Assignment

(SE-313)

Formal Methods in Software Engineering

NED University of Engineering and Technology
Karachi- 75270, Pakistan

Department of Software Engineering

Sara Naeem Aslam (SE-21015)

Meerab Tahir (SE-21019)

Section: A

Batch: 2021

Submitted To: Dr. Mustafa Latif

PATIENT MANAGEMENT SYSTEM

1. INTRODUCTION:

The Patient Management System is a comprehensive solution aimed at streamlining the registration process for surgical procedures. With a maximum capacity of 200 patients, the system is designed to efficiently handle the addition and removal of patients from the surgical register. Its primary objective is to provide a user-friendly interface for healthcare professionals to manage patient information seamlessly.

The Patient Management System ultimately aims to optimize patient registration processes, contribute to effective surgical planning, and improve the overall quality of healthcare services.

2. SCOPE:

The system is designed to facilitate the registration of patients for surgical procedures. It is assumed that the surgical capacity of the system is limited to a maximum of 200 patients on its register.
The system must have the capability to add and remove patients from the surgical register. Additionally, the register should support inquiries, allowing retrieval of the list of patients and the total number of registered patients.
Also, a check can be made to check if the patient is registered.

2.1. ADD PATIENT:

This function enables healthcare professionals to input and register new patients into the system, ensuring accurate and up-to-date information for surgical procedures.

2.2. REMOVE PATIENT:

This feature allows authorized users to efficiently remove patients from the surgical register, maintaining a dynamically updated and manageable patient list for better resource allocation.

2.3. VIEW PATIENT RECORD:

Users can access and review detailed patient records, facilitating informed decision-making by providing comprehensive insights into each patient's history and current status within the system.

3. SYSTEM INVARIANT:

The system has a constraint where it can only accommodate the addition of up to 200 patients to the surgical register at any given instance.

4. 4+1 ARCHITECTURE:

4.1. LOGICAL VIEWPOINT:

The logical view focuses on the conceptual aspect of the system.
The presented class diagram illustrates all the classes associated with the system, along with their respective attributes.

