Object-Oriented Programming Project Report

BTech 4th Semester

2024-2025

Project Title: Hospital Management System

Submitted To: Dr. Deepak Kumar Sharma

Submitted By:

1. Ayush Gupta
   SapId: 500125537
   Roll No: R2142231692
2. Kanakpreet Kaur
   SapId: 500127119
   Roll No: R2142232117
3. Shashank Dimri
   SapId: 500124397
   Roll No: R2142231682
4. Anshika Tomar
   SapId: 500119658
   Roll No: R2142230736

Abstract

A Java-based program called the Hospital Management System was developed to simulate and streamline core administrative and patient management functions within a hospital setting. Unlike traditional systems that rely on complex databases, this application uses file handling techniques to store and manage essential data related to patients, their treatments, room assignments, and billing, offering a lightweight yet effective solution for handling hospital workflows.

The project embraces the principles of Object-Oriented Programming (OOP) to represent real-world hospital components, with particular emphasis on patient admission, discharge, and billing processes.. The system provides a robust set of features such as adding new patients, viewing patient records, updating billing details, and discharging patients—ensuring a seamless flow from admission to exit. Key functionalities like room tracking and billing management are tightly integrated using structured file I/O operations. Moreover, discharge and billing updates are automated through interaction between modular components, maintaining clean separation of concerns and promoting code maintainability.

Classes such as AddPatient, DischargePatient, BillingDetails, Room, and ViewPatients encapsulate the core logic of patient lifecycle management. The system also includes supporting modules like Department and Employee for administrative context, but the primary focus remains on patient-centered care and financial tracking.

In conclusion, this project effectively demonstrates how Java programming can be used to simulate real-life hospital management scenarios, showcasing mastery in OOP, file I/O, and logical control structures. It serves as a valuable academic exercise and provides a practical foundation for understanding how software can enhance administrative workflows in healthcare settings.

Acknowledgement

We would like to express our sincere gratitude to Dr. Deepak Kumar Sharma for his invaluable guidance, encouragement, and continuous support throughout the development of this project. His expert insights, patience, and dedication to teaching have played a crucial role in shaping our understanding and approach towards building this application.

Under his mentorship, we were able to gain hands-on experience in Java programming, file handling techniques, and object-oriented design. His feedback helped us improve the structure, logic, and efficiency of our code, making this project a strong learning experience as well as a practical implementation.

We also extend our heartfelt thanks to the entire faculty of our department for creating an academically rich and supportive environment. Their inspiration and mentorship have been vital to our growth.

Finally, we would like to acknowledge the teamwork and dedication of every member involved in this project. It is through shared effort, late-night discussions, and mutual support that we were able to complete this work successfully.

Team Members:

Ayush Gupta

Anshika Tomar

Shashank Dimri

Kanakpreet Kaur

Reason for choosing of the Healthcare domain -

Healthcare is a domain where accuracy, timeliness, and organisation are critical—especially when it comes to managing patient information. In many hospitals, particularly smaller ones or those in developing regions, administrative processes like patient admissions, billing, and discharge are still handled manually, which increases the risk of errors, delays, and miscommunication. This project was developed to tackle these real-world challenges by building a lightweight Hospital Management System using Java and file handling instead of relying on complex databases. The goal was to create a solution that is simple to deploy, easy to use, and effective in organising patient records and billing data.

By focusing on patient-centric features like admission tracking, billing updates, and room availability, the system aims to simplify hospital workflows, improve data reliability, and provide a practical tool for hospitals looking to digitalise their operations without major infrastructure changes.

Main Functionalities and Highlights of the code:-

- Patient Management
  - Add new patients with details like name, disease, and room assignment.
  - View complete patient data including room status and discharge updates.
  - Exception handling ensures invalid or incomplete data entries are caught and managed gracefully.
- Billing System
  - Update billing details based on patient stay and services.
  - Ensure accurate and up-to-date financial records for each patient.
  - Handles file I/O exceptions to prevent loss or corruption of billing data.
- Room Management
  - View room availability, type, and occupancy status.
  - Alert when a patient is assigned to an already occupied room.
  - Exceptions are managed for missing or malformed room data.
- File Handling
  - All data is saved and retrieved using text files, ensuring portability.
  - Allows backend editing if needed, while maintaining structure and security.
  - Robust exception handling for reading/writing error, ensuring smooth program flow.

