ASSIGNMENT – 1

**Objective:** To design a basic system in a domain of your choosing (e.g. railway, school, sports, etc.) using Java that incorporates concepts from Units 1,2 & 3.

**Domain: Hospital Management system**

The hospital management system is a common and practical domain for showcasing programming skills and concepts. It involves the management of patients, doctors, and appointments. It allows for the application of object-oriented programming principles to model real-world entities and their interactions.

In a real hospital management system, patients register, book appointments with doctors, view their scheduled appointments, and access doctor information. Doctors register, view appointments, and can make changes as necessary. This system aims to provide an efficient way to manage hospital operations and improve patient-doctor interactions. The project is a basic structure for creating a hospital management system, and it will require many modifications and advancements to handle real-life problems.

**Documentation:**

**Class Structure:**

**Person Class:**

This class serves as the base class for both doctors and patients.

It contains common attributes like name, gender, and phone number, and provides methods for creating and displaying person details.

**Patient Class:**

Inherits from the Person class.

Adds attributes like ins (insurance), Pid (Patient ID), and maintains an array of patient objects.

Provides a method for creating patient details.

**Doctor Class:**

Also inherits from the Person class.

Adds attributes like splz (specialization), fee (consultation fees), and maintains an array of doctor objects.

Provides a method for creating doctor details.

**Appointment Class:**

Represents an appointment with a doctor.

Contains attributes for doctor, patient, date, time, and an array to maintain appointments.

Provides methods for displaying appointments, showing doctor-specific appointments, showing patient-specific appointments, cancelling appointments, and rescheduling appointments.

**Hospital Class:**

Contains methods for the menu-driven operations of the system.

Handles the flow between doctor and patient portals.

Offers menus for registration, appointment booking, listing appointments, and more.

**Code Flow:**

The program begins by asking the user to select their designation (Patient or Doctor).

Depending on the designation selected, the program enters either the patient or doctor portal:

**Patient Portal:**

Allows patients to register, book appointments, check their appointments, view doctor details, and see available doctors.

**Doctor Portal:**

Allows doctors to register, view scheduled appointments for the day, get a list of doctors, cancel appointments, and reschedule appointments.

In both portals, users can navigate the menu options using a menu-driven approach. The program guides users through registration, appointment booking, appointment viewing, and more.

The Appointment class handles the booking, viewing, and management of appointments between patients and doctors.

Data for patients and doctors are stored in arrays within their respective classes.

**Flow of the Code:**

1. The program starts by asking the user for their designation (patient or doctor).

2. The program enters the respective portal (patient or doctor).

3. In each portal, users can choose from various menu-driven options to perform actions such as registration, appointment booking, appointment viewing, or accessing information.

4. The program uses a set of classes to model patients, doctors, and appointments, allowing for data storage and management.