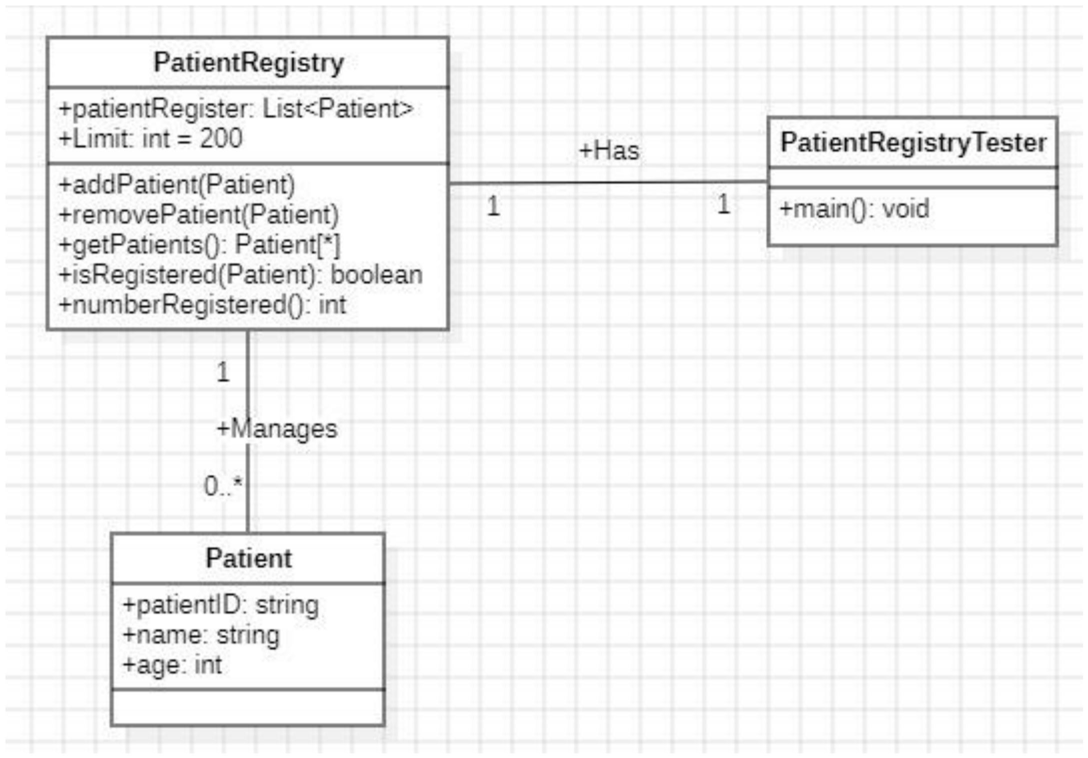

Fig 1: Class diagram of patient management system

EXPLANATION

Fig 1 displays the class diagram of a patient management system having three classes PatientRegistry, Patient, and PatientRegistryTester. PatientRegistry class maintains a limit of 200 patients. It provides methods to add, remove, retrieve the list, get the total number of patients, and check if a patient is registered. PatientRegistryTester is a testing class with a main method to test the functionality of the PatientRegistry class. The Patient class is a static nested class within the PatientRegistry class having attributes patientID, name, and age

4.2. PROCESS VIEWPOINT:

The process view illustrates the dynamic aspect of the system.
The provided sequence diagram delineates the chronological sequence of events unfolding within the system.