Complete Source Code of the Project

The following section includes the full implementation of the Hospital Management System. The project is divided into multiple classes, each responsible for specific functionalities, following Object-Oriented Programming (OOP) principles.

Class 1: Login (*Handles user authentication*)

```
import java.util.Scanner;
public class Login { private static final String CORRECT_USERNAME =
"HospitalManagement"; private static final String CORRECT_PASSWORD = "12345";
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   boolean isAuthenticated = false;

   System.out.println("===== Hospital Management System Login =====");

   while (!isAuthenticated) {
      System.out.print("Enter Username: ");
      String username = scanner.nextLine().trim();

      System.out.print("Enter Password: ");
      String password = scanner.nextLine().trim();

      if (username.equals(CORRECT_USERNAME) &&
password.equals(CORRECT_PASSWORD)) {
         System.out.println("\n Login Successful! Redirecting to Main Menu...\n");
         isAuthenticated = true;
         MainMenu.display();
      } else {
         System.out.println("\n Incorrect Username or Password. Try again.\n");
      }
   }

   scanner.close(); } }
```

Class 2: Main Menu (*Central Navigation Hub*)

```
import java.util.Scanner;
public class MainMenu {
    public static void display() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\n===== Hospital Management System =====");
            System.out.println("1. Add New Patient");
            System.out.println("2. View Patient Info");
            System.out.println("3. Departments");
            System.out.println("4. All Employee Info");
            System.out.println("5. Room Information");
            System.out.println("6. Discharge Patients");
            System.out.println("7. Update Patient Billing Details");
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");

            int choice;
            try {
                choice = Integer.parseInt(scanner.nextLine().trim());
            } catch (NumberFormatException e) {
                System.out.println(" Invalid input! Please enter a number.");
                continue;
            }
            switch (choice) {
                case 1:
                    AddPatient.addPatient(scanner);
                    break;
                case 2:
                    ViewPatients viewPatients = new ViewPatients();
                    viewPatients.displayData();
                    break;
                case 3:
                    Department department = new Department();
                    department.displayData();
                    break;
                case 4:
                    AllEmployeeInfo allEmployees = new AllEmployeeInfo();
                    allEmployees.displayData();
                    break;
                case 5:
                    Room room = new Room();
                    room.displayData();
                    break;
                case 6:
                    DischargePatient.dischargePatient(scanner);
                    break;
                case 7:
                    BillingDetails.updateBillingDetails(scanner);
                    break;
                case 8:
                    System.out.println(" Exiting the system. Thank You");
                    scanner.close();
                    return;
                default:
                    System.out.println(" Invalid choice! Please try again.");
            }
        }
    }
}
```