5. Both the portals have a bridge in between so that they can switch in between and work accordingly.

6. Error handling is incorporated with try-catch blocks to handle exceptions gracefully.

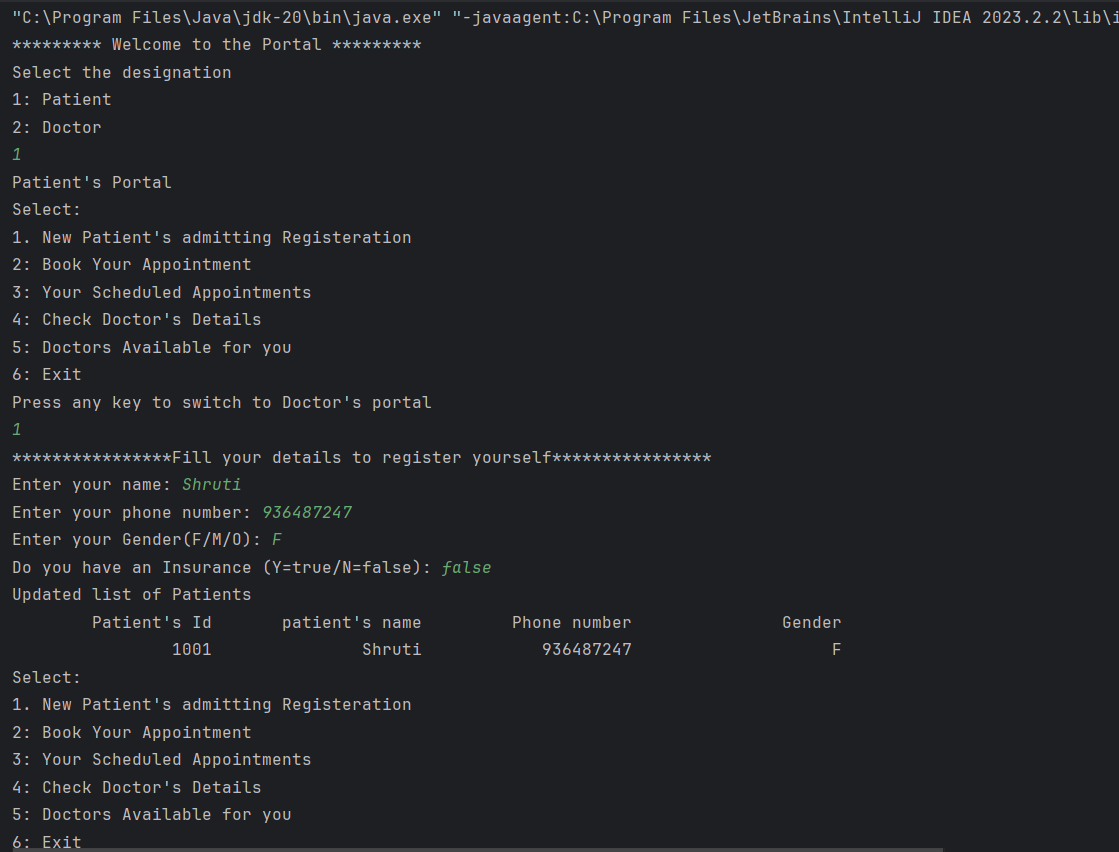
**Source Code**

import java.util.Formatter;  
import javax.print.Doc;  
import java.nio.ByteBuffer;  
import java.nio.CharBuffer;  
import java.util.Scanner;  
class Person{// Base class person that includes attributes and methods common for doctor and patient  
 Scanner sc= new Scanner(System.*in*);  
 String name, gender;  
 long phn;  
 void createPerson(){//Create person object  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Fill your details to register yourself\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.print("Enter your name: ");  
 this.name = sc.nextLine();  
  
 System.*out*.print("Enter your phone number: ");  
 this.phn = sc.nextLong();  
  
 System.*out*.print("Enter your Gender(F/M/O): ");  
 this.gender = sc.next();  
  
 }  
 void getDetail() {//function to show all details of person  
  
 System.*out*.println("Name = " + name);  
 System.*out*.println("Gender = " + gender);  
 System.*out*.println("Phone number = " + phn);  
 }  
}  
class Patient extends Person{//patient class to inherit person and adding some more attributes  
 static Patient[] *patients* = new Patient[20];  
  
 Scanner sc= new Scanner(System.*in*);  
 int Pid=1000;  
 Boolean ins;  
 static int *count* = 0;  
 void createPatient(){//Create patient object  
 super.createPerson();  
 System.*out*.print("Do you have an Insurance (Y=true/N=false): ");  
 this.ins = sc.nextBoolean();  
 *patients*[*count*] = this;  
 *count*++;  
 this.Pid+=*count*;  
 }  
 void getDetail(){  
 System.*out*.println("Patient's ID: "+Pid);  
 super.getDetail();  
 System.*out*.println("Have Insurance = "+ins);  
 }  
}  
  
class Doctor extends Person{//Inheriting person in doctor class  
 static Doctor[] *doctors* = new Doctor[20];  
 Scanner sc= new Scanner(System.*in*);  
 int Did=2000;  
 String splz;  
 int fee;  
 static int *count* = 0;  
  
 void createDoctor(){// creating doctor object  
 super.createPerson();  
 System.*out*.print("Enter your Speciallization: ");  
 this.splz = sc.nextLine();  
  
 System.*out*.print("Enter your Consultation fees: ");  
 this.fee = sc.nextInt();  
  
 *doctors*[*count*] = this;  
 *count*++;  
 this.Did+=*count*;  
 }  
 void getDetail(){  
 System.*out*.println("Doctor's ID: "+Did);  
 super.getDetail();  
 System.*out*.println("Spciallized in"+splz);  
 System.*out*.println("Consultation fee: "+fee);  
 }  
}  
class Appointment {// class for booking appointments  
 int appId = 3001;  
 private Doctor doctor;  
 private Patient patient;  
 private String date;  
 private String time;  
 static Appointment[] *appointments* = new Appointment[30];  
  
 static int *countAppointments* = 0;  
 public Appointment(Doctor doctor, Patient patient, String date, String time){ //constructor booking appointments  
 this.doctor = doctor;  
 this.patient = patient;  
 this.date = date;  
 this.time = time;  
 *appointments*[*countAppointments*] = this;  
 appId+=*countAppointments*;  
 *countAppointments*++;  
  
 }  
  
 public static void showAppointments(){//function to show appointments booked till now  
 //this will show all the appointments of all the doctors  
 int size = *appointments*.length;  
 System.*out*.printf("%20s %20s %20s %20s %20s \n","Appointment ID","Doctor's name","patient's name", "Date" ,"Time");  
 for(int i=0;i<*countAppointments*;i++){  
 System.*out*.printf("%20s %20s %20s %20s %20s\n \n",*appointments*[i].appId,*appointments*[i].doctor.name,*appointments*[i].patient.name,*appointments*[i].date,*appointments*[i].time);  
 }  
 }  
 public static void showDoctorApp(String docName) {//show appointments of specified doctor  
 System.*out*.printf("%20s %20s %20s %20s","Doctor's name","patient's name", "Date" ,"Time");  
 for(int i=0;i<*countAppointments*;i++){  
 if(*appointments*[i].doctor.name.equalsIgnoreCase(docName))  
 System.*out*.printf("%20s %20s %20s %20s\n \n",*appointments*[i].doctor.name,*appointments*[i].patient.name,*appointments*[i].date,*appointments*[i].time);  
 }  
 }  
  
 public static void showPatientApp(String patName){// show appointments of specified patient  
 boolean found = false;  
 int ind=-1;  
 for(int i=0;i<*countAppointments*;i++){  
 if(*appointments*[i].patient.name.equalsIgnoreCase(patName)){  
 ind = i;  
 found=true;  
 }  
 }  
 if(found){  
 System.*out*.printf("%20s %20s %20s %20s \n","Doctor's name","patient's name", "Date" ,"Time");  
 System.*out*.printf("%20s %20s %20s %20s\n \n",*appointments*[ind].doctor.name,*appointments*[ind].patient.name,*appointments*[ind].date,*appointments*[ind].time);  
 }  
 else{  
 System.*out*.println("No such Patient in records");  
 }  
 }  
  