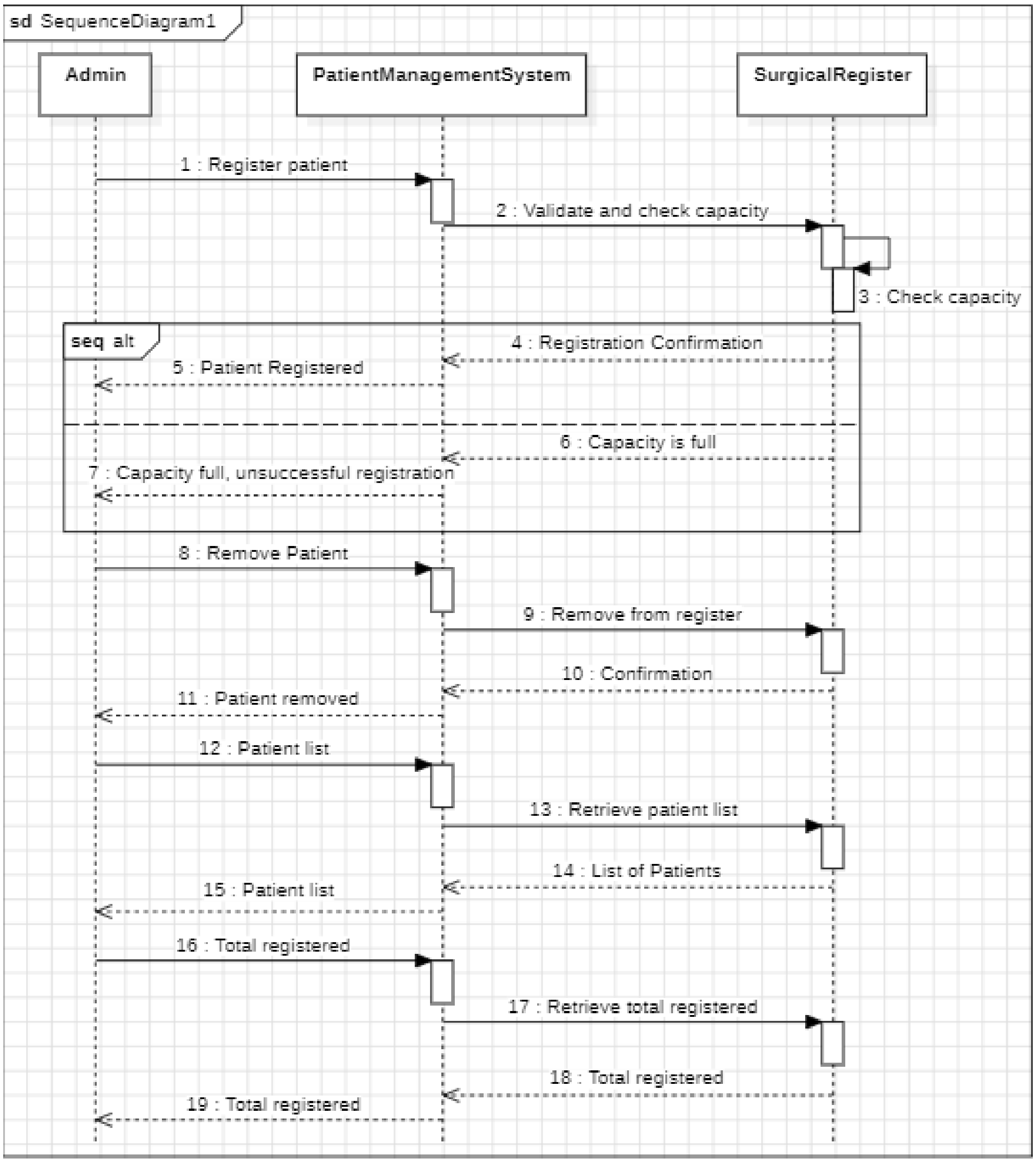

Fig 2: Sequence diagram of patient management system

EXPLANATION

Fig. 2 displays the sequence diagram featuring three entities: admin, patient management system, and surgical register. The admin registers the patient in the patient management system. The system checks if the capacity is less than 200; if so, it registers the patient. Otherwise, it displays 'capacity is full' and indicates an unsuccessful registration. If the admin wishes to remove a patient, the system will remove the patient from the surgical register and display a confirmation. Similarly, if the admin wants to see the list of registered patients, the system will retrieve patients from the surgical register and display them to the admin. The same process applies if the admin wants to see the total number of patients. The system retrieves the total number of patients from the surgical register and displays it to the admin.

4.3. PHYSICAL VIEWPOINT:

This view details the system's deployment and distribution across physical hardware.

The deployment diagram displayed below visually represents all system components and their interrelationships in terms of deployment.

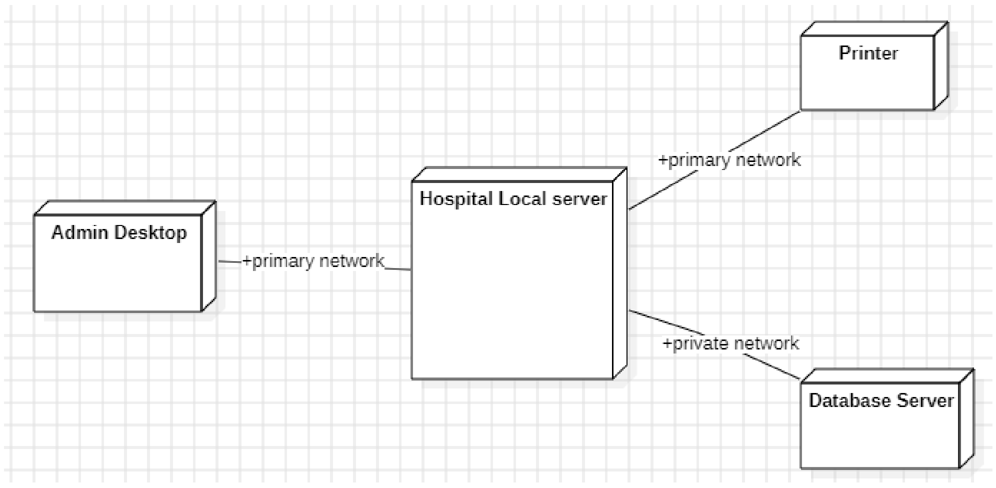

Fig 3: Deployment diagram of patient management system

EXPLANATION:

Figure 3 displays the deployment diagram featuring four nodes, illustrating the deployment structure of the patient management system. The process initiates at the desktop admin node, where the administrator arrives with their query. The request undergoes processing at the hospital's local server, with options to either be stored in a database server or printed out. The "<< >>" notation signifies the profile class, portraying the role of an existing met class within a profile.

4.4. DEVELOPMENT VIEWPOINT:

This view outlines the software modules and their organization during development.

The component diagram underneath illustrates all the system components associated with the Patient Management System.