Class 3: AddPatient (*Manages adding new patient records*)

```
import java.io.*;
import java.util.Scanner;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class AddPatient {
 private static final String PATIENT_FILE = "/Users/ayushgupta/Desktop/Hospital Management
System_Text based /data/patients.txt";
 private static final String ROOM_FILE = "/Users/ayushgupta/Desktop/Hospital Management
System_Text based /data/rooms.txt";
public static void addPatient(Scanner scanner) {
    System.out.println("\n===== Add New Patient =====");
```

```
String[] checkInDateTime = getCurrentDateTime();

String totalBill = "0";
String remainingAmount = "0";

String id;
while (true) {
    System.out.print("Enter Patient ID: ");
    id = scanner.nextLine().trim();
    if (!isPatientIdTaken(id)) {
        break;
    } else {
        System.out.println(" This Patient ID is already taken! Please enter a different ID.");
    }
}

System.out.print("Enter Name: ");
String name = scanner.nextLine().trim();
System.out.print("Enter Age: ");
String age = scanner.nextLine().trim();
System.out.print("Enter Gender (Male/Female): ");
String gender = scanner.nextLine().trim();
System.out.print("Enter Disease: ");
String disease = scanner.nextLine().trim();
System.out.print("Enter Deposit Amount: ");
String deposit = scanner.nextLine().trim();

String roomNumber;
while (true) {
    System.out.print("Enter Room Number: ");
    roomNumber = scanner.nextLine().trim();
    if (isRoomAvailable(roomNumber)) {
        updateRoomStatus(roomNumber, "Occupied");
        break;
    } else {
        System.out.println(" This room is already occupied. Choose another.");
    }
    }

    String patientData = id + "," + name + "," + age + "," + gender + "," + disease + "," + deposit +
"," + roomNumber + "," + checkInDateTime[0] + "," + checkInDateTime[1] + "," + totalBill +
"," + remainingAmount;

    try (FileWriter writer = new FileWriter(PATIENT_FILE, true)) {
        writer.write(patientData + "\n");
        System.out.println(" Patient added successfully!\n");
    } catch (IOException e) {
        System.out.println(" Error saving patient data: " + e.getMessage());
    }
}

private static String[] getCurrentDateTime() {
    LocalDateTime now = LocalDateTime.now();
    DateTimeFormatter formatterDate = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    DateTimeFormatter formatterTime = DateTimeFormatter.ofPattern("HH:mm");

    String[] dateTime = new String[2];
    dateTime[0] = now.format(formatterDate);
    dateTime[1] = now.format(formatterTime);

    return dateTime;
}

private static boolean isPatientIdTaken(String id) {
    try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length == 10 && details[0].trim().equals(id)) {
                return true;
            }
        }
    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
    }
    return false;
}

private static boolean isRoomAvailable(String roomNumber) {
  try (BufferedReader reader = new BufferedReader(new FileReader(ROOM_FILE))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length >= 2 && details[0].trim().equals(roomNumber) &&
details[1].trim().equalsIgnoreCase("Available")) {
                return true;
            }
        }
    } catch (IOException e) {
        System.out.println("Error reading room data: " + e.getMessage());
    }
    return false;
}

private static void updateRoomStatus(String roomNumber, String status) {
    File file = new File(ROOM_FILE);
    File tempFile = new File("temp_rooms.txt");

    try (BufferedReader reader = new BufferedReader(new FileReader(file));
         FileWriter writer = new FileWriter(tempFile)) {

        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length >= 2 && details[0].trim().equals(roomNumber)) {
                writer.write(roomNumber + "," + status + "," + details[2] + "," + details[3] + "\n");
            } else {
                writer.write(line + "\n");
            }
        }
    } catch (IOException e) {
        System.out.println(" Error updating room status: " + e.getMessage());
        return;
    }

    if (file.delete() && tempFile.renameTo(file)) {
        System.out.println(" Room status updated.");
    } else {
        System.out.println(" Error updating room file.");
    }
}
```

Class 4: ViewPatients (*Display Patient Record*)

```
import java.io.*;
public class ViewPatients implements Displayable { private static final String FILE_PATH = "/
Users/ayushgupta/Desktop/Hospital Management System_Text based /data/patients.txt";
public void displayData() {
    File file = new File(FILE_PATH);

    if (!file.exists()) {
        System.out.println(" Error: File does not exist at " + FILE_PATH);
        return;
    }
    if (file.length() == 0) {
        System.out.println(" No patient records found (File is empty).");
        return;
    }

   System.out.println("\n===== Patient Records =====");
    System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
            "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining Amount");

System.out.println("--------------------------------------------------------------------------------------------
-----------------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        boolean hasRecords = false;

        while ((line = reader.readLine()) != null) {
            line = line.trim();
            if (line.isEmpty()) {
                continue;
            }

            String[] details = line.split(",", 11);

            if (details.length == 11) {
                hasRecords = true;
                System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
                        details[0].trim(), details[1].trim(), details[2].trim(),
                        details[3].trim(), details[4].trim(), details[5].trim(), details[6].trim(),
                        details[7].trim(), details[8].trim(), details[9].trim(), details[10].trim());
            } else {
                System.out.println(" Skipping invalid record: " + line);
            }
        }

        if (!hasRecords) {
            System.out.println("\n No valid patient records found.");
        }

    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    ViewPatients viewPatients = new ViewPatients();
    viewPatients.displayData();
}

}
```

