JAVA CASE STUDY

BY:

B.satya Rahul(2023000870)

**OBJECTIVE**: HOSPITAL MANAGEMENT SYSTEM

**SCENARIO**:

"The hospital has different types of medical staff (doctors, nurses, administrative staff), each with distinct roles. The system needs to handle various functionalities like patient registration, appointment scheduling, emergency services, and more. Let’s see how specific programming concepts are applied to different components of the system."

**Key Concepts Implemented**:

**1. Method Overloading**

In the HMS, method overloading is used to create multiple ways of adding appointments for patients. This allows the system to handle both regular appointments and emergency consultations using the same method name but with different parameters.

**Example:**

We overload the addAppointment() method to schedule a regular appointment as well as an urgent appointment.

Here, the addAppointment() method is overloaded to handle both regular and emergency appointments. This allows flexibility in scheduling appointments for different types of consultations.

**2. Array Lists**

Array Lists are used in the HMS to store and manage lists of patients, doctors, and appointments. Array Lists provide a dynamic array that can grow as more data is added to the system, such as when new patients are added.

**Example:**

We use an Array List to store a list of patients and manage their information.

**3. Constructor Chaining/Usage of super, this Keywords**

In the HMS, constructor chaining is used when creating objects like Patient or Staff, where the constructor of the subclass calls the constructor of the superclass to initialize common attributes (like name, age, or ID). We also use this to refer to the current class’ attributes.

**4. Packages and Access Specifiers**

In the HMS, packages are used to organize the classes related to different parts of the system. We use access specifiers like public, private, and protected to control the visibility and accessibility of class members (methods and variables).

**Example:**

We organize our classes into a package called hospitalmanagement, and restrict access to certain data using access specifiers.

In this case, the Patient class is in the hospitalmanagement package. We restrict access to the name and age variables using the private access specifier and provide public getter methods to access these values. The Appointment class accesses Patient’s name through the getName() method.

**5. Abstract Classes**

Abstract classes are used in the HMS to define common behaviors for different types of medical staff, such as doctors and nurses, while leaving the specific implementation to subclasses. The abstract class HospitalStaff provides a template for staff roles, and each staff type (doctor, nurse) implements their own version of certain methods.

In this case, HospitalStaff is an abstract class with an abstract method performDuty(). The Doctor and Nurse classes extend HospitalStaff and implement the performDuty() method to define their specific roles.

**Conclusion:**

In this simplified Hospital Management System (HMS), we used essential object-oriented programming concepts to build a flexible and easy-to-maintain system. The use of method overloading, ArrayLists, constructor chaining, access specifiers, and abstract classes allows the system to manage patients, appointments, and hospital staff effectively, while keeping the code organized and adaptable for future changes.

**CODE:**

**import java.util.ArrayList;**

**import java.util.Scanner;**

**// Abstract class representing common behavior for hospital staff**

**abstract class HospitalStaff {**

**String name;**

**int id;**

**// Constructor for initializing common properties**

**public HospitalStaff(String name, int id) {**

**this.name = name;**

**this.id = id;**

**}**

**// Abstract method that will be implemented by subclasses**

**public abstract void performDuty();**

**}**

**// Class representing a Doctor, extending HospitalStaff**

**class Doctor extends HospitalStaff {**

**String specialty;**

**// Constructor chaining: Calling the parent class constructor with super**

**public Doctor(String name, int id, String specialty) {**

**super(name, id); // Calling the parent class constructor**

**this.specialty = specialty;**

**}**

**@Override**

**public void performDuty() {**

**System.out.println(name + " (Doctor) is diagnosing patients in " + specialty + " specialty.");**

**}**

**}**

**// Class representing a Nurse, extending HospitalStaff**

**class Nurse extends HospitalStaff {**

**public Nurse(String name, int id) {**

**super(name, id); // Calling the parent class constructor**

**}**

**@Override**

**public void performDuty() {**

**System.out.println(name + " (Nurse) is assisting in surgeries.");**

**}**

**}**

**// Patient class with access specifiers and method overloading**

**class Patient {**

**private String name;**

**private int age;**

**// Constructor to initialize patient details**

**public Patient(String name, int age) {**

**this.name = name;**

**this.age = age;**

**}**

**// Getter method for name (accessed outside class)**

**public String getName() {**

**return name;**

**}**

**// Getter method for age (accessed outside class)**

**public int getAge() {**

**return age;**

**}**

**// Method overloading: Overloaded method to schedule an appointment**

**public void scheduleAppointment(String date) {**

**System.out.println(name + " has a regular appointment scheduled for " + date);**

**}**

**// Overloaded method to schedule an emergency appointment**

**public void scheduleAppointment(String date, boolean isEmergency) {**

**if (isEmergency) {**

**System.out.println(name + " has an emergency appointment scheduled for " + date);**

**} else {**

**System.out.println(name + " has a regular appointment scheduled for " + date);**

**}**

**}**

**}**

**// Hospital class to manage patients and appointments using ArrayList**

**class Hospital {**

**private ArrayList<Patient> patients;**

**public Hospital() {**

**patients = new ArrayList<>(); // Initialize the ArrayList**

**}**

**// Method to add a patient to the hospital**

**public void addPatient(Patient patient) {**

**patients.add(patient);**

**System.out.println(patient.getName() + " has been added to the hospital system.");**

**}**

**// Method to display all patients in the system**

**public void displayPatients() {**

**System.out.println("Patients in the system:");**

**for (Patient patient : patients) {**

**System.out.println("Name: " + patient.getName() + ", Age: " + patient.getAge());**

**}**

**}**

**}**

**public class hospitalmanagement{**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**// Input Doctor and Nurse details**

**System.out.println("Enter Doctor Name: ");**

**String doctorName = scanner.nextLine();**

**System.out.println("Enter Doctor ID: ");**

**int doctorId = scanner.nextInt();**

**scanner.nextLine(); // Consume the newline**

**System.out.println("Enter Doctor Specialty: ");**

**String specialty = scanner.nextLine();**

**Doctor doctor = new Doctor(doctorName, doctorId, specialty);**

**System.out.println("Enter Nurse Name: ");**

**String nurseName = scanner.nextLine();**

**System.out.println("Enter Nurse ID: ");**

**int nurseId = scanner.nextInt();**

**scanner.nextLine(); // Consume the newline**

**Nurse nurse = new Nurse(nurseName, nurseId);**

**// Perform duties for Doctor and Nurse**

**doctor.performDuty();**

**nurse.performDuty();**

**// Input Patient details**

**Hospital hospital = new Hospital();**

**System.out.println("Enter the number of patients: ");**

**int numOfPatients = scanner.nextInt();**

**scanner.nextLine(); // Consume the newline**

**for (int i = 0; i < numOfPatients; i++) {**

**System.out.println("Enter Patient Name: ");**

**String patientName = scanner.nextLine();**

**System.out.println("Enter Patient Age: ");**

**int patientAge = scanner.nextInt();**

**scanner.nextLine(); // Consume the newline**

**Patient patient = new Patient(patientName, patientAge);**

**hospital.addPatient(patient);**

**// Scheduling appointments for each patient**

**System.out.println("Enter appointment date for " + patientName + ": ");**

**String appointmentDate = scanner.nextLine();**

**System.out.println("Is this an emergency appointment? (true/false): ");**

**boolean isEmergency = scanner.nextBoolean();**

**scanner.nextLine(); // Consume the newline**

**patient.scheduleAppointment(appointmentDate, isEmergency);**

**}**

**// Displaying all patients in the hospital**

**hospital.displayPatients();**

**}**

**}**