



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SCHOOL OF COMPUTING
Faculty of Engineering

SEMESTER II 2022/2023

SUBJECT :

SECJ2154 OBJECT-ORIENTED PROGRAMMING

SECTION :

10

LECTURER :

Dr. Nurfazrina binti Mohd Zamry

PROJECT TITLE :

Hospital Registration System

GROUP MEMBERS :

1. Lew Chin Hong A21EC0044
2. Yeo Chun Teck A21EC0148
3. Huam Jun Xiang A21EC0031

Introduction

Java, being one of the most widely used programming languages, offers a robust and versatile platform for developing a wide range of applications. This mini-project report presents an overview of a Java-based software project aimed at presenting multiple basic concepts of Java language—for example, Classes, Inheritance, Polymorphism, Association, Attributes and Methods.

Throughout this report, we will provide detailed insights into our project's approach, design decisions, and implementation techniques, offering a comprehensive view of our development process. We believe that this project report will serve as a valuable resource for understanding the project's objectives, methodology, and outcomes, while also showcasing the versatility and effectiveness of Java as a programming language for software development.

UML Description

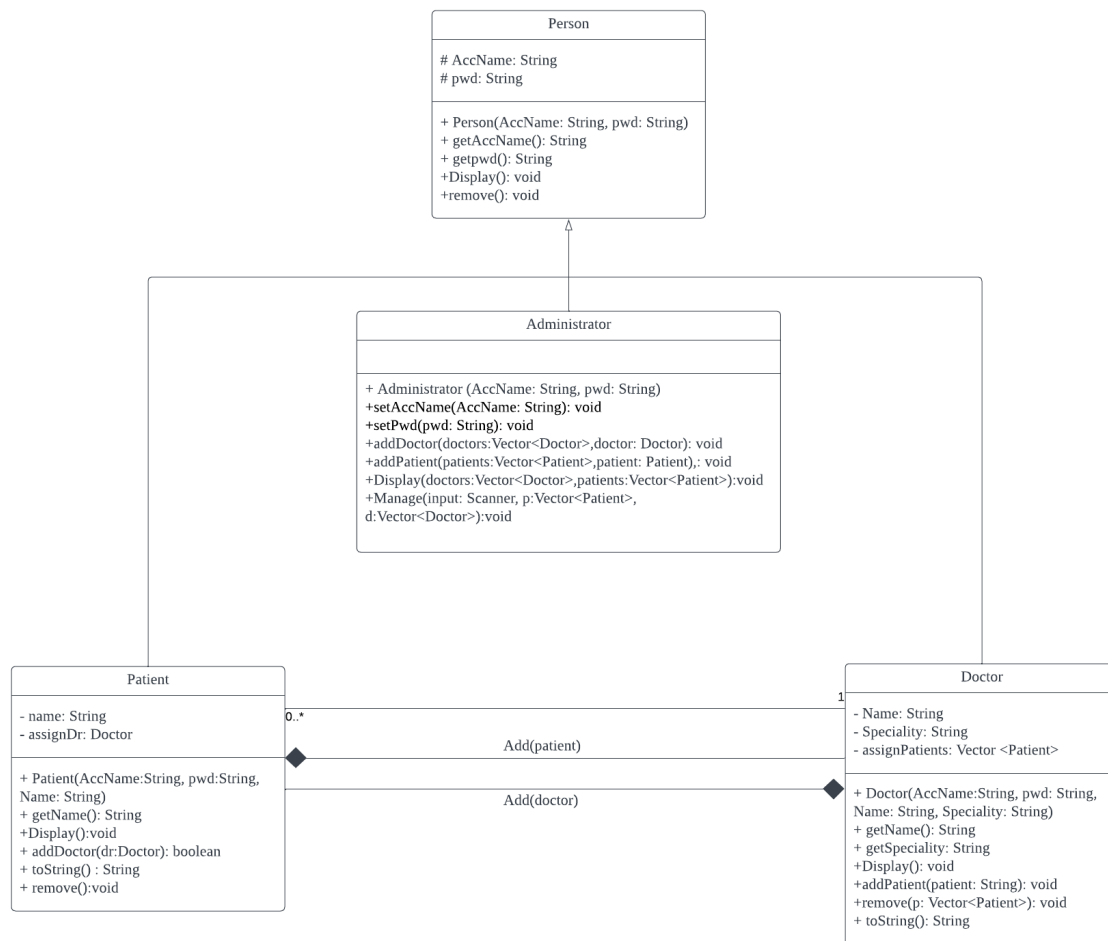


Figure 1: UML Diagram of Mini Project

The figure above is the UML diagram of this mini project. There are four classes included in this mini project which are Person, Administrator, Patient and Doctor. The Person class is a parent class to be inherited by the other three classes. The administrator is a class that can perform various functions. For example, add new patients and doctors and assign an existing doctor to existing patients in the system. The patient class is able to add and remove his doctor(only one doctor for each patient). Finally, the doctor class also do a similar function to the patient class: the doctor is able to add and remove his patients. (one doctor can have multiple patients)

Implementation

Technologies Used

- Java: Java SE 16
- IDE: Visual Studio Code
- Additional Libraries or Frameworks: None

Development Environment

For the development of this project, some technologies are used in the development environment. In this project, we used Visual Studio Code as our IDE. Visual Studio Code is a popular and widely used source-code editor in Java development, among other programming languages. It is a lightweight and extensible code editor developed by Microsoft. The main reason for using it as IDE for this project is its powerful code editing features, and it has a vast extension ecosystem that allows developers to customize their environment according to their needs. With this, it has excellent support for Java development through various extensions that could be considered tools.

Before the development of this project, the “Java Extension Pack” is installed in the Visual Studio Code. The extensions in the “Java Extension Pack” provide features such as code editing, debugging, syntax highlighting, and code completion. These extensions enhance the Java development experience. Here are some of the tools and features it offers:

Tools	Features
IntelliSense:	This tool provides intelligent code completion suggestions as we type, making it easier to write Java code by offering relevant classes, methods, and variables
Code Navigation	The extension allows us to navigate through our Java codebase efficiently. We can go to the definition of classes, methods, and variables, and quickly jump between different parts of our code
Code Formatting	It provides automatic code formatting capabilities, allowing us to format our Java code according to predefined style guidelines or our custom configuration
Code Snippets	The extension includes a collection of code snippets for common Java constructs. These snippets can be quickly inserted into our code,

	saving us time and effort
Debugging	It integrates with the debugger for Java, enabling us to debug our Java applications directly from Visual Studio Code. We can set breakpoints, inspect variables, and step through our code for effective debugging
Testing Support	The extension supports running and debugging JUnit and TestNG tests within Visual Studio Code. We can execute our unit tests and view the results directly in the IDE

These tools successfully improved our productivity and streamline our Java development workflow within the Visual Studio Code IDE.