 public static void DenyApp(int AppId){//To cancel an appointment  
  
 int ind=0;  
 boolean found = false;  
 for(int i=0;i<*countAppointments*;i++){  
 if(*appointments*[i].appId == AppId){  
 ind =i;  
 found = true;  
 }  
 }  
 *countAppointments*--;  
 if(found) {  
 for (int i = ind; i < *countAppointments* - 1; i++) {  
 *appointments*[i] = *appointments*[i + 1];  
 }  
 *appointments*[*countAppointments*] = null;  
 System.*out*.println("Appointement Cancelled!");  
 }  
 }  
 public static void rescheApp(int id,String t, String d){//To Reschedule any appointment  
 int ind=0;  
 boolean found = false;  
 for(int i=0;i<*countAppointments*;i++){  
 if(*appointments*[i].appId==id){  
 ind = i;  
 found = true;  
 *appointments*[i].time = t;  
 *appointments*[i].date = d;  
 break;  
 }  
 }  
 if(found) {  
 System.*out*.println("Re-Schedule done!");  
 System.*out*.printf("%20s %20s %20s %20s \n", "Doctor's name", "patient's name", "Date", "Time");  
 System.*out*.printf("%20s %20s %20s %20s \n\n", *appointments*[ind].doctor.name, *appointments*[ind].patient.name, *appointments*[ind].date, *appointments*[ind].time);  
 }  
 else{  
 System.*out*.println("No records to ReSchedule for the given input");  
 }  
 }  
}  
  
public class Hospital {  
 static void showDocMenu(){//menu driven options for Doctor's portal  
 System.*out*.println("1. New Doctor joining Registeration");  
 System.*out*.println("2: Scheduled Appointments for the day");  
 System.*out*.println("3: Get Doctor's List");  
 System.*out*.println("4: Deny an Appointment");  
 System.*out*.println("5: Reschedule Appointment");  
 System.*out*.println("6: Exit");  
 System.*out*.println("Press any key to switch to Patient's Portal");  
 }  
  
 static void showPatMenu(){//menu driven options for Patient's portal  
 System.*out*.println("1. New Patient's admitting Registeration");  
 System.*out*.println("2: Book Your Appointment");  
 System.*out*.println("3: Your Scheduled Appointments");  
 System.*out*.println("4: Check Doctor's Details");  
 System.*out*.println("5: Doctors Available for you");  
 System.*out*.println("6: Exit");  
 System.*out*.println("Press any key to switch to Doctor's portal");  
 }  
 static void docOperations() {//function contains The Menu Driven operations for doctor's portal  
 Scanner sc = new Scanner(System.*in*);  
  
 while (true) {  
 System.*out*.println("Select: ");  
 *showDocMenu*();  
 int choice = sc.nextInt();  
 switch (choice) {  
 case (1)://New Doctor joining Registeration  
 Doctor doc = new Doctor();  
 doc.createDoctor();  
 Doctor[] arr = Doctor.*doctors*;  
 System.*out*.printf("%20s %20s %20s %20s %20s \n", "Doctor's name", "Phone number", "Gender", "Speciallization", "Consultation fees");  
 for (int i = 0; i < Doctor.*count*; i++)  
 System.*out*.printf("%20s %20s %20s %20s %20s\n", arr[i].name, arr[i].phn, arr[i].gender, arr[i].splz, arr[i].fee);  
 break;  
 case (2)://Scheduled Appointments for the day  
 Appointment.*showAppointments*();  
 break;  
 case (3)://Get Doctor's List  
 Doctor[] arr1 = Doctor.*doctors*;  
 if(Doctor.*count*==0){  
 System.*out*.println("no records");  
 break;  
 }  
 System.*out*.printf("%20s %20s %20s %20s %20s \n", "Doctor's name", "Phone number", "Gender", "Speciallization", "Consultation fees");  
 for (int i = 0; i < Doctor.*count*; i++)  
 System.*out*.printf("%20s %20s %20s %20s %20s \n", arr1[i].name, arr1[i].phn, arr1[i].gender, arr1[i].splz, arr1[i].fee);  
 break;  
 case (4)://Deny an Appointment  
 System.*out*.println("These are the scheduled appointments");  
 Appointment.*showAppointments*();  
 System.*out*.print("Enter the Appointment ID: ");  
 int id = sc.nextInt();  
 Appointment.*DenyApp*(id);  
 break;  
  
