



SC2002

SCMA Group3

Hospital Management
System

Aanya, Jervis, Ke Jun, Maeko, Viona



TABLE OF CONTENTS



01

PROJECT INTRO



02

CODE
IMPLEMENTATION



03

TEST CASES



04

CONCLUSION

BACK

NEXT

WHAT IS HMS

Why we need Hospital Management System:

- Provides a user-friendly interface to streamline hospital operations
- Enables different roles to efficiently fulfill their responsibilities
- Enhances coordination among staff through a centralised HMS
- Creates a smooth, user-centered hospital experience



BACK

NEXT

ROLES



Patient

- Information access
- Manage own appointments



Doctor

- Manage patients' medical records
- Appointment scheduling



Pharmacist

- Prescription management
- Inventory management



Administrator

- Staff management
- View appointments
- Inventory management

BACK

NEXT

CODE IMPLEMENTATION

We organised our code into the following folders:

```
✓ TrialProgram •  
  > AdminManager •  
  > Appointment  
  > Database  
  > DoctorManager •  
  > Inventory •  
  > LoginPage  
  > Menu  
  > PatientManager  
  > PharmacistMa... •  
  > Roles  
  > StaffManager  
J HospitalApp.java
```

USER LOGIN & CHANGE PWD

```
*** Medical Records ***
1. View Medical Record
2. Update Personal Information

*** Appointment Management ***
3. View Available Appointment Slots
4. Schedule an Appointment
5. Reschedule an Appointment
6. Cancel an Appointment
7. View Scheduled Appointments

*** Past Appointments ***
8. View Past Appointment Outcome Records

*** Security ***
9. Change password

*** Logout ***
10. Logout
=====
8
=====YOUR OUTCOME RECORDS=====
DR_ID      DATE_TIME      PAST_DIAG      SERVICE      MED_NAME      TREATMENT_PLANS
=====

No past appointment records found for Patient ID: PA001
===== Patient Menu =====

*** Medical Records ***
1. View Medical Record
2. Update Personal Information

*** Appointment Management ***
3. View Available Appointment Slots
4. Schedule an Appointment
5. Reschedule an Appointment
6. Cancel an Appointment
7. View Scheduled Appointments

*** Past Appointments ***
8. View Past Appointment Outcome Records

*** Security ***
9. Change password

*** Logout ***
10. Logout
=====
9
Enter the old password: password
This is your first time changing the password.
Enter new password: Password@123
Enter confirmation password: Passwo
```

BACK

NEXT

Code Implementation



PATIENT TEST CASES

BACK

NEXT



PATIENT

```
===== Hospital Management System =====
=====
Please enter your choice to continue.
1. Login
2. Forget UserID
3. Exit
Your choice (1-3): 1
```

BACK

NEXT

Code Implementation



DOCTOR TEST CASES

BACK



NEXT

DOCTOR

```
===== Hospital Management System =====
=====
Please enter your choice to continue.
1. Login
2. Forget UserID
3. Exit
Your choice (1-3): 1
Enter your User ID: D001
Enter your password: password
Login Successful. Remember to change your password
===== Doctor Menu =====

*** Medical Records ***
1. View Patient Medical Records
2. Update Patient Medical Records

*** Appointment Management ***
3. View Personal Schedule
4. Set Availability for Appointments
5. Accept or Decline Appointment Requests
6. View Upcoming Appointments
7. Record Appointment Outcome

*** Security ***
8. Change password

*** Logout ***
9. Logout
```

BACK

NEXT

Code Implementation



PHARMACIST TEST CASES

BACK

NEXT



PHARMACIST

```
===== Hospital Management System =====
=====
Please enter your choice to continue.
1. Login
2. Forget UserID
3. Exit
Your choice (1-3): 1
Enter your User ID: ph@n
Enter your password: pass
```

BACK

NEXT

Code Implementation



ADMIN TEST CASES

BACK



NEXT

ADMIN

```
===== Hospital Management System =====
=====
Please enter your choice to continue.
1. Login
2. Forget UserID
3. Exit
Your choice (1-3): 1
Enter your User ID: a@01
Enter your password: password
Login Successful. Remember to change your password
===== Administrator Menu =====

*** Staff Management ***
1. View and Manage Hospital Staff

*** Appointments ***
2. View Appointments details

*** Medical Inventory ***
3. View and Manage Medication Inventory
4. Approve Replenishment Requests

*** Security ***
5. Change password

*** Logout ***
6. Logout
|
```

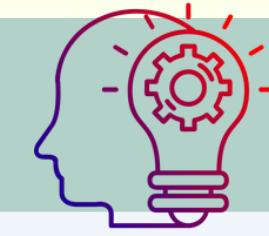
java: Ready Live Share Chat: 4 new

Ln 115, Col 78 Spaces: 4 UTF-8

BACK

NEXT

DESIGN PRINCIPLES USED



Model-View-Controller (MVC)