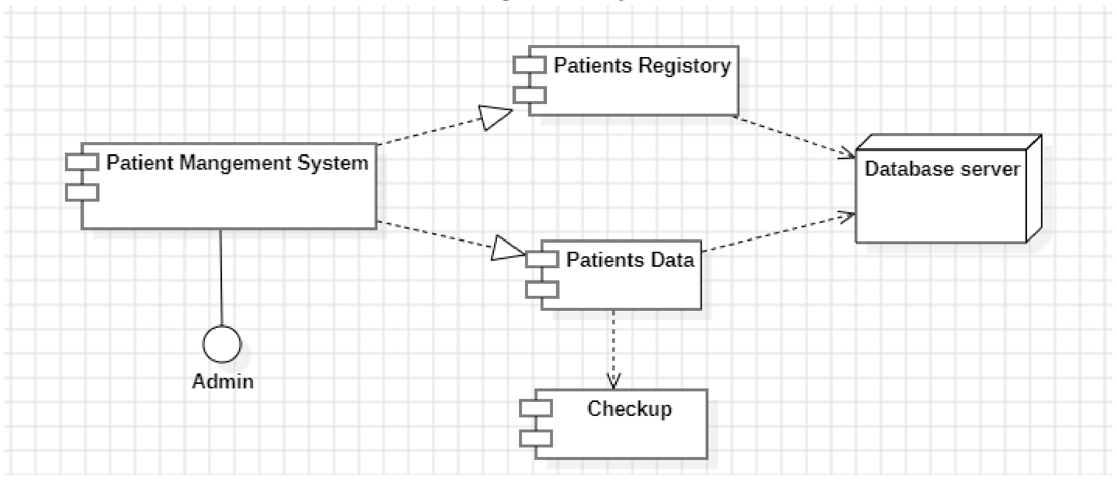

Fig 4: Component diagram of patient management system

EXPLANATION:

The component diagram illustrates the system's components, along with their provided and required interfaces, ports, and the relationships between patient data and the hospital. The key components in the diagram comprise:

- The System
- Database Server
- Patient Registry

The process involves the administration registering a patient by inputting the patient's record, which is subsequently stored in the hospital's database server.

4.5. +1 SCENARIO:

This view provides specific instances or scenarios to illustrate how the system works.

The depicted use case diagram outlines the actors and the various use cases involved in the system, illustrating their interrelations.