Features Implemented

- Classes
 - There are four classes in this mini project which are Person, Administrator, Patient and Doctor.
- Inheritance
 - Person class is the parent class for the rest three classes which are Administrator, Patient and Doctor.

```
public class Person {
```

Figure 2: Parent Person Class

```
public class Administrator extends Person{
```

Figure 3: Child Administrator Class

```
public class Doctor extends Person{
```

Figure 4: Child Doctor Class

```
public class Patient extends Person{
```

Figure 5: Child Patient class

- Polymorphism

- Person class provides a method called Display(), the Display() method displays different information based on the class the Display() method in. For example, Display() method in Administrator class displays all the patients and doctors' info but in Patient class the method only displays the currently logged in patient's name and his assigned doctor's name.

```
public void Display(Vector<Doctor> doctors, Vector<Patient>patients){
    System.out.println("\n-----");
    System.out.println("Display Patient Info." );
    System.out.println("-----");
    for(int i = 0; i<patients.size(); i++){
        System.out.print("Patient No." + (i+1));
        System.out.print(patients.get(i).toString()+ "\n");
    }
    System.out.println("-----");
    System.out.println("Display Doctor Info." );
    System.out.println("-----");
    for(int i = 0; i<doctors.size(); i++){
        System.out.println("Doctor No." + (i+1));
        System.out.print(doctors.get(i).toString()+ "\n");
    }
}
```

Figure 6: Display() for Administrator Class

```
public void Display(){
    System.out.println("\nPatient Information");
    System.out.println("Patient name: " + Name);
}
```

Figure 7: Display() for Patient Class

- Associations

- The Doctor class and Patient class associate with each other. This technique can be shown by having Doctor class have an instance of patient and Patient class have an instance of doctor.

```
public class Doctor extends Person{
    private String Name;
    private String Speciality;
    Vector<Patient> assignPatients= new Vector<>();
}
```

```
public class Patient extends Person{
    private String Name;
    private Doctor assignedDr;
```

Figure 8: Both Classes have Instance of Each Other

- Scanner input
 - This system is able to let users input their information to register as either Administrator, Doctor or Patient. We using java.util.Scanner package to achieve receiving user input

```
import java.util.Scanner;
```

```
Scanner input = new Scanner(System.in);
```

```
String PaAccName=null;
System.out.println("\n=====");
System.out.println("Patient register page");
System.out.println("=====");
System.out.println("Enter your account name:");
PaAccName = input.nextLine();
```

Figure 9: Example of Using Scanner to Receive User Input

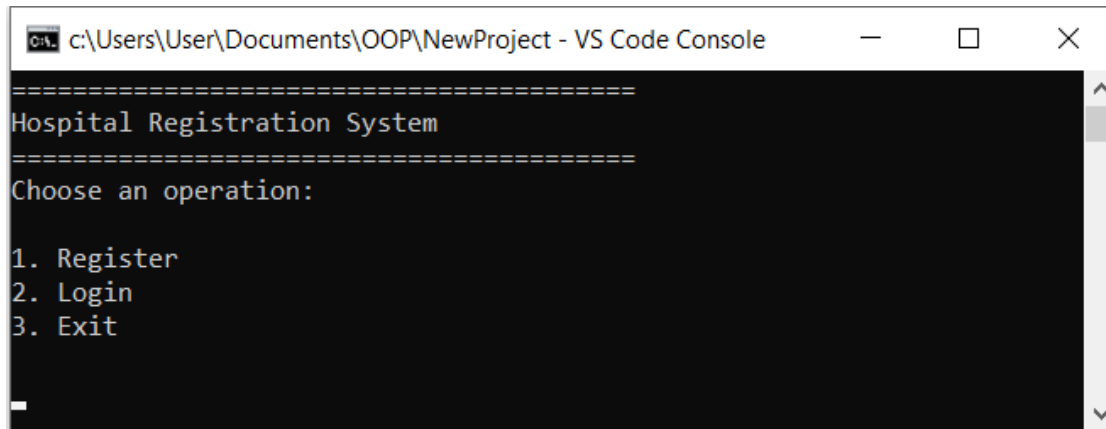
- Array to store objects
 - Since one doctor is able to have multiple patients at one time. So we are using vectors to store patients in Doctor class.

```
import java.util.Vector;
```

```
Vector<Patient> assignPatients= new Vector<>();
```

Figure 10: Example of Using Vector to Store Patient in Doctor Class

- ❖ Error handling
 - We are using an error handling technique to validate the input of users to prevent any false input entered by users.
 - In the system, there are two types of error handling techniques implemented. They are “Try and Catch” statement and if statement
 - These both types error handling techniques have their own implementation in different part of the system
 - “Try and Catch” syntax are used in the option input section while if-else statement is used in procedures validation section to achieve certain functionality
 - The below would explain how these error handling techniques implemented:
- ❖ “Try and Catch” syntax
 - The below are the examples of option selection interface:

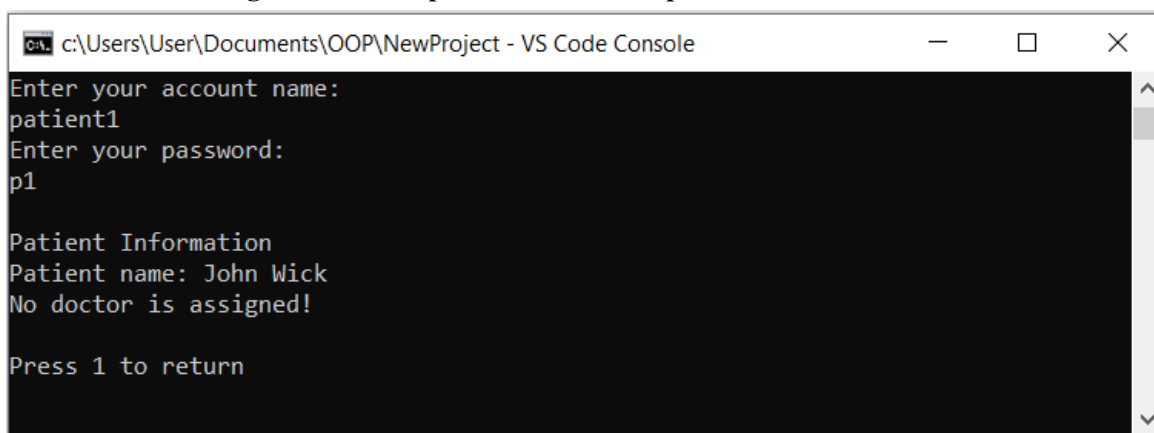


```
c:\Users\User\Documents\OOP\NewProject - VS Code Console

=====
Hospital Registration System
=====
Choose an operation:

1. Register
2. Login
3. Exit
```