 case (5)://Reschedule Appointment  
 System.*out*.println("available appointments");  
 Appointment.*showAppointments*();  
 System.*out*.println("Enter App id: ");  
 id = sc.nextInt();  
 System.*out*.println("Enter new date: ");  
 String date = sc.next();  
 System.*out*.println("Enter new time");  
 String time = sc.next();  
 Appointment.*rescheApp*(id,time,date);  
 break;  
 case(6)://Exit  
 System.*exit*(0);  
 default://switch to Patient's Portal  
 *patOperations*();  
 }  
 }  
 }  
 static void patOperations(){//function contains The menu driven operations for patient's portal  
 Scanner sc = new Scanner(System.*in*);  
 while (true) {  
 System.*out*.println("Select: ");  
 *showPatMenu*();  
 int choice = sc.nextInt();  
 switch (choice) {  
 case (1)://New Patient's admitting Registeration  
 Patient pat = new Patient();  
 pat.createPatient();  
 System.*out*.println("Updated list of Patients");  
 Patient[] ar = Patient.*patients*;  
 System.*out*.printf("%20s %20s %20s %20s \n","Patient's Id","patient's name", "Phone number" ,"Gender", "Insurance");  
 for (int i = 0; i < Patient.*count*; i++)  
 System.*out*.printf("%20s %20s %20s %20s \n", ar[i].Pid, ar[i].name, ar[i].phn, ar[i].gender, ar[i].ins);  
 break;  
 case (2)://Book Your Appointment  
 Doctor doc = new Doctor();  
 doc.createDoctor();  
 Patient pat1 = new Patient();  
 pat1.createPatient();  
 String date, time;  
 System.*out*.print("Enter date: ");  
 date = sc.next();  
 System.*out*.print("Enter time: ");  
 time = sc.next();  
 Appointment ap = new Appointment(doc,pat1,date,time);  
 Appointment.*showAppointments*();  
 break;  
 case (3)://Your Scheduled Appointments  
 String name;  
 System.*out*.print("enter patient's name: ");  
 name = sc.next();  
 Appointment.*showPatientApp*(name);  
 break;  
 case (4)://Check Doctor's Details  
 System.*out*.println("Showing doctors list");  
 Doctor[] arr = Doctor.*doctors*;  
 System.*out*.printf("%20s %20s \n", "Doctor's id", "Doctor's name");  
 for (int i = 0; i < Doctor.*count*; i++)  
 System.*out*.printf("%20s %20s \n", arr[i].Did, arr[i].name);  
 System.*out*.println("which doctor's details you want enter id: ");  
 int id = sc.nextInt();  
 boolean found = false;  
 for (int i = 0; i < Doctor.*count*; i++){  
 if(arr[i].Did == id){  
 found = true;  
 arr[i].getDetail();  
 }  
 }  
 if(!found){  
 System.*out*.println("No records");  
 }  
 break;  
 case (5)://Doctors Available for you  
 if(Doctor.*count*==0){  
 System.*out*.println("No doctors available right now");  
 break;  
 }  
 Doctor[] arr1 = Doctor.*doctors*;  
 System.*out*.printf("%20s %20s %20s %20s \n", "Doctor's name", "Phone number", "Gender", "Speciallization", "Consultation fees");  
 for (int i = 0; i < Doctor.*count*; i++)  
 System.*out*.printf("%20s %20s %20s %20s \n", arr1[i].name, arr1[i].phn, arr1[i].gender, arr1[i].splz, arr1[i].fee);  
 break;  
 case(6)://Exit  
 System.*exit*(0);  
 default://switch to Doctor's portal  
 *docOperations*();  
 }  
 }  
 }  
 public static void main(String[] args) {  
 try{  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("\*\*\*\*\*\*\*\*\* Welcome to the Portal \*\*\*\*\*\*\*\*\*");  
 System.*out*.println("Select the designation");  
 System.*out*.println("1: Patient \n2: Doctor");  
 int person = sc.nextInt();  
 sc.nextLine();  
 switch (person) {  
 case (1):  
 System.*out*.println("Patient's Portal");  
 *patOperations*();  
 break;  
 case (2):  
 System.*out*.println("Doctor's Portal");  
 *docOperations*();  
 break;  
 }  
 }catch(Exception e){  
 System.*out*.println("Some Error Occured exiting...");  
 System.*out*.println(e);  
 }  
 }  
}