Model

- + Handles data and rules
- + Manages data for different roles (e.g. Doctor, Patient, etc.)

```
Doctor
<<Property>> -gender : String
<<Property>> -age : String
-upcomingAppointments : ArrayList<Appointment> = new ArrayList<>()
-file : String = "Database/CSV/DB/DR_DB.csv"

+getName() : String
+setName(name : String) : void
+getDoctorID() : String
+setDoctorID(doctorID : String) : void
+setDoctorAppointments(upcomingAppointments : ArrayList<Appointment>) : void
+getDocAppointments() : ArrayList<Appointment>
+Doctor(doctorID : String)
```

View

- + Handles user interface rendering
- + Implemented in DoctorMenu, PatientMenu etc.

```
DoctorMenu
~sc : Scanner = new Scanner(System.in)
-has-a : Doctor

+DoctorMenu(id : String)
+displayMenu() : void
+menuActions(id : String) : void
```

Controller

- + Connects Model & View
- + Handles method implementation (DoctorController, PharmacistController etc.)

```
DoctorController
-appointmentsFilePath : String = "Database/CSV/APPOINTMENTS.csv"
-outcomeFilePath : String = "Database/CSV/OUTCOME.csv"
-doctorAvailFilePath : String = "Database/CSV/DOCTORS_AVAILABILITY.csv"
~scanner : Scanner = new Scanner(System.in)

+viewPersonalSchedule(doctor : Doctor) : void
+viewUpcomingAppointments(doctor : Doctor) : void
+viewPatientMedicalRecords(doctor : Doctor) : void
+updatePatientMedicalRecords(doctor : Doctor) : void
+setAvailability(doctor : Doctor) : void
+displayTimeSlots(date : LocalDate, doctorID : String) : boolean
-getTimeSlot(index : int) : String
+acceptOrDecline(doctor : Doctor) : void
+recordAppointmentOutcome(doctor : Doctor) : void
```

BACK

NEXT

SOLID PRINCIPLES

SINGLE RESPONSIBILITY PRINCIPLE

Every class should have one and only **one** responsibility

⊕ Ensure every class has a **distinct** role

⊕ **DoctorSchedule.java's** role is to manage/ update doctor's schedule

```
DoctorSchedule
-AVAILABILITY_FILE_PATH : String = "Database/CSV/DOCTORS_AVAILABILITY.csv"
~DrAvailFilePath : String = "Database\CSV\DOCTORS_AVAILABILITY.csv"
~sc : Scanner = new Scanner(System.in)
+viewDoctorSchedule(doctorID : String) : void
+displayAvailableTimeSlots(date : LocalDate, doctorID : String) : String
-formatTimeSlot(startHour : int, endHour : int) : String
-formatHour(hour : int) : String
+updateDocSchedule(dID : String, date : String, time : String, avail : String) : void
+checkDocAvail(dID : String, date : String, time : String) : boolean
+docAvailDates(filePath : String, drID : String, targetDate : String) : boolean
+docAvailTimes(filePath : String, drID : String, targetDate : String, targetTime : String) : boolean
+initializeAvailability(doctorID : String) : void
+checkAvailabilityExists(doctorID : String) : boolean
+updateAvailability(date : LocalDate, timeSlot : int, doctorID : String) : void
+getTimeIndex(time : String) : int
+getTime(time : String) : String
+getTimeAlpha(time : String) : String
```

```
MedicalRec Manager
~patientsFilePath : String = "Database/CSV/DB/PATIENT_DB.csv"
+viewMedicalRecords(patientID : String) : void
+viewAllMedicalRecordsForDoc(doctorID : String) : void
```

⊕ **MedicalRecManager.java's** role is to allow Patient/ Doctor to view Medical Records

BACK

NEXT

SOLID PRINCIPLES

OPEN/CLOSED PRINCIPLE

Classes should be open for **extension** but closed to modification

```
package Menu;

import LoginPage.ChangePassword;

public abstract class UserMenu {
    public UserMenu() {
    }

    public void displayMenu() {
        System.out.println("This ID has no assigned role.");
    }

    public void menuActions(String var1) {
        System.out.println("This ID has no assigned role.");
    }

    public void clearConsole() {
        System.out.print("\u001b[H\u001b[2J");
        System.out.flush();
    }

    public void changePassword(String var1) {
        ChangePassword.changePassword(var1);
    }
}
```

 Other classes extend UserMenu (abstract class)