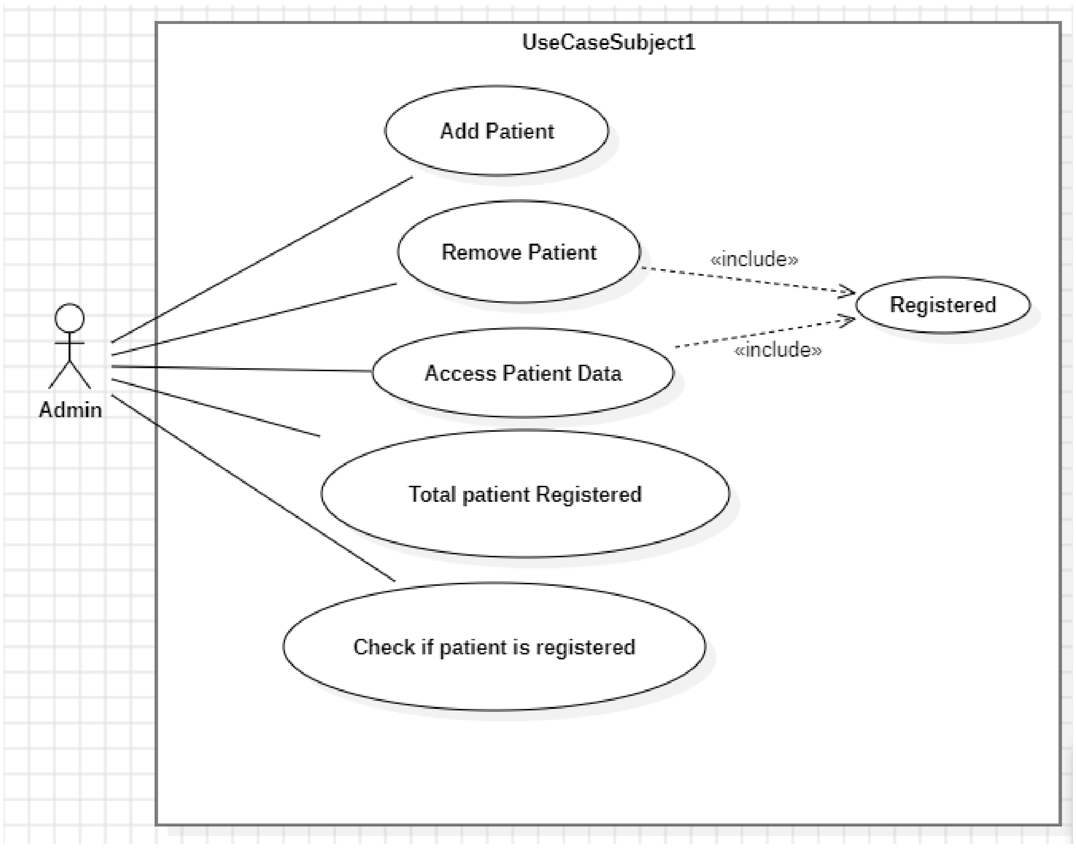

Fig 3: Showing Use case diagram of patient management system

EXPLANATION:

Figure 3 illustrates all the use cases and their interactions with the actor. The identified use cases encompass:

- Adding a patient record
- Deleting a patient record
- Viewing a patient record
- Checking if a patient is registered
- Determining the total number of registered users

The administration is tasked with overseeing the management of patient records.

5. VDM SPECIFICATION:

Utilizes the Vienna Development Method (VDM) to specify the system's types, values, state, and operations.

- Defines the “Patient” type and specifies operations for adding, removing, getting patients, checking registration, and getting the number of registered patients.

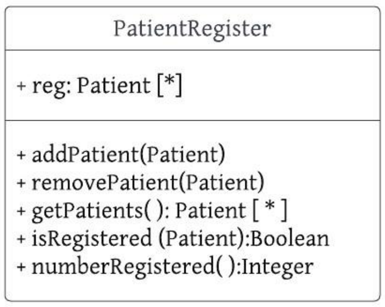
PatientRegister
+ reg: Patient [*]
+ addPatient(Patient)
+ removePatient(Patient)
+ getPatients( ): Patient [*]
+ isRegistered (Patient):Boolean
+ numberRegistered( ):Integer

Fig 1: Showing attributes and operation involve in VDM specification

types

Patient = TOKEN;

values

LIMIT: ℕ = 200

state PatientRegister **of**
reg: Patient-**set**
inv-mk-PatientRegister **(r)** ∆ **card r** ≤ *LIMIT*
inv-mk-PatientRegister **(r)** ∆ **= { }**
end

operations

- - an operation that add patient to the system

addPatient(patientIn: Patient)
ext wr reg: Patient-**set**
pre patient ∉ reg ∧ **card** reg < LIMIT
post reg = $\overline{\text{reg}}$ ∪ {patientIn}

- - an operation that remove patient to the system

removePatient(patientIn: Patient)
ext wr reg: Patient-**set**
pre patientIn ∈ reg
post reg = $\overline{\text{reg}}$ \{patientIn}

- - an operation that fetch patient record from the system

getPatients( ) output: Patient-**set**
ext rd reg: Patient-**set**
pre TRUE
post output **=** reg

- - an operation that check if patient registered

IsRegistered (patientIn: Patient) query: B
ext wr reg: Patient-**set**
pre TRUE
post query ⇔ patientIn **ϵ** reg

- - an operation that check no of patient registered

numberRegistered ( ) total: ℕ
ext rd reg: Patient-**set**
pre TRUE
post total = **card** reg

6. JAVA CODE IMPLEMENTATION:

The `**PatientRegistry**` class is implemented in Java, encapsulating patient registration functionality.
The class includes a nested `Patient` class, exception handling for registry full, and methods for adding, removing, and querying patient records.