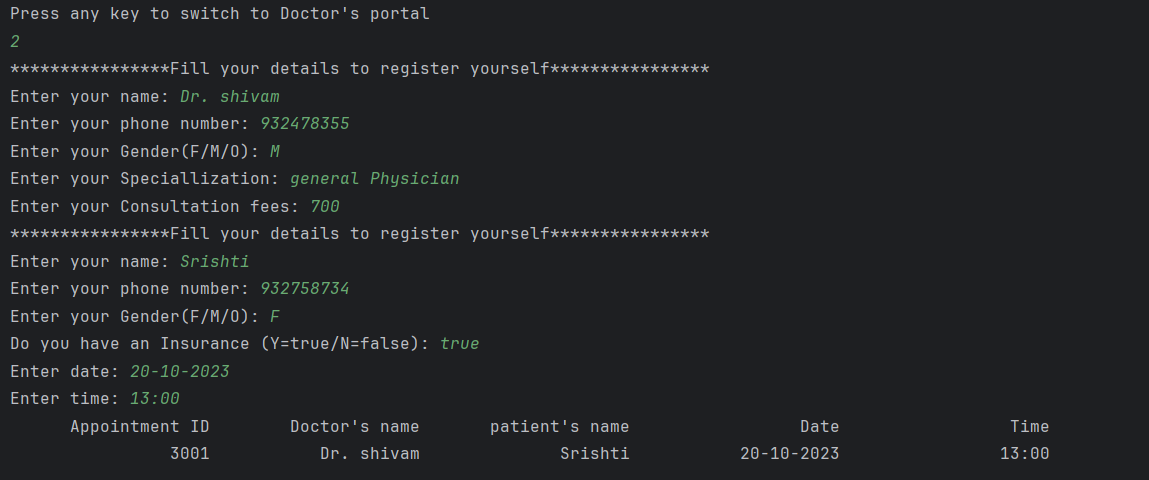
**Outputs**

**Patient’s portal**

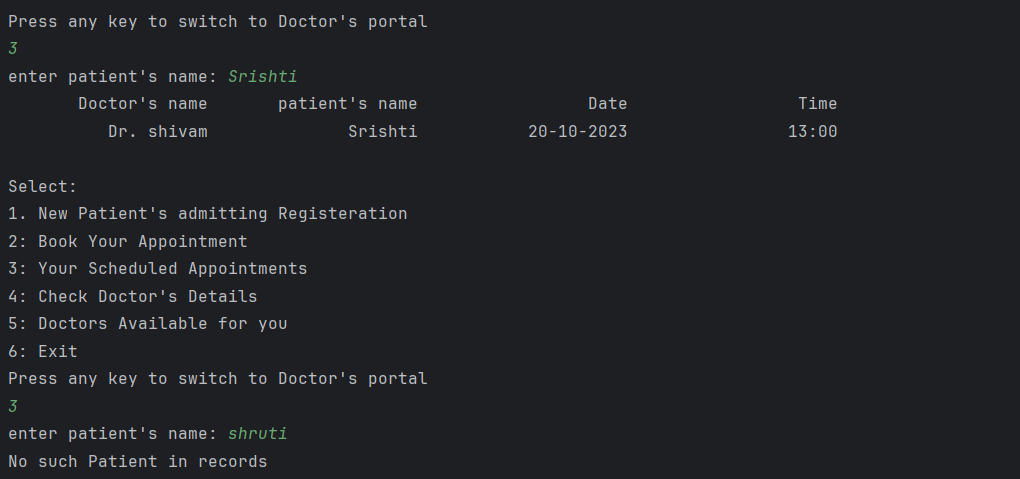
**1. New Entry**

****

**2. Booking Appointment**

****

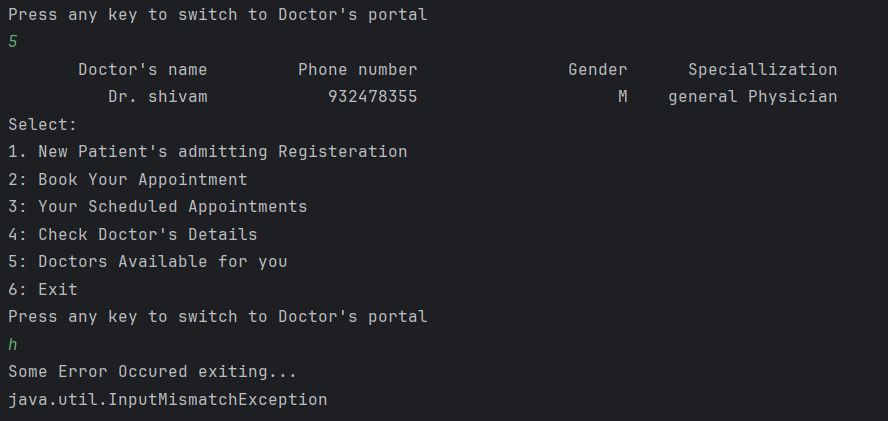
**3. Show scheduled Appointments**

****

**4. Doctor’s List and their details**

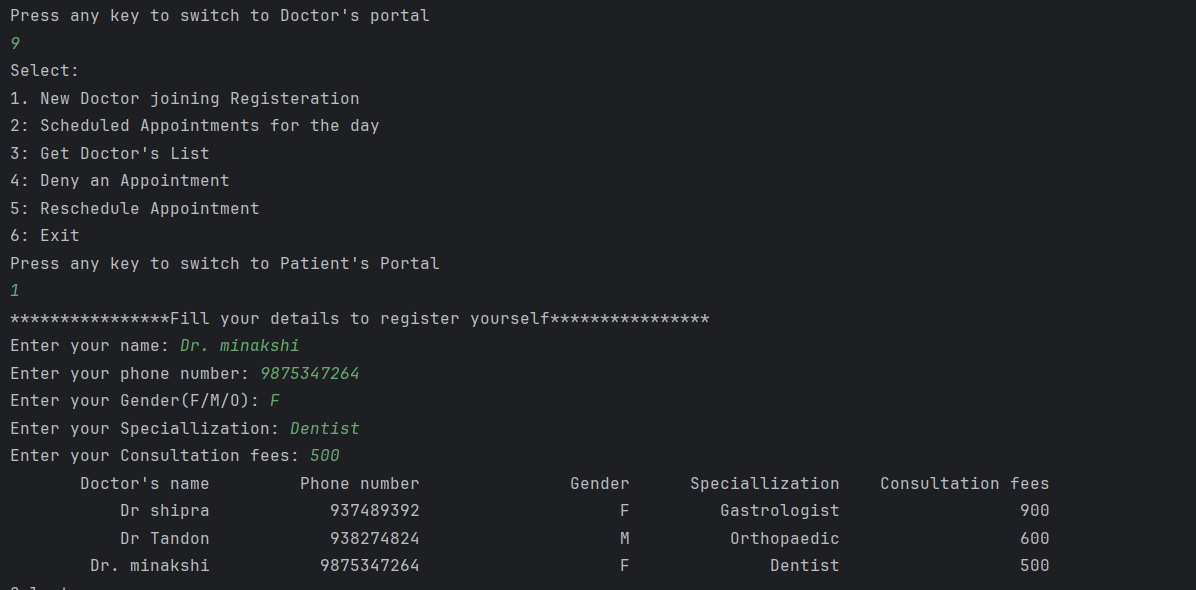
****

**5. Doctor’s List and exception handling for incorrect input**

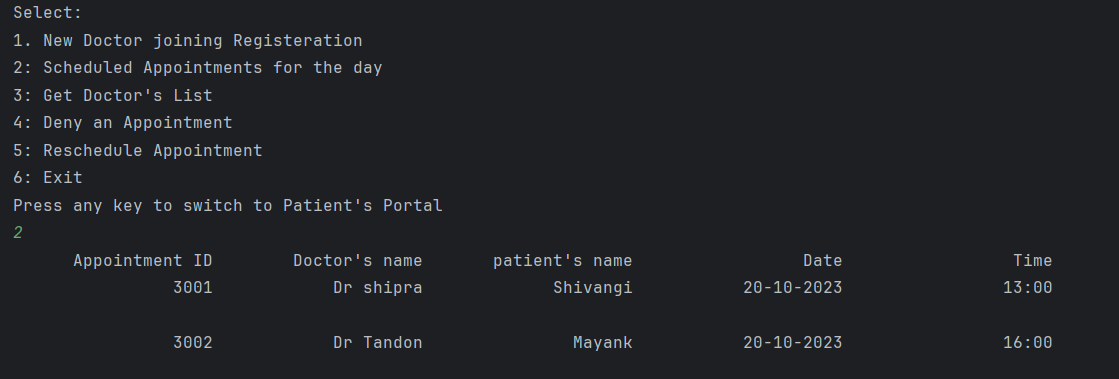
****

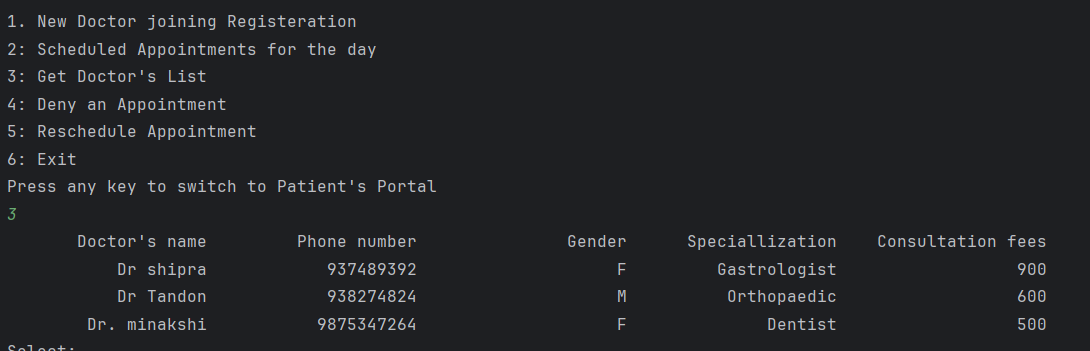