Figure 11: Example of First Kind Option Selection Section



```
c:\Users\User\Documents\OOP\NewProject - VS Code Console

Enter your account name:
patient1
Enter your password:
p1

Patient Information
Patient name: John Wick
No doctor is assigned!

Press 1 to return
```

Figure 12: Example of Second Kind Option Selection Section

- The option input section is designed to accept integer type input
- If the user enters the input that is in the type other than integer, an `InputMismatchException` would occur
- “Try and Catch” syntax is used to handle this exception
- The try statement would terminate if the exception occurs
- Then the catch statement would be executed.
- In the catch statement, the user would be kept remained in the option input section until they enter input in integer type

```
import java.util.InputMismatchException;
```



```

while(operationLoop){
    int Choice1=0;
    boolean InputCorrect1 =false;
    while(!InputCorrect1){
        try{
            mainMenu();
            Choice1=input.nextInt();
            InputCorrect1=true;
        }catch(InputMismatchException ex){
            input.nextLine();
        }
    }
}

```

Figure 13: Example of Using Try and Catch to Validate User Input

❖ **“If” statement**

- If statement is implemented to ensure the system procedures to be executed according to the business regulations
- It is implemented in different procedures in the system
- The below would describe how “if else statement” ensure validity of different procedures:
 - User register procedure

```

=====
Patient register page
=====
Enter your account name:
p1

=====
The account has existed.Please register a new account
Click 1 to exit the page or click any button if want to retry

=====
Enter your account name:
=====
p1

=====
The account has existed.Please register a new account
Click 1 to exit the page or click any button if want to retry
1
=====
Hospital Registration System
=====
Choose an operation:

1. Register
2. Login
3. Exit

```

Figure 13: Example of Registration Procedure Validation

- In the system, there is no possibility to have at least 2 account that have same username
- If an existed account is registered again, an error message would be displayed to notify the user that the account has existed
- The user could retry the procedure or exit to the main menu

```

49 case 1: //if Choice2 is first choice - Patient
50 String PaAccName=null;
51 //Patient Registration Start-----
52 System.out.println("\n=====");
53 System.out.println("Patient register page");
54 System.out.println("=====");
55 System.out.println("Enter your account name:");
56 PaAccName = input.nextLine();
57 if(!patients.isEmpty()){ //if there is none of patients
58     while(LoopFlag){ //Section Looping
59         for(int i=0;i<patients.size();i++){ //Looping through patient vector
60             if(patients.get(i).getAccName().equals(PaAccName)){ //Error Display if acc registered exists
61                 //Error Message Displaying-----
62                 System.out.println("\n=====");
63                 System.out.println("The account has existed.Please register a new account");
64                 System.out.println("Click 1 to exit the page or click any button if want to retry");
65                 if (input.nextLine().equals(anObject+"1")) {
66                     Terminate = true; //Set Terminate to true to exit to the main menu
67                     LoopFlag = false; // Set LoopFlag to false to exit the outer while loop
68                     break;
69                 }
70
71                 System.out.println("\n=====");
72                 System.out.println("Enter your account name:");
73                 System.out.println("=====");
74             }
75             PaAccName = input.nextLine();
76             LoopFlag=true;
77             break;
78         }
79         LoopFlag=false; //Deactivate Section Looping Flag
80     }
81 }
82 }
83
84 }
85 if(Terminate){
86     break;
87 }
88 System.out.println("Enter your password:");
89 String PaPwd = input.nextLine();
90 System.out.println("Enter your name:");
91 String PaName = input.nextLine();
92
93 admin.addPatient(patients,new Patient(PaAccName, PaPwd, PaName)); //Add new patient into patient vector
94 System.out.println("\nPatient registered successfully!\n");
95 break;
96 //Patient Registration End

```

Figure 14: Programming implementation of the Registration Procedure

- According to the implementation above, if condition in Line 57 is implemented to check the emptiness of the patient list
- If the patient vector is not empty, the if statement would be executed but it would not be executed if patient vector is empty
- In other way, the checking procedures of registration information is not executed for the first registered user but it is executed for subsequent users

- For the if statement that start from line 60, the existence of registered account is checked before registering the account by referring the account name variable
- If the registered account existed, then an error message is displayed and the user would choose to retry the process or exit to the main menu

■ User login procedure

```
Enter your account name:
p1
Enter your password:
12
Invalid input!
Click 1 to exit the page or click any button if want to retry

Enter your account name:
p1
Enter your password:
12
Invalid input!
Click 1 to exit the page or click any button if want to retry
1
```