PatientRegistry.java:

```
import java.util.HashSet;
import java.util.Set;

public class PatientRegistry {
    // Types
    public static class Patient {
        private String id;
        private String name;
        private int age;

        public Patient(String id, String name, int age) {
            this.id = id;
            this.name = name;
            this.age = age;
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null || getClass() != obj.getClass())
                return false;
            Patient patient = (Patient) obj;
            return id.equals(patient.id);
        }

        @Override
        public int hashCode() {
            return id.hashCode();
        }

        @Override
        public String toString() {
            return "Patient{" +
                    "id='" + id + '\'' +
                    ", name='" + name + '\'' +
                    ", age=" + age +
                    '}';}}
    // Values
    private static final int LIMIT = 200;

    public static int getLimit() {
        return LIMIT;
    }

    public static class RegistryFullException extends RuntimeException {
        public RegistryFullException(String message) {
            super(message);}}

    // State
    private Set<Patient> reg = new HashSet<>();

    // Operations
    public void addPatientRecord(String id, String name, int age) {
        Patient patient = new Patient(id, name, age);
        addPatient(patient);
    }

    public void deletePatientRecord(String id) {
        Patient patientToRemove = null;
        for (Patient patient : reg) {
            if (patient.id.equals(id)) {
                patientToRemove = patient;
                break; }}

        if (patientToRemove != null) {
            removePatient(patientToRemove);
            System.out.println("Patient Record Deleted Successfully!");
        } else {
            System.out.println("No such patient with ID '" + id + "' exists.");}}

    public void addPatient(Patient patientIn) {
        if (reg.size() < LIMIT) {
            if (!reg.contains(patientIn)) {
                reg.add(patientIn);
            } else {
                System.err.println("Error: Patient with ID " + patientIn.id + " is already
registered.");
            }
        } else {
            throw new RegistryFullException("Patient registry is full. Cannot add more
patients.");   }}

    public void removePatient(Patient patientIn) {
        reg.remove(patientIn);
    }
    public Set<Patient> getPatients() {
        return new HashSet<>(reg);
    }

    public boolean isRegistered(Patient patientIn) {
        return reg.contains(patientIn);
    }

    public int numberRegistered() {
        return reg.size();}}
```

7. JAVA TESTING:

The **`PatientRegistryTester`** class is implemented for testing the functionalities of the `PatientRegistry` class.
Provides a menu-driven interface for adding patients, deleting patients, viewing all patients, checking patient registration, and displaying the total number of registered patients.

PatientRegistryTester.java:

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Set;

public class PatientRegistryTester {

    public static void main(String[] args) {
        // Create an instance of the PatientRegistry
        PatientRegistry patientRegistry = new PatientRegistry();
        Scanner scanner = new Scanner(System.in);

        try {
            // Continuously display the menu until the user chooses to exit
            while (true) {
                // Display the menu options
                printMenu();

                int choice;
                try {
                    // Read the user's choice
                    choice = scanner.nextInt();
                } catch (InputMismatchException e) {
                    // Handle invalid input (non-integer)
                    System.err.println("Invalid input. Please enter a number.");
                    scanner.nextLine();  // Consume the invalid input
                    continue;
                }
                scanner.nextLine();  // Consume the newline character
                // Process the user's choice
                switch (choice) {
                    case 1:
                        addPatient(patientRegistry, scanner);
                        break;

                    case 2:
                        deletePatient(patientRegistry, scanner);
                        break;

                    case 3:
                        viewAllPatients(patientRegistry);
                        break;

                    case 4:
                        checkIfPatientRegistered(patientRegistry, scanner);
                        break;

                    case 5:
                        displayTotalPatients(patientRegistry);
                        break;

                    case 6:
                        exitProgram();
                        break;

                    default:
                        // Invalid choice
                        System.out.println("Invalid choice. Please enter a number between 1
and 6.");
                        break;
                }
            }
        } catch (Exception e) {
            // Handle unexpected errors
            System.err.println("An unexpected error occurred: " + e.getMessage());
        } finally {
            // Close the Scanner using try-with-resources
            scanner.close();}}