Class 5: Displayable (*Interface for uniform data display*)

```
public interface Displayable {
   void displayData();
}
```

Class 6: Department (*Display Hospital Departments*)

```
import java.io.*;
public class Department implements Displayable { private static final String FILE_PATH = "/
Users/ayushgupta/Desktop/Hospital Management System_Text based /data/departments.txt";
public void displayData() {
   File file = new File(FILE_PATH);

   if (!file.exists()) {
      System.out.println(" Error: Departments file does not exist at " + FILE_PATH);
      return;
   }

   if (file.length() == 0) {
      System.out.println(" No department records found (File is empty).");
      return }

   System.out.println("\n===== Hospital Departments =====");
   System.out.printf("%-15s %-25s %-20s%n", "Dept ID", "Department Name", "Head of
Department");
   System.out.println("------------------------------------------------------------------");
 try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
     String line;
      boolean hasRecords = false;

      while ((line = reader.readLine()) != null) {
         String[] details = line.split(",");
         if (details.length == 3) {
            hasRecords = true;
            System.out.printf("%-15s %-25s %-20s%n", details[0], details[1], details[2]);
         } else {
            System.out.println(" Skipping invalid record: " + line);
            }
        }

        if (!hasRecords) {
            System.out.println("\n No valid department records found.");
        }

    } catch (IOException e) {
        System.out.println(" Error reading department data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    Department department = new Department();
    department.displayData();
}
 }
```

Class 7: Employee (*Doctor, Nurse, AdminStaff)*

```
public class Employee {
    protected String empId, name, deptId, email, specialization, shift, contact;
    protected double salary;
    public Employee(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
        this.empId = empId;
        this.name = name;
        this.deptId = deptId;
        this.salary = salary;
        this.email = email;
        this.specialization = specialization;
        this.shift = shift;
        this.contact = contact;
    }

    public void displayInfo() {
        System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s %-15s%n",
                empId, name, "General Employee", deptId, salary, email, specialization, shift, contact);
    }
}

class Doctor extends Employee {
   public Doctor(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }
   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s %-15s%n",
               empId, name, "Doctor", deptId, salary, email, specialization, shift, contact);
   }
}
class Nurse extends Employee {
   public Nurse(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }
   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s %-15s%n",
               empId, name, "Nurse", deptId, salary, email, specialization, shift, contact);
   }
}

class AdminStaff extends Employee {
   public AdminStaff(String empId, String name, String deptId, double salary, String email,
String specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }
  public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s %-15s%n",
               empId, name, "Administrator", deptId, salary, email, specialization, shift, contact);
   }
}
```