Figure 15: Example of Login Procedure Validation

- In the system, the account is logged in by using the information during registration which is account name and password
- If both of these are not matched, an error message would be displayed to notify the user that the input is invalid
- Then the user would choose to retry the process or exit to the main menu

```

193 System.out.println(x:"Enter your account name: ");
194 String PaAccName = input.nextLine();
195 System.out.println(x:"Enter your password: ");
196 String PaPwd = input.nextLine();
197
198 boolean PaAccNameMatch = false; //Error Flag for Account Name Not Matching
199 boolean PaPasswordMatch = false; //Error Flag for Password Not Matching
200 int PaIndex = -1;
201 while ((!PaPasswordMatch || !PaAccNameMatch)&&!Terminate) { //Password or Account Mismatch Looping
202     for (int i = 0; i < patients.size(); i++) { //Looping through Patient vector
203         if (PaPwd.equals(patients.get(i).getpwd())){ //If Password is matched
204             PaIndex = i; //Initialize to the index where password is matched
205             if(PaAccName.equals(patients.get(PaIndex).getAccName())){ //If Account is matched also
206                 PaAccNameMatch = true; //Deactivate Error Flag
207                 PaPasswordMatch = true;
208                 break;
209             }
210         }
211     }
212     if (!PaPasswordMatch || !PaAccNameMatch) { //If there is none of account matched
213         //Error Message Displaying-----
214         System.out.println(x:"Invalid input!");
215         System.out.println(x:"Click 1 to exit the page or click any button if want to retry");
216         if (input.nextLine().equals(anObject:"1")) {
217             Terminate = true;
218             break;
219         }
220
221         //Error Message Displaying-----
222         //Procedure Repeat-----
223         System.out.println(x:"Enter your account name: ");
224         PaAccName = input.nextLine();
225         System.out.println(x:"Enter your password: ");
226         PaPwd = input.nextLine();
227         //Procedure Repeat-----
228     }
229 }
230 if(Terminate){
231     LoginLoop=false;
232     break;
233 }
234 Boolean PatientOperation = true; //Patient Session Termination Flag Initialization
235 //Patient Info Printing-----
236 System.out.println(patients.get(PaIndex).toString());
237 //Patient Info Printing-----
238 System.out.println(x:"Press 1 to return");
239 while(PatientOperation){ //Patient Session Looping
240     InputCorrect1=false; //Mismatch Input Flag Initialization
241     int back=0; //Back value
242     while(InputCorrect1==false){ //Input Retry Looping for Input
243         try{ //InputMismatchException Try and Catch
244             back = input.nextInt();
245             input.nextLine();

```

```

246         InputCorrect1=true;
247
248     } catch (InputMismatchException ex){
249         input.nextLine();
250     }
251 }
252 if(back == 1){                                     //if back value=1
253     LoginLoop = false;                             //Terminate Patient Operation & Login Session
254     PatientOperation=false;
255 }
256 }
257 //Patient Login End-----

```

Figure 16: Programming implementation of the Login Procedure

- According to the implementation above, if condition in Line 203 is implemented to check password is matched or not while the if condition in Line 205 is implemented to check account is matched or not
- For if condition in Line 212: if both of those above are not matched, an error message would be displayed
- Then the user would choose to retry the process or exit to the main menu

■ Administrator Management Procedure

```

There is none of patients or doctors or both in the system.Hence there 's no need to manage them

=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return

```

Figure 17: Example of Administrator Management Procedure Validation

- If there is none of patients or doctors or both in the system, a message would be displayed to notify the management feature is not available and the administrator could not get access to this functionality

```

if(patients.isEmpty()||doctors.isEmpty()){
    System.out.println(x:"There is none of patients or doctors or both in the system.Hence there 's no need to manage them"); //Message to avoid the admistrator to get access
    break;                                                         //to this management functionality to avoid
}                                                                    //unnecessary problems

```

Figure 18: Programming implementation of the Administrator Management Procedure

```
-----  
Doctor List  
-----  
Doctor No.1: p1  
  
Enter doctor No. to assign a doctor  
2  
Invalid doctor no. !  
  
Enter doctor No. to assign a doctor  
1
```

Figure 19: Example of Administrator Management Procedure Validation

- In assigning the patient to the doctor in administrator management, there is a list of doctor and patient to be displayed
- Based on the list, the administrator choose the doctor he would like to assign by input the index displayed on the list
- If the input is outside of the range of indexes displayed, an error message would be displayed
- Then the administrator would retry the procedure until he input the valid index

```
if(D>d.size() || D<0){//error message is printed out if the entered index number>doctor vector size  
    System.out.println(x:"Invalid doctor no. !");  
    inputCorrect = false;//set the boolean to false to remain in the while loop  
}
```

Figure 20: Programming implementation of the Administrator Management Procedure

```

-----
Doctor List
-----
Doctor No.1: p1
Doctor No.2: d2

Enter doctor No. to assign a doctor
2

-----
Patient List
-----
Patient No.1: name1
Patient No.2: p2

Enter patient No. to assign doctor to that patient
1

-----
This patient has already been assigned a doctor. If you want to assign another new doctor to him, please remove the current responsible doctor
The current responsible doctor is p1
-----

=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return

2
Enter [add]/[remove] to assign/remove
add

-----
Doctor List
-----
Doctor No.1: p1
Doctor No.2: d2

Enter doctor No. to assign a doctor
1

-----
Patient List
-----
Patient No.1: name1
Patient No.2: p2

Enter patient No. to assign doctor to that patient
1
The doctor has already been assigned to this patient

```

Figure 21: Example of Administrator Management Procedure Validation

- If the administrator assigns the patient who has assigned to a doctor to another doctor, then a message would be displayed to state that the patient has a doctor assigned to him and it also display the name of responsible doctor of that patient
- If the administrator assigns the patient to the responsible doctor, then a message would be displayed to state that the patient has been assigned to that doctor

```
public boolean addDoctor(Doctor dr){  
    //uml changes  
    if(assignedDr!=null){  
        //if the patient already has a doctor  
        if(assignedDr==dr){  
            //if the same doctor assigned to the same patient  
            System.out.println(x:"The doctor has already been assigned to this patient");  
            return false;  
        }  
        //if the patient already assigned another doctor  
        System.out.println(x:"\n-----");  
        System.out.println(x:"This patient has already been assigned a doctor. If you want to assign another new doctor to him, please remove the current responsible doctor");  
        System.out.println("The current responsible doctor is "+assignedDr.getAccName());  
        System.out.println(x:"\n-----");  
        return false;  
    }  
    //if the patient has no doctor  
    assignedDr = dr;  
    System.out.println(x:"Patient assigned successfully!");  
    return true;  
}
```

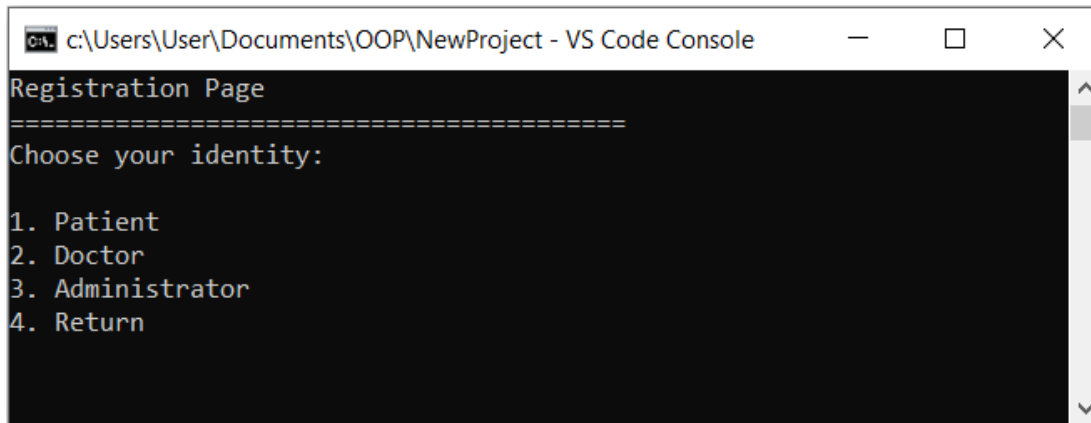
Figure 22: Programming implementation of the Administrator Management Procedure

