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	Lab Group	Signature	Date
ALAN LEE LEMAN	SC2002	SCS6		17/11/24
CHEONG KEAT JIN, DAVID	SC2002	SCS6		17/11/24
PHAM NGUYEN VU HOANG	SC2002	SCS6	Hoang	17/11/24
SEAN LENG WEI XUAN	SC2002	SCS6		17/11/24
YONG JUN HAN	SC2002	SCS6		17/11/24

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.

1.0.   Design Considerations

The Hospital Management System is an application aimed at automating the management of hospital operations for patients, staff, inventory and appointments. It facilitates the efficient management of hospital resources, enhances patient experience, and streamlines administrative processes. Considering that most users will not have technical knowledge of the system, we have designed it with extensive features and ease-of-use in mind. Hence, users are able to perform various operations on one system, reducing operational costs.

1.1.   Approach

We wanted to design a system that retains reusability, extensibility and maintainability. Hence, we tried our best to achieve loose coupling and high cohesion by following the SOLID principles wherever possible.

1.2.   Principles

Single Responsibility Principle

To achieve high cohesion, we avoided having classes with excessive functions. An example of this would be the AdministratorHandler class only handling logic specific to administrators, such as approving replenishment requests. It then depends on the <<IMedicineHandler>> interface to help it fetch and process replenishment requests. This separation of duties follows SRP by offloading responsibilities for medicine handling to another class/interface. Doing so allows for reusability and makes it easier to analyse and modify the code.

Open-Closed Principle

In the system, we use interfaces, such as <<IAppointmentHandler>>, <<IDoctorHandler>>, so that the system is open to extension but closed to modification. These interfaces can be reused through inheritance but their implementation does not need to be. This means that newly created classes can implement these interfaces without changing the existing source code. This shows that the system adheres to OCP.

Liskov Substitution Principle

To follow LSP, we ensured that subclasses could perform as their respective superclasses perform without impacting the functionality of the system. In the system, Doctor extends Staff, inheriting its core properties and behaviours. Since Doctor does not alter the contract of Staff, it can be substituted wherever a Staff object is expected. Hence, modifications made in subclasses do not compromise the functionality of their respective superclasses and this helps to achieve system reliability.

Interface Segregation Principle

We wanted interfaces to only contain methods relevant to the specific implementing class so as to prevent "fat interfaces", which can lead to unnecessary coupling and reduced flexibility. For example, the system uses the interface <<IPatientHandler>>, which is specific to patient-related operations, such as addPatient( ), findPatientById( ), updateContactInfo( ) and viewMedicalRecord( ). As shown, PatientHandler implements <<IPatientHandler>> without unnecessary dependencies on appointment or medicine-related methods and follows ISP.

Dependency Injection Principle

To achieve loose coupling, we wanted high-level and low-level modules to depend on a common abstraction. As the low-level module implements the abstraction, allowing the high-level module to remain agnostic to the implementation. In the system, dependency inversion is applied using interfaces such as in the case of <<IPatientHandler>>. This allows DoctorHandler, a high-level class, to depend on this abstraction instead of the concrete implementation of PatientHandler.

1.3. Assumptions

We assumed that if a staff member were to go to the hospital as a patient, they would be given a new ID so that their roles would not conflict with each other. This would allow them to access different menus according to their needs, but through separate IDs.

We assumed that staff members and patients are already assigned to the Hospital Management System to better facilitate demonstration and testing.

We assumed that there will be no need to update and return the values of medicine, staff details, patient details to our database. Meaning that the values will not be written to the database files which we read from initially.