Class 8: AllEmployeeInfo (Display All Employee Info Using Polymorphism)

```
import java.io.*;
public class AllEmployeeInfo implements Displayable { private static final String FILE_PATH =
"/Users/ayushgupta/Desktop/Hospital Management System_Text based /data/employees.txt";
private Employee[] employees; private int employeeCount = 0;
public AllEmployeeInfo() {
   employees = new Employee[50];
   loadEmployeesFromFile();
}

public void displayData() {
   if (employeeCount == 0) {
      System.out.println(" No employee records found.");
      return;
   }

   System.out.println("\n===== Employee Records =====");
   System.out.printf("%-10s %-25s %-15s %-10s %-10s %-20s %-15s %-15s %-15s%n",
         "Emp ID", "Name", "Designation", "Dept ID", "Salary", "Email", "Specialization",
"Shift", "Contact");

System.out.println("----------------------------------------------------------------------------------------
----------------------------------");

   for (int i = 0; i < employeeCount; i++) {
      employees[i].displayInfo();
   }
}

private void loadEmployeesFromFile() {
   File file = new File(FILE_PATH);

   if (!file.exists() || file.length() == 0) {
      System.out.println(" No employees found (File missing or empty).");
      return;
   }
```

```
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;

        while ((line = reader.readLine()) != null && employeeCount < employees.length) {
            String[] details = line.split(",");
            if (details.length == 9) {
                String empId = details[0].trim();
                String name = details[1].trim();
                String designation = details[2].trim();
                String deptId = details[3].trim();
                double salary = Double.parseDouble(details[4].trim());
                String email = details[5].trim();
                String specialization = details[6].trim();
                String shift = details[7].trim();
                String contact = details[8].trim();

                if (designation.equalsIgnoreCase("Doctor")) {
                    employees[employeeCount++] = new Doctor(empId, name, deptId, salary, email,
specialization, shift, contact);
                } else if (designation.equalsIgnoreCase("Nurse")) {
                    employees[employeeCount++] = new Nurse(empId, name, deptId, salary, email,
specialization, shift, contact);
                } else {
                    employees[employeeCount++] = new AdminStaff(empId, name, deptId, salary,
email, specialization, shift, contact);
                }
            }
        }
    } catch (IOException | NumberFormatException e) {
        System.out.println(" Error loading employee data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    AllEmployeeInfo allEmployees = new AllEmployeeInfo();
    allEmployees.displayData(); }
}
```

Class 9: Room (D*isplaying Room Details*)

```
import java.io.*;
public class Room implements Displayable { private static final String ROOM_FILE = "/Users/
ayushgupta/Desktop/Hospital Management System_Text based /data/rooms.txt";

public void displayData() {
   File file = new File(ROOM_FILE);

   if (!file.exists() || file.length() == 0) {
      System.out.println(" No room records found.");
       return;
   }

   System.out.println("\n===== Room Information =====");
   System.out.printf("%-10s %-15s %-10s %-20s%n", "Room No", "Availability", "Price",
"Room Type");
   System.out.println("------------------------------------------------------");

   try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
       String line;

       while ((line = reader.readLine()) != null) {
           String[] details = line.split(",");

           if (details.length == 4) {
               String roomNo = details[0].trim();
               String availability = details[1].trim();
               String price = details[2].trim();
               String roomType = details[3].trim();

               if (availability.equalsIgnoreCase("Occupied")) {
                   System.out.printf("%-10s %-15s %-10s %-20s  Room is Occupied!%n",
                           roomNo, availability, price, roomType);
               } else {
                   System.out.printf("%-10s %-15s %-10s %-20s%n",
                           roomNo, availability, price, roomType);
               }
```

```
            }
        }
    } catch (IOException e) {
        System.out.println(" Error reading room data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    Room room = new Room();
    room.displayData();
}

}
```