User Interface

```
c:\Users\User\Documents\OOP\NewProject - VS Code Console  
=====  
Hospital Registration System  
=====  
Choose an operation:  
  
1. Register  
2. Login  
3. Exit
```

Figure 23: Main user interface

This interface is the main user interface. It will be prompted once the program is run. It allows the user to choose one of the operations provided: Register, Login and Exit by entering the corresponding number.

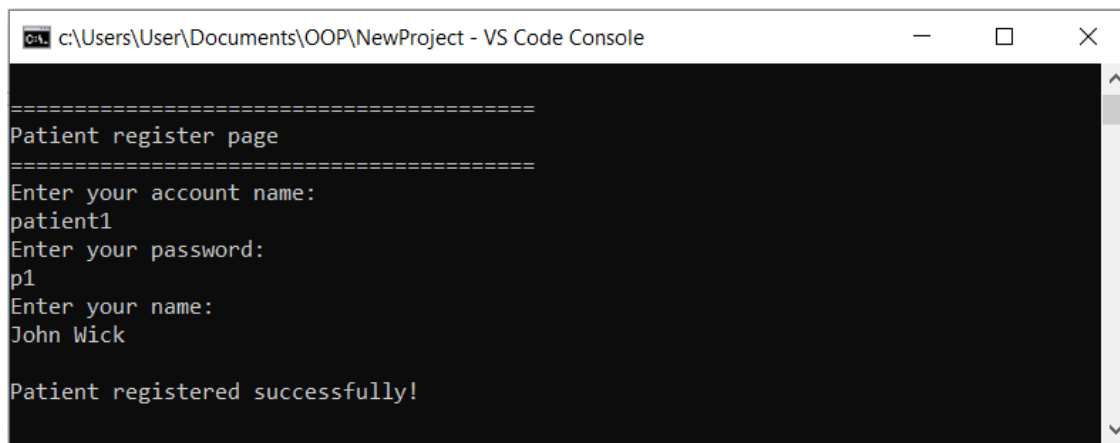
A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console displays the following text:

```
Registration Page
=====
Choose your identity:

1. Patient
2. Doctor
3. Administrator
4. Return
```

Figure 124: Registration user interface

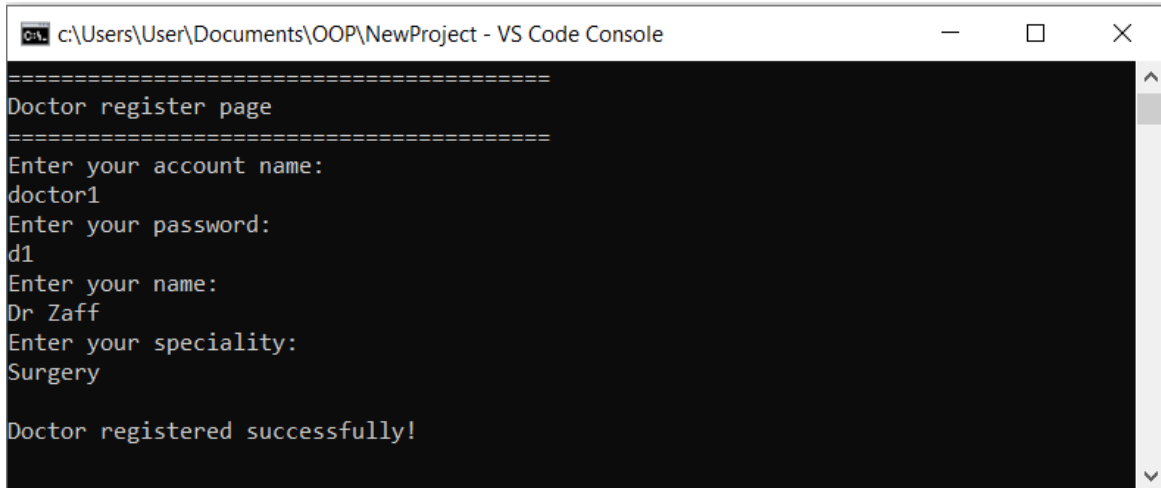
This interface is the registration user interface. It will be prompted after the user entered “1” at the main UI (“Register” operation is chosen). In this UI, the user needs to choose his/her identity, which is Patient, Doctor or Administrator by entering the corresponding number. Besides, the user may return to the main UI by entering the number “4”.

A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console displays the following text:

```
=====
Patient register page
=====
Enter your account name:
patient1
Enter your password:
p1
Enter your name:
John Wick
Patient registered successfully!
```

Figure 25: Patient registration user interface

This interface is the patient registration user interface. It will be prompted after the user entered “1” at the registration UI (Patient identity is chosen). In this UI, the user has to enter the account name, password and name. Once the required information is filled in, the message “Patient registered successfully !” is shown.