 **Scalability:** New features can be added without disrupting the existing code

 **Stability:** Core code remains unchanged, reducing the risk of bugs

BACK

NEXT

SOLID PRINCIPLES

LISKOV SUBSTITUTION PRINCIPLE

```
public void displayMenu(String id) {  
    UserMenu userMenu;  
    String role = findRoleByID(id);  
    switch(role) {  
        case "PATIENT":  
            userMenu = new PatientMenu();  
            break;  
        case "DOCTOR":  
            userMenu = new DoctorMenu();  
            break;  
        case "PHARMACIST":  
            userMenu = new PharmacistMenu();  
            break;  
        case "ADMINISTRATOR":  
            userMenu = new AdminMenu();  
            break;  
        default:  
            System.out.println("Invalid Role. ");  
            return;  
    }  
    userMenu.menuActions(id);  
}
```

- + Objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
- + The objects of our subclasses (E.g. Patient, Doctor) will behave in the same way as the objects of our superclass (E.g. User)
- + Each role will override the menuActions() method to carry out the actions specific to their roles

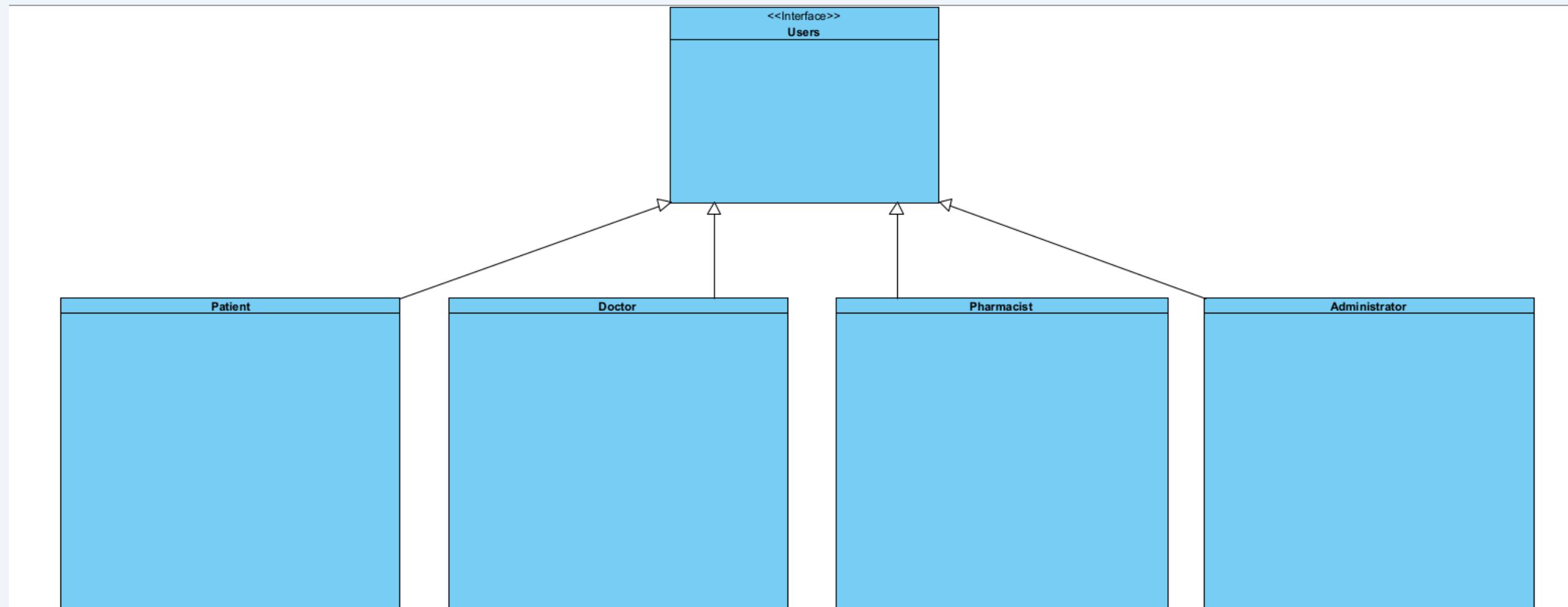
[BACK](#)[NEXT](#)

SOLID PRINCIPLES

INTEGRATION SEGREGATION PRINCIPLE

A class should not be forced to implement methods it doesn't need

- Each role will only implement the methods that it needs and avoid unnecessary dependencies