Class 10: DischargePatients (Discharging Patients)

```
import java.io.*;
import java.util.Scanner;
public class DischargePatient { private static final String PATIENT_FILE = "/Users/ayushgupta/
Desktop/Hospital Management System_Text based /data/patients.txt"; private static final String
ROOM_FILE = "/Users/ayushgupta/Desktop/Hospital Management System_Text based /data/
rooms.txt";
public static void dischargePatient(Scanner scanner) {
    String[] patients = new String[100];
    int patientCount = 0;
    String dischargedPatientRoom = null;

   System.out.println("\n===== Patient List =====");
    System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s
%-12s%-12s%n",
            "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining");

System.out.println("------------------------------------------------------------------------------------------
-----------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
        String line;
        while ((line = reader.readLine()) != null && patientCount < patients.length) {
            line = line.trim();
            if (line.isEmpty()) continue;

            String[] details = line.split(",", -1);
            if (details.length == 11) {
                System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
                    details[0].trim(), details[1].trim(), details[2].trim(), details[3].trim(),
                    details[4].trim(), details[5].trim(), details[6].trim(),
                    details[7].trim(), details[8].trim(), details[9].trim(), details[10].trim());
                patients[patientCount++] = line;
            } else {
                System.out.println(" Skipping invalid record: " + line);
            }
        }

        if (patientCount == 0) {
            System.out.println(" No patients found.");
            return;
        }

    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
        return;
    }

    System.out.print("\nEnter Patient ID to discharge: ");
    String patientId = scanner.nextLine().trim();
    boolean found = false;
    String[] updatedPatients = new String[patientCount];
    int updatedCount = 0;

    for (int i = 0; i < patientCount; i++) {
        String[] details = patients[i].split(",");
        if (details.length == 11 && details[0].trim().equals(patientId)) {
            dischargedPatientRoom = details[6].trim();
```

```
                found = true;
            } else {
                updatedPatients[updatedCount++] = patients[i];
            }
        }

        if (!found) {
            System.out.println(" No patient found with ID: " + patientId);
            return;
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(PATIENT_FILE))) {
            for (int i = 0; i < updatedCount; i++) {
                writer.write(updatedPatients[i]);
                writer.newLine();
            }
        } catch (IOException e) {
            System.out.println(" Error updating patient records: " + e.getMessage());
        }

        if (dischargedPatientRoom != null) {
            updateRoomStatus(dischargedPatientRoom);
        }

        System.out.println(" Patient with ID " + patientId + " has been discharged successfully!");
    }

    private static void updateRoomStatus(String roomNumber) {
        String[] rooms = new String[100];
        int roomCount = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(ROOM_FILE))) {
            String line;
            while ((line = reader.readLine()) != null && roomCount < rooms.length) {
                String[] details = line.split(",");
                if (details.length == 4 && details[0].trim().equals(roomNumber)) {
                    details[1] = "Available";
                    line = String.join(",", details);
                }
                rooms[roomCount++] = line;
            }
        } catch (IOException e) {
            System.out.println(" Error reading room data: " + e.getMessage());
            return;
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ROOM_FILE))) {
            for (int i = 0; i < roomCount; i++) {
                writer.write(rooms[i]);
                writer.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error updating room records: " + e.getMessage());
        }
    }


    }
```