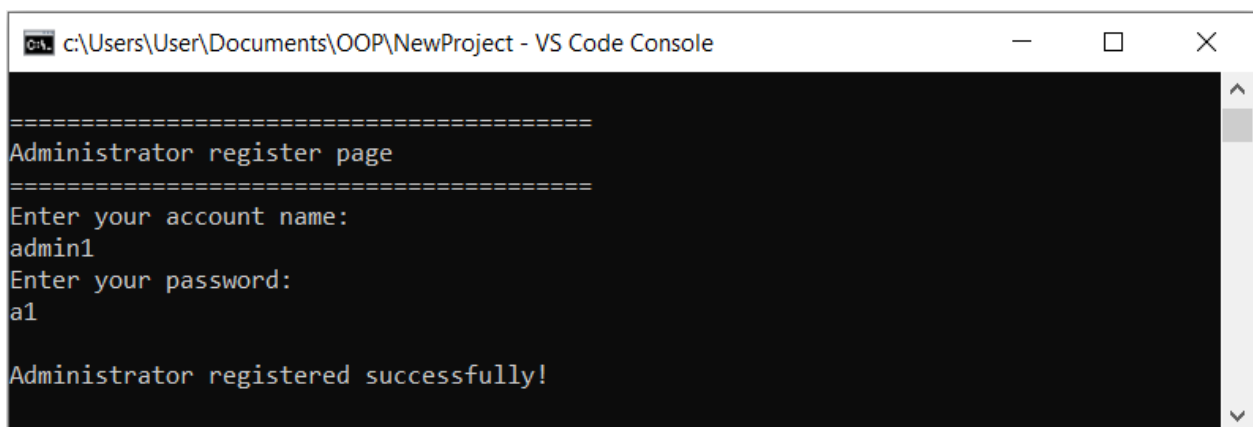
A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console output shows the "Doctor register page" with prompts for account name, password, name, and speciality. The user has entered "doctor1", "d1", "Dr Zaff", and "Surgery". The final message is "Doctor registered successfully!".

```
=====
Doctor register page
=====
Enter your account name:
doctor1
Enter your password:
d1
Enter your name:
Dr Zaff
Enter your speciality:
Surgery

Doctor registered successfully!
```

Figure 26: Doctor registration user interface

This interface is the doctor registration user interface. It will be prompted after the user entered “2” at the registration UI (Doctor identity is chosen). In this UI, the user has to enter the account name, password, name and also the speciality. Once the required information is filled in, the message “Doctor registered successfully !” is shown.

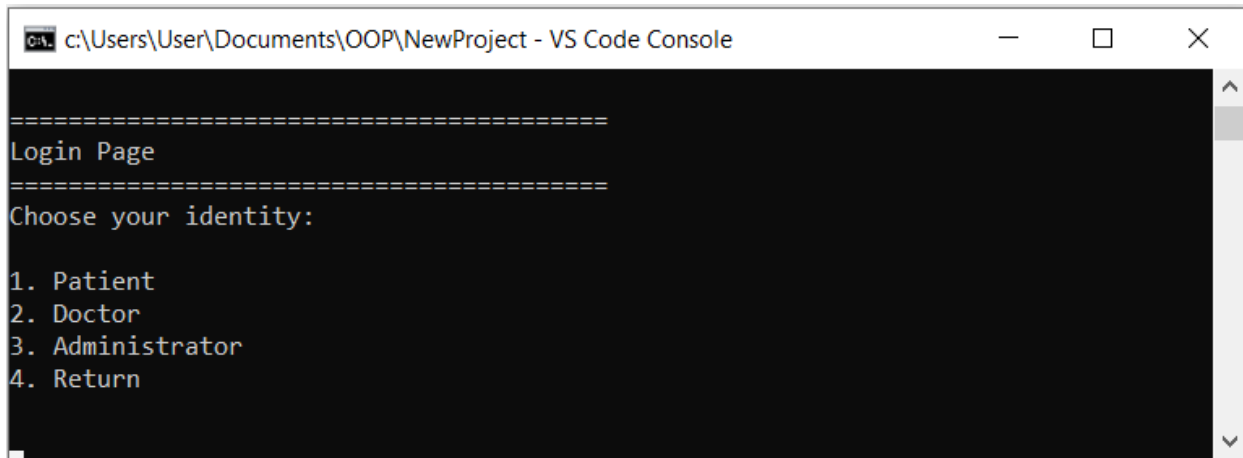
A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console output shows the "Administrator register page" with prompts for account name and password. The user has entered "admin1" and "a1". The final message is "Administrator registered successfully!".

```
=====
Administrator register page
=====
Enter your account name:
admin1
Enter your password:
a1

Administrator registered successfully!
```

Figure 27: Administrator registration page

This interface is the administrator registration user interface. It will be prompted after the user entered “3” at the registration UI (Administrator identity is chosen). In this UI, the user has to enter the account name and password. Once the required information is filled in, the message “Administrator registered successfully !” is shown.

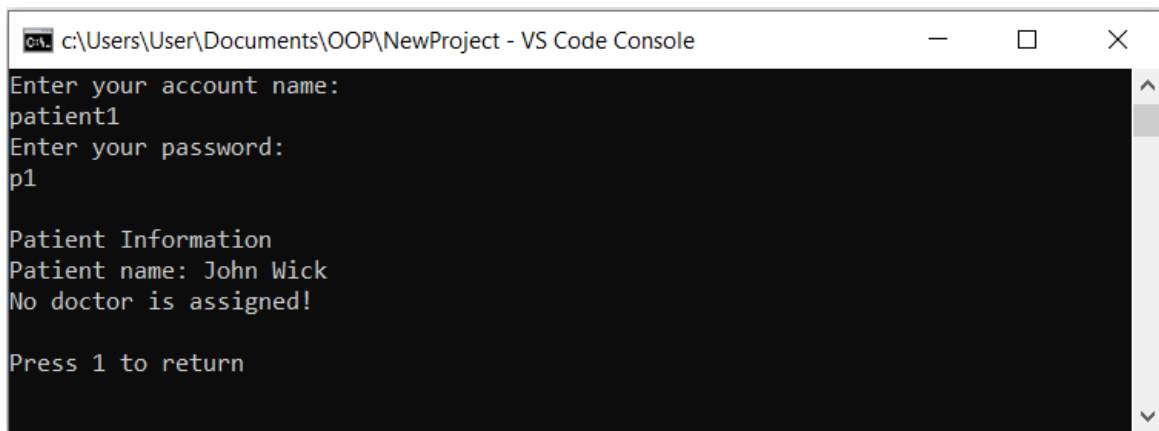
A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console displays a login menu with the following text: "=====
Login Page
=====
Choose your identity:

1. Patient
2. Doctor
3. Administrator
4. Return". The console has a dark background and a light-colored scrollbar on the right side.

```
=====  
Login Page  
=====  
Choose your identity:  
  
1. Patient  
2. Doctor  
3. Administrator  
4. Return
```

Figure 28: Login user interface

This interface is the login user interface. It will be prompted after the user entered “2” at the main UI (“Login” operation is chosen). In this UI, the user also needs to choose his/her identity, which is Patient, Doctor or Administrator by entering the corresponding number. Besides, the user may return to the main UI by entering the number “4”.

A screenshot of a VS Code console window titled "c:\Users\User\Documents\OOP\NewProject - VS Code Console". The console displays the following text: "Enter your account name:
patient1
Enter your password:
p1

Patient Information
Patient name: John Wick
No doctor is assigned!

