D001', Password: pass
       Role: Doctor Name: John Smith , Gender: Male , Age: 45}

Staff {StaffID='D002', Password: password
       Role: Doctor Name: Emily Clarke , Gender: Female , Age: 38}

Staff {StaffID='P001', Password: pass
       Role: Pharmacist Name: Mark Lee , Gender: Male , Age: 29}

Staff {StaffID='A001', Password: pass
       Role: Administrator Name: Sarah Lee , Gender: Female , Age: 40}
```

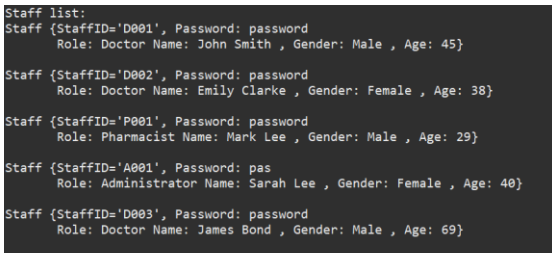
Furthermore, admins are able to add staff, edit their detail, or remove staff through the manage staff menu.



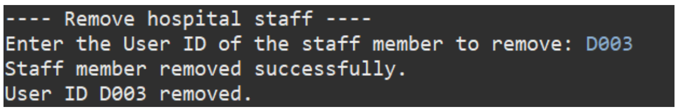
```
-----------Manage hospital staff------------
Select action:
1. Add Staff
2. Update Staff Details
3. Remove Staff
4. Back
1
---- Add hospital staff ----
Enter User ID: D003
Enter Name: James Bond
Enter Role: Doctor
Enter Gender: Male
Enter Age: 69
Staff member added successfully.
[D003, James Bond, Doctor, Male, 69 years old] added successfully.
```



```
---- Update hospital staff ----
Enter User ID of staff to update:
D003
Staff ID        : D003
Name            : James Bond
Gender          : Male
Age             : 69
Role            : Doctor
Select item to edit:
1. Name
2. Gender
3. Age
4. Role
5. Back
3
Enter new staff age:
42
Age changed successfully!
```



```
Staff list:
Staff {StaffID='D001', Password: password
       Role: Doctor Name: John Smith , Gender: Male , Age: 45}

Staff {StaffID='D002', Password: password
       Role: Doctor Name: Emily Clarke , Gender: Female , Age: 38}

Staff {StaffID='P001', Password: password
       Role: Pharmacist Name: Mark Lee , Gender: Male , Age: 29}

Staff {StaffID='A001', Password: pas
       Role: Administrator Name: Sarah Lee , Gender: Female , Age: 40}

Staff {StaffID='D003', Password: password
       Role: Doctor Name: James Bond , Gender: Male , Age: 69}
```



```
---- Remove hospital staff ----
Enter the User ID of the staff member to remove: D003
Staff member removed successfully.
User ID D003 removed.
```

## 4. Reflection

Initial working process, we tried to simply divide the work of creating each class of user with its own implementation according to what the user can do in their own individual menu.

However, we quickly realised there is often a need for a user to interact with another class outside its own. For example, the need for doctor to access a patient's medical records or information, patients need to see the availability of doctors, administrators need to see replenishment requests sent by pharmacists, etc. Hence, we had to reformulate our plan.

By finding the common functionalities needed in each user's menu, we were able to finally decide what methods to put in each user's class. This also makes the system easy to modify and comprehensive.

We recognise the gaps in our knowledge and experience that causes some parts of our system to not be completely effective or clear. For example, our program is still very tightly coupled. This is something that we can strive to achieve in the future.

## 5. Links

**Github:** https://github.com/Katstengels/2002-Assignment.git

**Test Cases:** https://github.com/Katstengels/2002-Assignment/blob/Final/Documents/Test%20Cases.pdf

**UML:** https://github.com/Katstengels/2002-Assignment/blob/Final/Documents/UML%20pdf%20v2.pdf