2.0.   UML Class Diagram
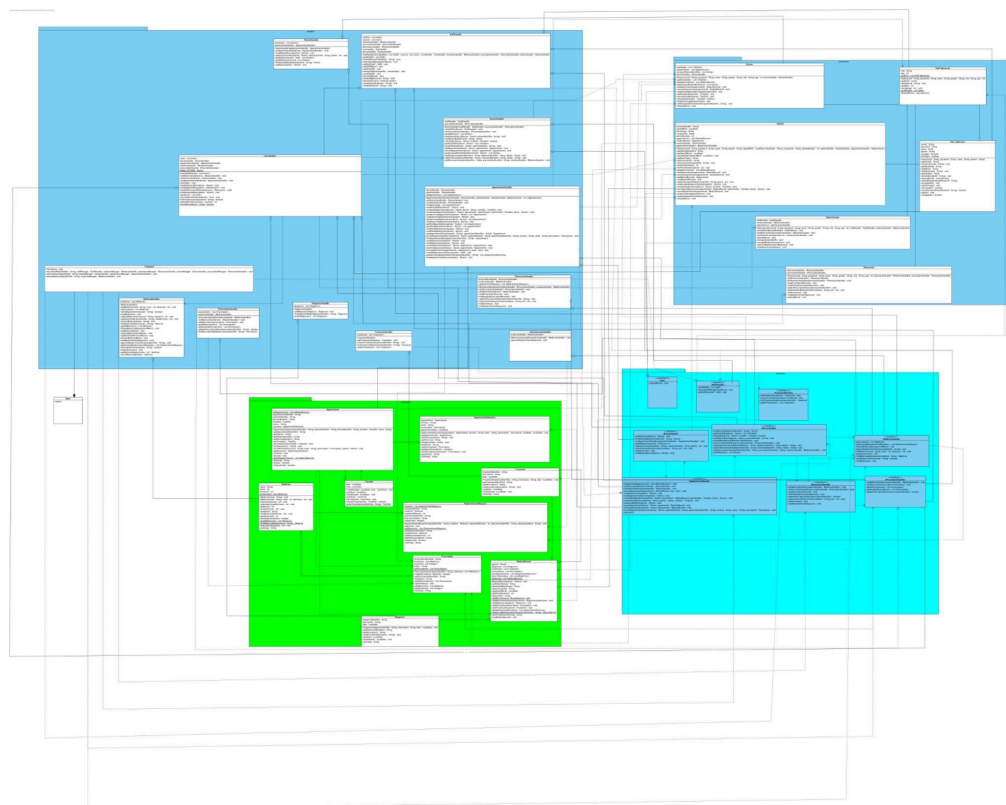
interfaces	userclasses	resources	handlers
IUser	Administrator	Appointment	AdministratorHandler
IStaffHandler	Doctor	AppointmentOutcome	AppointmentHandler
ITreatmentHandler	Patient	Diagnosis	DiagnosisHandler
IPatientHandler	Pharmacist	MedicalRecord	DoctorHandler
IDoctorHandler	Staff	Medicine	MedicineHandler
IMedicineHandler	User	Prescription	PatientHandler
IPrescriptionHandler		ReplenishmentRequest	PharmacistHandler
IPharmacistHandler		TimeSlot	PrescriptionHandler
IAppointmentHandler		Treatment	StaffHandler
			TreatmentHandler
			TxtImport
			UserHandler

2.1.   Importing and Database Operations

Our system reads data from text files, converting the information into entity classes which are classified under *userclasses*.These entity classes hold the responsibility of data, without user interaction or logic implemented. Through this, they are able to encapsulate data of the system, only accessible through get and set methods.

2.2.   Main

This class handles the interaction between the user and the system. It serves as an entry point and enables users to interact via menus and inputs.

2.3.   Implementation of Role-specific Functions

We have opted to create a higher level class to manage the entity classes. These classes are found under *handlers* and are responsible for the logic and implementation. Such as *DoctorHandler* which deals with doctor-related operations, which interact with Doctor entities, Patient, etc. This ensures that there is no direct interaction between our *Main* class and entity classes.

2.4.   Dynamic and Engaging Interactive System

Our system encompasses a variety of user options, that upon selection, activate the respective role specific functions we have implemented to process user requests.

To prevent overcomplicating the input procedure for each of the different roles, we have implemented the use of indexes for user inputs. By displaying the options in the CLI itself, users can select each option using numbers. As such, we reduce the need for implementing a complicated input checking and error handling system to ensure that the inputs are correct.

3.0. Test Cases

	Test cases	Expected Outcome
Login Test Cases		
1	Login as Patient	Successfully log in to Patient ID (P1)
2	Login as Doctor	Successfully log in to Staff ID (S1)
3	Login with incorrect password	Unable to log in, prompted to retry
Patient Test Cases		
4	View Medical Records	Records of Patient (P1) displayed. Expected to be empty for diagnosis, treatments and past appointments.
5	Update Personal Information	User input of contact information and phone number updated to Medical Records of Patient (P1). Choice to leave unchanged.
6	View Available Appointment Slots	Display of available Doctors in this case Doctor (S1) and their available time slots. (Valid index): Expect to see 2 as we have pre-loaded them. (Enter incorrect index / 'E'): Exit the user from the menu and go back to Patient menu
7	Schedule an Appointment	Display of available Doctors and their available time slots. Able to choose from time slots to be scheduled. (Valid index): Appointment request will be sent to respective doctor and can be viewed from the Doctor Menu (Invalid index): Return to the Patient Menu
8	Reschedule an Appointment	Display of patient (P1) appointments, if any. Include details such as appointment ID, Time of appointment and Status. (Valid index) : Prompt user to input index of appointment intended for rescheduling. Display of Doctor's remaining available time slots and prompting for user to input index. Successful selection will reschedule the appointment to new time slot and re-open the previous timeslot. (Invalid index): Return to Patient Menu
9	Cancel an Appointment	Display of current appointments if any. (Has appointment): Display of appointment details. Prompt the user to select the appointment index intended for cancellation. Successful cancellation will free up the