Press 1 to return". The console has a dark background and a light-colored scrollbar on the right side.

```
Enter your account name:  
patient1  
Enter your password:  
p1  
  
Patient Information  
Patient name: John Wick  
No doctor is assigned!  
  
Press 1 to return
```

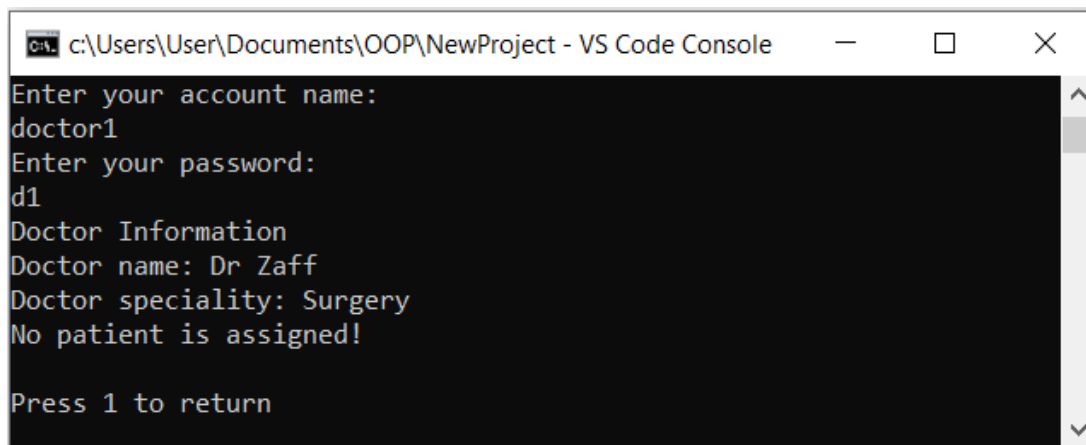
Figure 29.1: Patient login user interface - Valid input

```
Enter your account name:
patient1
Enter your password:
p2
Invalid input!
Click 1 to exit the page or click any button if want to retry

Enter your account name:
_
```

Figure 29.2: Patient login user interface - Invalid input

This interface is the patient login user interface. It will be prompted after the user entered “1” at the login user interface (Login as a patient). The user is required to enter his/her account name and password to log in to his/her account. If the entered data is valid, the user information, including the patient's name and assigned doctor's name will be displayed (if no doctor is assigned, a message “No doctor is assigned!” will be prompted), as shown in Figure 18.1. Otherwise, an error message “Invalid input!” will be prompted and the system will ask the user to retry or exit to the main menu. If the user wants to retry, he would reenter the account name and password, as shown in Figure 29.2. Once the corresponding patient information is displayed, the user may enter “1” to return to the main UI.



```
c:\Users\User\Documents\OOP\NewProject - VS Code Console
Enter your account name:
doctor1
Enter your password:
d1
Doctor Information
Doctor name: Dr Zaff
Doctor speciality: Surgery
No patient is assigned!

Press 1 to return
```

Figure 30.1: Doctor login user interface - Valid input

```
Enter your account name:
doctor1
Enter your password:
d2
Invalid input!
Click 1 to exit the page or click any button if want to retry

Enter your account name:
```

Figure 30.2: Doctor login user interface - Invalid input

This interface is the doctor login user interface. It will be prompted after the user entered “2” at the login user interface (Login as a doctor). The user is required to enter his/her account name and password to log in to his/her account. If the entered data is valid, the user information, including the doctor’s name, doctor’s speciality and assigned patient’s name will be displayed (if no patient is assigned, a message “No patient is assigned!” will be prompted), as shown in Figure 19.1. Otherwise, an error message “Invalid input!” will be prompted and the system will ask the user to retry or exit to the main menu. If the user wants to retry, he would reenter the account name and password, as shown in Figure 30.2.. Once the corresponding doctor information is displayed, the user may enter “1” to return to the main UI.

```
c:\Users\User\Documents\OOP\NewProject - VS Code Console
Enter your account name:
admin1
Enter your password:
a1

=====
Administrator operation page
=====
Choose an operation to perform

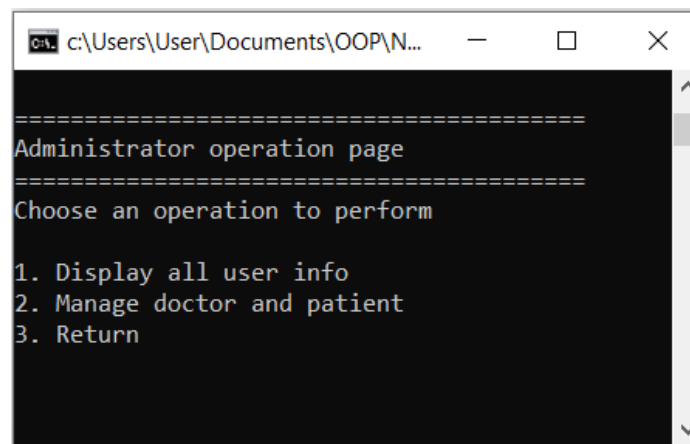
1. Display all user info
2. Manage doctor and patient
3. Return
```

Figure 31.1: Administrator login user interface - Valid input

```
Enter your account name:
admin1
Enter your password:
a2
Invalid input!
Click 1 to exit the page or click any button if want to retry
Enter your account name:
_
```

Figure 31.2: Administrator login user interface - Invalid input

This interface is the administrator login user interface. It will be prompted after the user entered “3” at the login user interface (Login as an administrator). The user is required to enter his/her account name and password to log in to his/her account. If the entered data is valid, the “Administrator operation user interface” will be displayed (this interface will be explained below), as shown in Figure 20.1. Otherwise, an error message “Invalid input!” will ask the user to retry or exit to the main menu. If the user wants to retry, he would reenter the account name and password, as shown in Figure 30.2.