    // Display the menu options
    private static void printMenu() {
        System.out.println("\n ********** Patient Management System ********** ");
        System.out.println("\n1. Add Patient Record");
        System.out.println("2. Delete Patient Record");
        System.out.println("3. View All Patients");
        System.out.println("4. Check if Patient is Registered");
        System.out.println("5. Display Total Number of Registered Patients");
        System.out.println("6. Exit");
        System.out.print("Select an option (1-6): ");}

    private static void addPatient(PatientRegistry patientRegistry, Scanner scanner) {
        System.out.print("Enter Patient ID: ");
        String id = scanner.nextLine();
        System.out.print("Enter Patient Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Patient Age: ");
        int age = scanner.nextInt();
        patientRegistry.addPatientRecord(id, name, age);
        System.out.println("Patient Record Added Successfully!");}

    private static void deletePatient(PatientRegistry patientRegistry, Scanner scanner) {
        System.out.print("Enter Patient ID to Delete: ");
        String deleteId = scanner.nextLine();
        patientRegistry.deletePatientRecord(deleteId); }

    private static void viewAllPatients(PatientRegistry patientRegistry) {
        Set<PatientRegistry.Patient> patients = patientRegistry.getPatients();
        System.out.println("All Patients:");
        if (patients.isEmpty()) {
            System.out.println("No patients registered.");
        } else {
            for (PatientRegistry.Patient patient : patients) {
                System.out.println(patient);}}}

    private static void checkIfPatientRegistered(PatientRegistry patientRegistry,
Scanner scanner) {
        System.out.print("Enter Patient ID to Check: ");
        String patientId = scanner.nextLine();
        boolean isRegistered = patientRegistry.isRegistered(new
PatientRegistry.Patient(patientId, "", 0));

        if (isRegistered) {
            System.out.println("Patient with ID '" + patientId + "' is registered.");
        } else {
            System.out.println("Patient with ID '" + patientId + "' is not registered.");}}

    private static void displayTotalPatients(PatientRegistry patientRegistry) {
        int totalPatients = patientRegistry.numberRegistered();
        System.out.println("Total Number of Registered Patients: " + totalPatients);}

    private static void exitProgram() {
        System.out.println("Exiting Program. Goodbye!");
        System.exit(0);}}
```

OUTPUT:

```
********** Patient Management System **********
1. Add Patient Record
2. Delete Patient Record
3. View All Patients
4. Exit
Select an option (1-4): 1
Enter Patient ID: 210
Enter Patient Name: Muhammad Ali
Enter Patient Age: 35
Patient Record Added Successfully!

1. Add Patient Record
2. Delete Patient Record
3. View All Patients
4. Exit
Select an option (1-4): 3
All Patients:
Patient {id='210', name='Muhammad Ali', age=35}

1. Add Patient Record
2. Delete Patient Record3. View All Patients
4. Exit
Select an option (1-4): 2
Enter Patient ID to Delete: 210
Patient Record Deleted Successfully!

1. Add Patient Record
2. Delete Patient Record
3. View All Patients
4. Exit
```

8. TEST CASES:

Test cases are defined for adding patients, removing patients, viewing patient records, and checking patient registration.
Invalid and valid scenarios are considered to ensure proper functionality.

i. ADDING PATIENT:

Classes	**Invalid**	**valid**	**Invalid**
Range	LIMIT < 1	$1 \leq$ LIMIT ≤ 200	LIMIT > 200
Test cases	0	170	202
Expected Output	System should generate error	System should add the patient	System should generate error

ii. REMOVE PATIENT:

Classes	Invalid	valid
Test cases	Id != existing Id	Id = existing Id
Expected Output	System should generate error	System should remove the patient

iii. VIEW PATIENT:

Classes	Invalid	valid
Test cases	No of patient = 0	No of patient $\leq$ 200
Expected Output	No patient exist	Show the record of all patients.

iv. PATIENT REGISTRATION:

Classes	Invalid	valid
Test cases	Id != existing Id	Id = existing Id
Expected Output	No patient with such Id exist	System should show the registered patient

9. CONCLUSION:

Overall, the document presents a well-structured and detailed approach to designing, specifying, and implementing a Patient Management System, providing insights into its architecture and functionality. The inclusion of VDM specification and test cases enhances the clarity and completeness of the document.

10. GITHUB LINK:

https://github.com/MeerabTahir/PatientManagementVDM-