**Doctor’s Portal**

**1. Create New Entry for new joined Doctor (switched to doctor’s portal)**

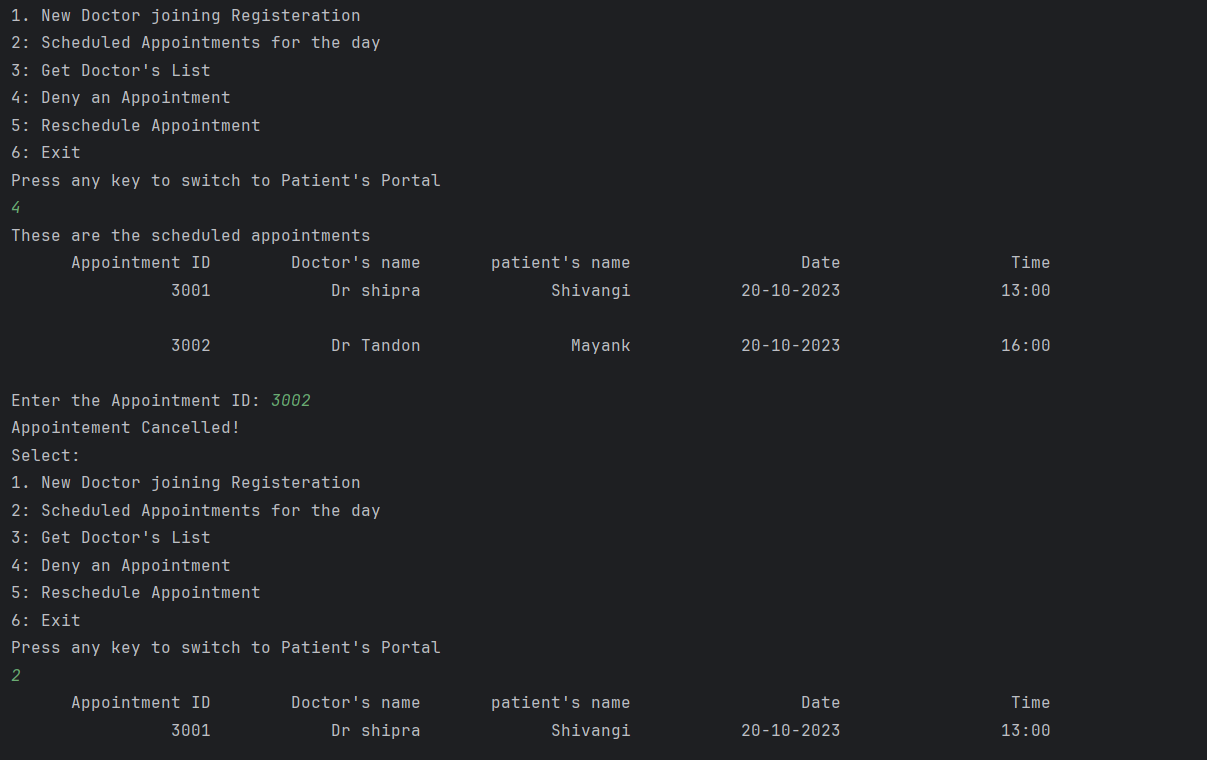
****

**2. Scheduling and Showing Appointments**

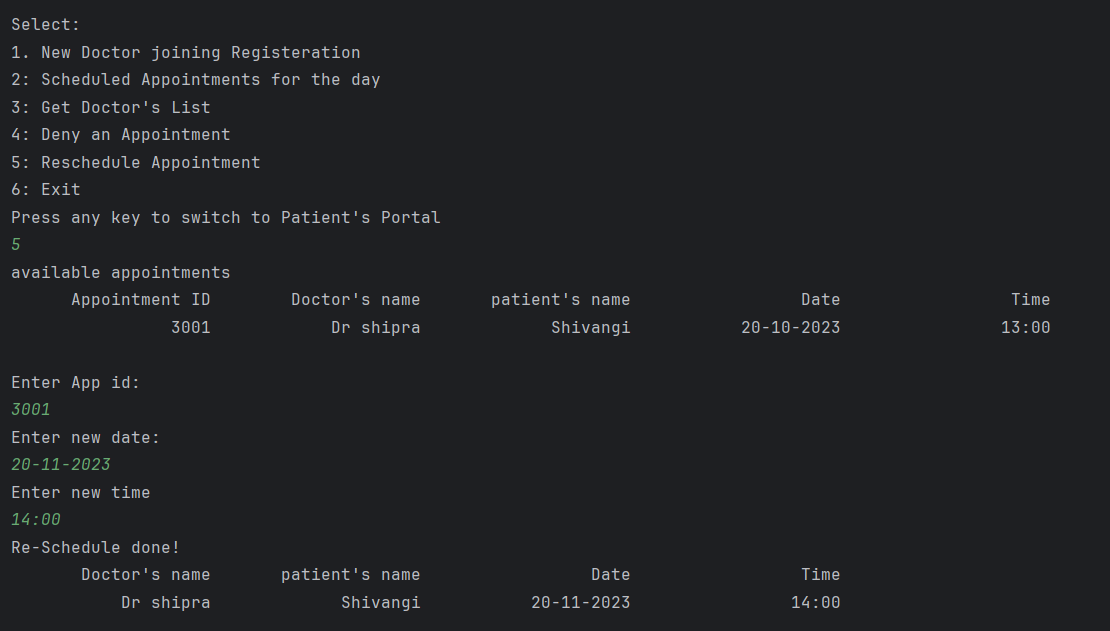
****

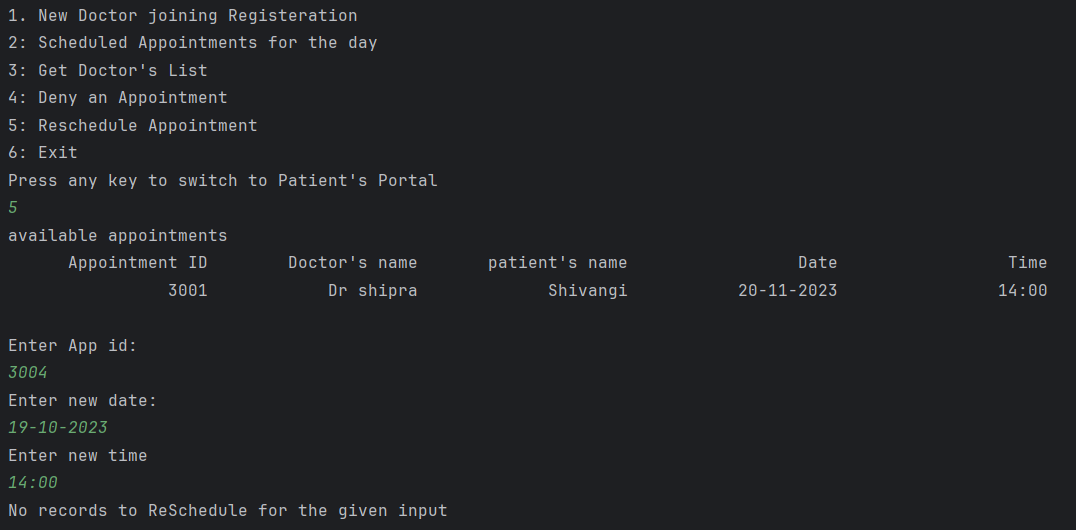
**3. Doctor’s list**

**4. Denying any Appointment**

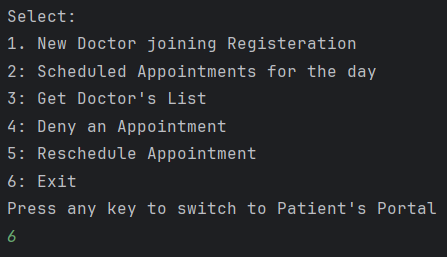
****

**5. Reschedule The Appointment (Modification in date & time)**

****

****

**6. Exiting from the system**

****

**Explanation of concepts Implemented in code**

**1. Import Statements (Lines 1-3):**

- `import java.util.Formatter;`: Importing the `Formatter` class from the `java.util` package.

- `import javax.print.Doc;`: Importing the `Doc` class from the `javax.print` package.

- `import java.nio.ByteBuffer;` and `import java.nio.CharBuffer;`: Importing classes related to byte and character buffers from the `java.nio` package.

**2. Class Definitions (Lines 4-146):**

- The code defines several classes: `Person`, `Patient`, `Doctor`, and `Appointment`. These classes are used to model a hospital management system.