A screenshot of a Windows command prompt window. The title bar shows the file path "c:\Users\User\Documents\OOP\N...". The window contains the following text: "=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return". The text is displayed in a monospaced font on a black background.

```
=====  
Administrator operation page  
=====  
Choose an operation to perform  
  
1. Display all user info  
2. Manage doctor and patient  
3. Return
```

Figure 32: Administrator operation user interface

The Administrator operation user interface will be prompted once the administrator login information is valid. At this interface, the user may enter the number:

- “1” to display all user information
- “2” to Manage doctor and patient (Assign doctor to patient / Remove patient from a doctor)
- “3” to return to the main UI

```
c:\Users\User\Documents\OOP\NewProject - VS Code Console
-----
Display Patient Info.
-----
Patient No.1
Patient Information
Patient name: John Wick
No doctor is assigned!

-----
Display Doctor Info.
-----
Doctor No.1
Doctor Information
Doctor name: Dr Zaff
Doctor speciality: Surgery
No patient is assigned!

=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return

-
```

Figure 33: User Information user interface

This interface is the user information user interface. It will be prompted after the user entered “1” at the administrator operation user interface. All the users' (patients & doctors) information will be displayed in this user interface. After displaying the user information, the Administrator operation user interface will be redisplayed.


```
c:\Users\User\Documents\OOP\NewProject - VS Code Console
Enter [add]/[remove] to assign/remove
add

-----
Doctor List
-----
Doctor No.1: Dr Zaff

Enter doctor No. to assigned a doctor
1

-----
Patient List
-----
Patient No.1: John Wick

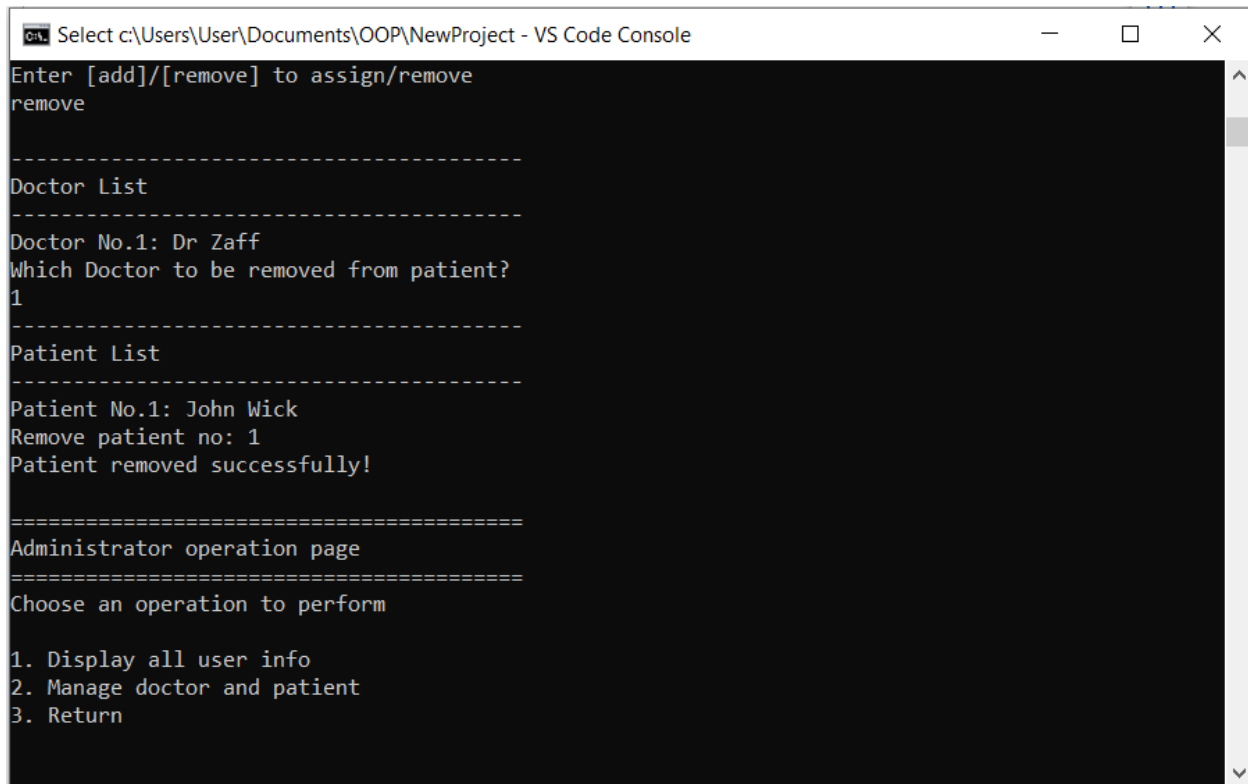
Enter patient No. to assign doctor to that patient
1
Patient assigned successfully!

=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return
```

Figure 34.1: Manage doctor and patient user interface – assign a doctor to patient

This interface is the “Manage Doctor and Patient” user interface. It will be prompted after the user entered “2” at the administrator operation user interface. The user needs to enter the word “add” or “remove” to assign a doctor to a patient or remove a patient from a doctor. In this figure, the user enters “add” and then a doctor list is displayed. Next, the system will ask the user to enter doctor no. to assign the doctor to a patient. After selecting a doctor, a patient list will be displayed. The user also needs to enter patient no. to assign that patient the selected doctor. Finally, a message “Patient assigned successfully” is prompted and the administrator operation user interface is redisplayed.



```
Select c:\Users\User\Documents\OOP\NewProject - VS Code Console
Enter [add]/[remove] to assign/remove
remove

-----
Doctor List
-----
Doctor No.1: Dr Zaff
Which Doctor to be removed from patient?
1
-----
Patient List
-----
Patient No.1: John Wick
Remove patient no: 1
Patient removed successfully!

=====
Administrator operation page
=====
Choose an operation to perform

1. Display all user info
2. Manage doctor and patient
3. Return
```

Figure 34.2: Manage doctor and patient user interface – remove a patient from the doctor

This interface is the “Manage Doctor and Patient” user interface. It will be prompted after the user entered “2” at the administrator operation user interface. The user needs to enter the word “add” or “remove” to assign a doctor to a patient or remove a patient from a doctor. In this figure, the user enters “remove” and then a doctor list is displayed. Next, the system will ask the user to enter doctor no. to remove the doctor from a patient. After selecting a doctor, a patient list will be displayed. The user also needs to enter patient no. to remove that patient from the selected doctor. Finally, a message “Patient removed successfully” is prompted and the administrator operation user interface is redisplayed.

Conclusion

In conclusion, this Java project has been a successful endeavor that has achieved its objectives and enhanced our Java knowledge. Throughout the development process, we utilized the powerful features and capabilities of Java to create an efficient and useful program. Although this is only a mini program, we can sense what Java can achieve if we plan to do bigger and more complex programs. We appreciate being able to do this mini project to learn more deeply about Java programming and programming techniques of Java.