		time slot on the Doctor's schedule and remove said appointment from the patient's current schedule. (No appointment): Either the request has not been approved by the doctor, or no request has been made.
10	View Scheduled Appointments	Display of appointments, if any. Include details such as appointment ID, time of appointment and status. (Has appointment): Display of appointment details. (No appointment): Either the request has not been approved by the doctor, or no request has been made.
11	View Past Appointment Outcome Records	Display of past appointment outcomes written by the doctor. Includes details such as: Appointment date, ID, services provided, prescription given and doctor notes.
12	Logout	Display a message indicating successful log out and returning to login page.
Doctor Test Cases		
13	View Patient Records	Display of patients assigned to doctor (S1), in this case patient (P1). Includes details such as: Patient particulars, diagnosis history, treatment history and past appointments. (Has patient): Display the details mentioned above (No patient): Display of error message indicating to patients assigned.
14	Update Patient Records	Display of patients currently assigned to doctor (S1), if any. (Has patient): Display of patient's name and patient ID. Selection of the patient's index will prompt the doctor to perform 3 actions: add diagnosis, add treatment and add prescription. Add Diagnosis: Doctor prompted to input Diagnosis ID, and diagnosis details. Details will be updated to the patient's medical records. Add Treatment: Doctor prompted to input treatment ID, treatment Details. Details will be updated to the patient's medical records. Add Prescription: Doctor prompted to select medicine + quantity based on the medicine available in the inventory. Prescription will then be added to the patient's medical records. (No patient): Display of error message indicating no patients. Returned to Doctor Menu.
15	View Schedule	Display of current available time slots of respective doctor, Doctor (S1) in this case. Upcoming appointments

		will also be displayed if any have been accepted by the doctor.
16	Set Availability	Prompts doctor to input the available date and time. Time slot will be added to the availability list for all patients to see.
17	Accept or Decline Appointment	Option for doctor to either accept or decline a specific appointment request made by patients

(Has request): Display of current appointment request, including the ID, time of appointment and status. Doctor will then be prompted to select the appointment request in question and either accept or decline.

Accept: if accepted, status will be changed to Confirmed and will be updated to patient's and doctor's upcoming scheduled appointments.

Decline: if declined, appointment will not appear in both the patient's and doctor's upcoming scheduled appointments.

(No request): Display of error message indicating no appointment requests. Return to the doctor menu. |
| 18 | View Upcoming Appointment | Display of list of upcoming appointments if any including details such as appointment ID, time and status |
| 19 | Record Appointment Outcome | Display of list of confirmed appointments that the doctor has accepted, if any.

(Has appointments): Prompts doctor to select appointment index, services provided, notes if any, and prescription if any. Prescription will prompt the doctor to input the medicine and quantity. Appointment Outcome will be recorded and updated in the patient's medical records.

(No appointments): Display of error message indicating no Confirmed appointments. |

<table>
<tr><td colspan="3" align="center">Pharmacist Test Cases</td></tr>
</table>

20	View Prescription Records	Display of the prescription records if any. Includes details of prescription ID, medicines prescribed and quantity, and the status of the prescription (dispensed or not).
21	View Pending Prescription Records	Display of pending prescription records that have not been completed yet.
22	Update Prescription Status	Display of prescriptions if any.

(Has prescriptions): Status will be changed to Dispensed, and the medicine will be deducted from the medicine Inventory.