**3. Inheritance(Lines 16 and 41):**

- The `Patient` and `Doctor` classes both extend the `Person` class. This represents an inheritance relationship, where `Patient` and `Doctor` inherit attributes and methods from the `Person` class.

**4. Instance Variables and Methods (Lines 18-30 and Lines 43-84):**

- The `Person` class contains instance variables like `name`, `gender`, `phn`, and methods like `createPerson()` and `getDetail()` for creating and displaying person details.

**5. Static Arrays (Lines 20-21 and Lines 45-46):**

- The `patients` and `doctors` arrays are used to store instances of patients and doctors, respectively. These are declared as static arrays with a maximum size of 20.

**6. Constructors (Lines 58-77):**

- The `Doctor` and `Patient` classes have constructors `createDoctor()` and `createPatient()` that initialize their attributes by calling the `createPerson()` method from the base `Person` class.

**7. Appointment Class (Lines 86-141):**

- The `Appointment` class is used for booking appointments. It contains instance variables like `doctor`, `patient`, `date`, and `time`.

- The constructor `public Appointment(Doctor doctor, Patient patient, String date, String time)` is used to create appointments.

- Several methods like `showAppointments()`, `showDoctorApp()`, `showPatientApp()`, `DenyApp()`, and `rescheApp()` are used to manage appointments.

**8. Menu-Driven Operations (Lines 143-221 and Lines 223-316):**

- The `showDocMenu()` and `showPatMenu()` methods display menu options for the doctor and patient portals, respectively.

- The `docOperations()` and `patOperations()` methods implement the menu-driven operations for doctors and patients.

- These operations include registering new doctors/patients, booking appointments, displaying appointments, viewing doctor details, and more.

**9. Main Method (Lines 318-376):**

- The `main` method is the entry point of the program.

- It starts by asking the user to select their designation (patient or doctor) and then calls the corresponding portal functions (e.g., `patOperations()` for patients and `docOperations()` for doctors).

- Exception handling is included to handle any errors that may occur during program execution.

**10. Flow Control (Throughout the code):**

The code uses various control structures like loops (e.g., `while` loops for menu-driven options), conditionals (e.g., `switch` statements for menu choices), and conditional statements to control the flow of the program.

**11. Input/Output (Throughout the code):**

- The `Scanner` class is used to take user input for various attributes.

- `System.out.println()` is used to display messages and results to the user.

**12. Data Storage (Lines 20-21 and Lines 45-46, Lines 86-141):**

- The program uses arrays (`patients`, `doctors`, and `appointments`) to store and manage data about patients, doctors, and appointments.

**13. Constructor Overriding(Lines 58-77):**

- In the `Doctor` and `Patient` classes, the constructors `createDoctor()` and `createPatient()` are used to create instances of doctors and patients, respectively.

- These constructors override the default constructor of the base class, `Person`. By calling `super.createPerson()`, they invoke the constructor of the base class and initialize the common attributes (`name`, `gender`, and `phn`).

- This is an example of constructor overriding, where a subclass provides its own implementation of a constructor, but still invokes the constructor of the base class.

**14. Method Overriding (Lines 32-40, Lines 87-116, and Lines 143-316):**

- In the `Person`, `Patient`, and `Doctor` classes, the `getDetail()` method is defined.

- The `getDetail()` method in the `Person` class is overridden in both the `Patient` and `Doctor` classes to provide customized implementations.

- In the `Appointment` class, several methods like `showAppointments()`, `showDoctorApp()`, `showPatientApp()`, `DenyApp()`, and `rescheApp()` are defined, and these methods are called with the corresponding objects.

- This is an example of method overriding, where a subclass provides its own implementation of a method that is already defined in the base class. It allows each subclass to have its version of the method.

Certainly, here are the key points to highlight what you've learned from the provided Java code:

**CONCLUSION AND RESULTS**

Learned Concepts:

1. **Basic Concepts and Syntax of Java**: - I have gained an understanding of the fundamental syntax and structure of the Java programming language.

2. **Inheritance**:- Inheritance concepts have been learned and applied, including constructor overriding and method overriding, showcasing how classes can inherit and extend functionality from other classes.

3. **Exception Handling**:- You have encountered exception handling in Java, which is crucial for managing and responding to errors during program execution.

4. **String Handling:** - The code involves string handling, such as capturing and displaying user input using the `Scanner` class, and manipulating strings to display information to users.

5. **Arrays**: - You have worked with arrays, specifically static arrays, to store and manage data about patients, doctors, and appointments.

These concepts provide a solid foundation for understanding object-oriented programming in Java and can be extended to develop more complex and feature-rich applications. To continue improving MY Java skills, consider exploring more advanced topics like data structures, design patterns, and database integration.

**Scope for Improvement:**

1. **Code Organization**: The code could benefit from better organization, with the separation of classes into different files or packages for improved maintainability.

2. **Exception Handling**: While the code includes a basic exception handling mechanism, it could be enhanced to handle specific exceptions more gracefully. For example, handling input validation errors or database-related exceptions.

3. **User Experience**: The program's user interface is text-based and may not provide the best user experience. Consider using a graphical user interface (GUI) or web-based interface to make it more user-friendly.

4. **Data Persistence**: The code currently uses static arrays to store data. In a real-world application, a database or some form of data persistence mechanism would be more suitable.

5. **Security**: The code does not address security concerns, such as user authentication or data encryption. In a production environment, security measures would be crucial.

6. **Error Handling**: The code could be improved by providing more informative error messages to users in case of unexpected inputs or errors.