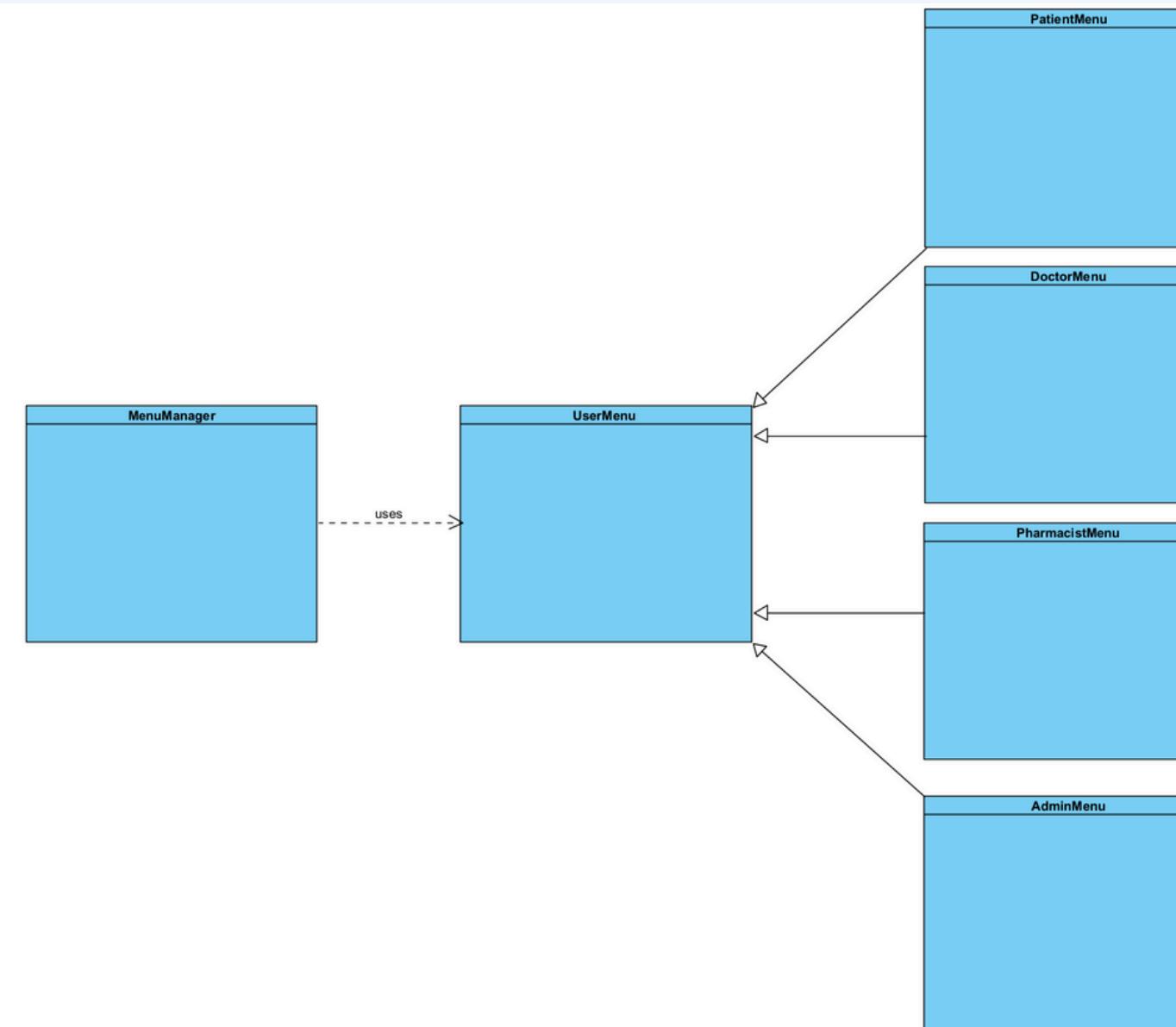


BACK

NEXT

SOLID PRINCIPLES

DEPENDENCY INVERSION PRINCIPLE



- + High-level modules should not depend on low-level modules.
- + Abstractions should not depend on details. Details should depend on abstractions.
- + MenuManager (High-level module) is depending on abstraction class UserMenu
- + Each role has a menu (Low-level module) that implements the UserMenu

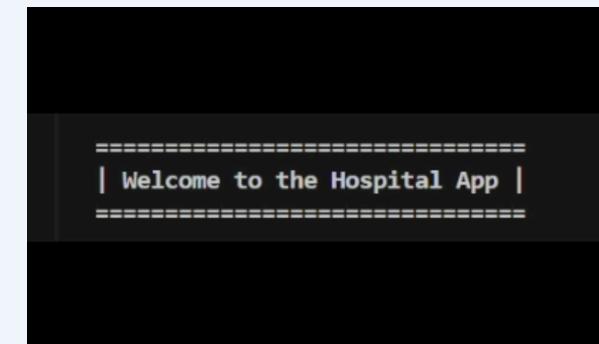
BACK

NEXT

Additional Features



Password
Hashing



Blinking
Interface



Forget UserID



Clear Console

BACK

NEXT



THANK YOU