(No prescriptions): Display of error message indicating |

		no prescription records.
23	View Inventory	Display of all medicines in the medicine inventory and their stock level. The alert level is an indication of whether to approve a request for stock replenishment. If current stock is below the level, the replenishment request will go through to the admin. Else, there will be no request submitted.
24	Submit Replenishment Request	Display of the current medicine inventory including the medicine name, and quantity in stock. Successful selection of the medicine index will prompt the pharmacist to input the amount to replenish and a notice of whether the request was successful or not will be submitted. (Above alert level): Replenishment request will be denied, indicating the current stock level and alert level. (Below alert level): Display notice of successful submission of request.
25	View Replenishment Request	Display of the replenishment request, if any. Includes the replenishment ID, Medicine requested and quantity, and status of the request.
Administrator Test Cases		
26	Manage Staff	Display of actions allowed for administrator namely: add Staff, Update Staff, Remove Staff, View All Staff, Filter Staff, and Return. Add Staff: Prompts admin to enter new user ID, password, Name, gender, role, and age. Successful notice will be displayed. Update Staff: Displays the list of all the current staff. Prompts admin to select the staff index, and change the details accordingly. Updated details will be added to the staff list. Remove Staff: Displays the list of all the current staff. Prompts the admin to select the staff index up for removal. Successful notice of removal will be displayed. View All Staff: Will display the list of all the current staff. Filter Staff: Will display the menu for which the admin can select from to filter the staff list by, namely: Role, Gender, Age, Show All. - Role: Display the list of doctors based on the role input by the admin. - Gender: Display the list of staff based on the gender input by the admin.

		- Age: Display the list of staff based on the age input by the admin. Show All: Displays all the staff. Includes details such as: ID, name, role, gender, and age.
27	Manage Medicine Stock	Displays the medicine management menu which includes the actions the admin can take such as: View Medicine, Add New Medicine, Update Medicine Stock, Update Stock Alert Level, Remove Medicine. View Medicine: Displays the entire list of the current medicine inventory including the names, stock, and alert level. Add New Medicine: Prompts the admin to input the new medicine name, quantity, and alert level. Successful notice will be displayed indicating the medicine name, quantity and alert level. Update Medicine Stock: Displays the entire medicine inventory and prompts the admin to select the medicine index to be updated. Prompts the admin to input the new quantity. Successful notice will be displayed indicating the medicine name and updated quantity. Update Stock Alert Level: Displays the entire medicine inventory and prompts the admin to select the medicine index to be updated. Prompts the admin to input the new alert level for said medicine. Successful notice will be displayed indicating the medicine name and updated alert level. Remove Medicine: Displays the entire medicine inventory and prompts the admin to select the medicine index to be removed. Successful notice will be displayed indicating the medicine name.
28	Approve Replenishment Request	Displays the list of replenishment requests submitted by the pharmacist, if any. (Has request): Prompts the admin to input the request index. Displays the updated quantity and respective medicine name. (No request): Displayed error message indicating no requests have been made.
29	View Appointment Details	Displays the list of all current appointments made, if any.

4.0   Reflections

4.1.   Difficulties Encountered

<u>Building from scratch</u>

It was difficult for us to conceptualise the entire hospital management system and design the appropriate structure from scratch. There were many features that we wanted or needed to incorporate, which led to numerous classes and complex interactions between them. To tackle this, we decomposed the system down into smaller, more manageable sub-systems by coming out with a rough overview of its main components. We looked at the various roles given in the manual and how they would need to interact with the system and each other. This allowed us to determine the necessary classes and the appropriate methods, leading to a smooth and coherent structure.

<u>Different programming styles</u>

To complete the project in a timely manner, each team member was tasked to code different components of the system. However, when we came together to combine our code and build the system, we faced many problems trying to understand and implement the code written by other team members. Hence, communicating with one another was very important and allowed us to understand and adapt to different programming styles within a short period of time.

<u>Unfamiliarity with Github</u>

Some team members were new to Github and were not familiar with its many features. This caused their workflow to be less efficient and took them more time to adapt and debug their code. This issue was alleviated when more experienced team members gave advice to them on how to set up and use Github, allowing them to operate more efficiently.

4.2. Knowledge Learnt

We learnt how to build a rudimentary database, which could read from and write to txt files so that data could be saved even when the program has ended. In doing so, we also learnt how to use hashtables in Java and how they helped with data organisation. Furthermore, we gained a better understanding of object-oriented principles concepts, and could convert our theoretical knowledge into practical application.

4.3. Further Improvements

We could improve the usability of our system by allowing users to change their password in case they forgot it. Under our current system, any subsequent password changes have to be done manually and on site, which is inconvenient and time-consuming for patients and staff.

Given more time, we could also better standardise programming styles and formats throughout the system to improve the readability of our code. This would make it easier for other people to understand code and implement additional features to our system.