Class 11: BillingDetails (*Update the Billing details of the patients*)

```
import java.io.*; import java.util.Scanner;
public class BillingDetails { private static final String PATIENT_FILE = "/Users/ayushgupta/
Desktop/Hospital Management System_Text based /data/patients.txt";
public static void updateBillingDetails(Scanner scanner) {
    String[] patients = new String[100];
    int patientCount = 0;

   System.out.println("\n===== Patient Billing Details =====");
    System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-15s %-15s
%-15s%n",
            "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining Amount");

System.out.println("-----------------------------------------------------------------------------------------
---------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
        String line;
        while ((line = reader.readLine()) != null && patientCount < patients.length) {
            String[] details = line.split(",", -1);
            if (details.length == 11) {
                System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-15s %-15s
%-15s%n",
                        details[0], details[1], details[2], details[3], details[4], details[5], details[6],
                        details[7], details[8], details[9], details[10]);
                patients[patientCount++] = line;
            } else {
                System.out.println("invalid record: " + line);
            }
        }
    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
        return;
    }

    System.out.print("\nEnter Patient ID to update billing details: ");
    String patientId = scanner.nextLine().trim();
    boolean patientFound = false;

    for (int i = 0; i < patientCount; i++) {
        String[] details = patients[i].split(",");

        if (details[0].equals(patientId)) {
            patientFound = true;
            System.out.println("\n Updating billing details for: " + details[1]);

            double depositAmount = Double.parseDouble(details[5]);
            double totalBill = Double.parseDouble(details[9]);
            double remainingAmount = Double.parseDouble(details[10]);

            while (true) {
```

```
            System.out.println("\nChoose an option:");
            System.out.println("1. Update Total Bill");
            System.out.println("2. Pay Remaining Amount");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            String choice = scanner.nextLine().trim();

            if (choice.equals("1")) {
                System.out.print("Enter new Total Bill Amount: ");
                double newTotalBill = Double.parseDouble(scanner.nextLine().trim());

                if (totalBill == 0) {
                    remainingAmount = newTotalBill - depositAmount;
                } else {
                    remainingAmount += (newTotalBill - totalBill);
                }

                totalBill = newTotalBill;
                System.out.println(" Updated Remaining Amount: " + remainingAmount);

            } else if (choice.equals("2")) {
                System.out.print("Enter amount to pay: ");
                double payment = Double.parseDouble(scanner.nextLine().trim());

                if (payment > remainingAmount) {
                    System.out.println(" Payment exceeds remaining amount! Enter a valid
amount.");
                    continue;
                }

                remainingAmount -= payment;
                System.out.println(" Payment successful! New Remaining Amount: " +
remainingAmount);

            } else if (choice.equals("3")) {
                break;
            } else {
                System.out.println(" Invalid choice. Please enter 1, 2, or 3.");
                continue;
            }

            details[9] = String.valueOf(totalBill);
            details[10] = String.valueOf(remainingAmount);
            patients[i] = String.join(",", details);
        }
        break;
    }
}

if (!patientFound) {
    System.out.println(" No patient found with ID: " + patientId);
    return;
}

try (BufferedWriter writer = new BufferedWriter(new FileWriter(PATIENT_FILE))) {
    for (int i = 0; i < patientCount; i++) {
        writer.write(patients[i]);
        writer.newLine();
    }
    System.out.println(" Billing details updated successfully!");
} catch (IOException e) {
    System.out.println("Error updating patient data: " + e.getMessage());
}
}

}
```

Code Snippets with Explanation and Sample Output

1. Login Module

Code:

```
import java.util.Scanner;
public class Login { private static final String CORRECT_USERNAME =
"HospitalManagement"; private static final String CORRECT_PASSWORD = "12345";
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   boolean isAuthenticated = false;

   System.out.println("===== Hospital Management System Login =====");

   while (!isAuthenticated) {
      System.out.print("Enter Username: ");
      String username = scanner.nextLine().trim();

      System.out.print("Enter Password: ");
      String password = scanner.nextLine().trim();

      if (username.equals(CORRECT_USERNAME) &&
password.equals(CORRECT_PASSWORD)) {
         System.out.println("\n Login Successful! Redirecting to Main Menu...\n");
         isAuthenticated = true;
         MainMenu.display();
      } else {
         System.out.println("\n Incorrect Username or Password. Try again.\n");
      }
   }

   scanner.close();
}

}
```

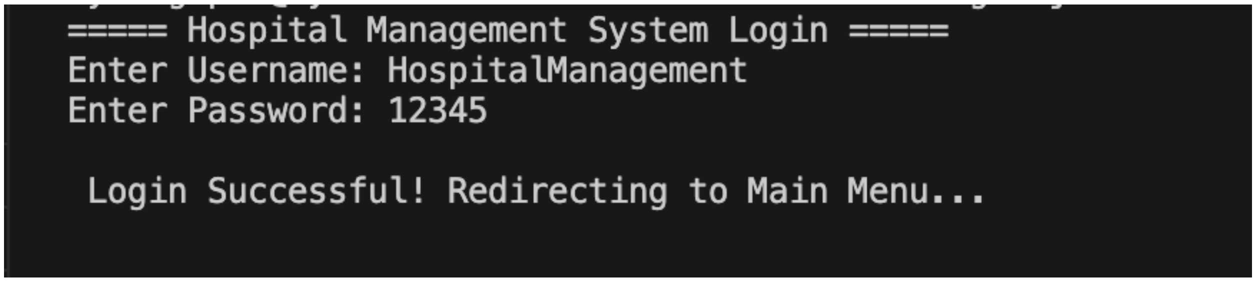
Concepts Used:

- Scanner
- String Comparision
- while loop for repeated login attempts
- if-else conditional logic
- Method calling (MainMenu.display())

Explanation:

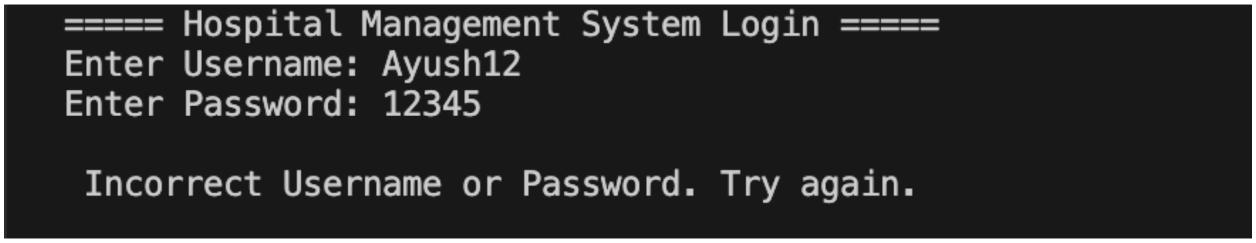
This code performs user authentication for the Hospital Management System. It sets the correct username and password through constants and then employs the Scanner class to accept input from the user. A while loop is employed to repeatedly ask the user for their credentials until they provide the correct ones. The program verifies the input through the equals() method for exact string matches. If the credentials are valid, it shows a success message and invokes the MainMenu.display() method to take the user to the main menu. If the login is unsuccessful, it shows an error message and continues the process. Once the authentication is complete, the scanner is closed to release system resources.

Output: When Login Successfull:



```
===== Hospital Management System Login =====
Enter Username: HospitalManagement
Enter Password: 12345

 Login Successful! Redirecting to Main Menu...
```

Output: When Login Fails:



```
===== Hospital Management System Login =====
Enter Username: Ayush12
Enter Password: 12345

 Incorrect Username or Password. Try again.
```

2. Main Menu Module

Code:

```
import java.util.Scanner;

public class MainMenu {
    public static void display() {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n===== Hospital Management System =====");
            System.out.println("1. Add New Patient");
            System.out.println("2. View Patient Info");
            System.out.println("3. Departments");
            System.out.println("4. All Employee Info");
            System.out.println("5. Room Information");
            System.out.println("6. Discharge Patients");
            System.out.println("7. Update Patient Billing Details");
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");

            int choice;
            try {
                choice = Integer.parseInt(scanner.nextLine().trim());
            } catch (NumberFormatException e) {
                System.out.println(" Invalid input! Please enter a number.");
                continue;
            }

            switch (choice) {
                case 1:
                    AddPatient.addPatient(scanner);
                    break;
                case 2:
                    ViewPatients viewPatients = new ViewPatients();
                    viewPatients.displayData();
                    break;
                case 3:
                    Department department = new Department();
                    department.displayData();
                    break;
                case 4:
                    AllEmployeeInfo allEmployees = new AllEmployeeInfo();
                    allEmployees.displayData();
                    break;
                case 5:
                    Room room = new Room();
                    room.displayData();
                    break;
                case 6:
                    DischargePatient.dischargePatient(scanner);
                    break;
                case 7:
                    BillingDetails.updateBillingDetails(scanner);
                    break;
                case 8:
                    System.out.println(" Exiting the system. Thank You");
                    scanner.close();
                    return;
                default:
                    System.out.println(" Invalid choice! Please try again.");
            }
        }
    }
}
```

Concepts Used:

- Scanner class for input
- while loop for continuous display
- switch-case for menu handling
- try-catch block for input validation
- Constructor
- Method calling and object creation

Explanation:

This is the code for the main menu of the Hospital Management System. It starts with declaring a Scanner object to receive user input and then repeatedly displays a list of choices through a while(true) loop. The loop keeps the user in the menu until he chooses to come out of it.

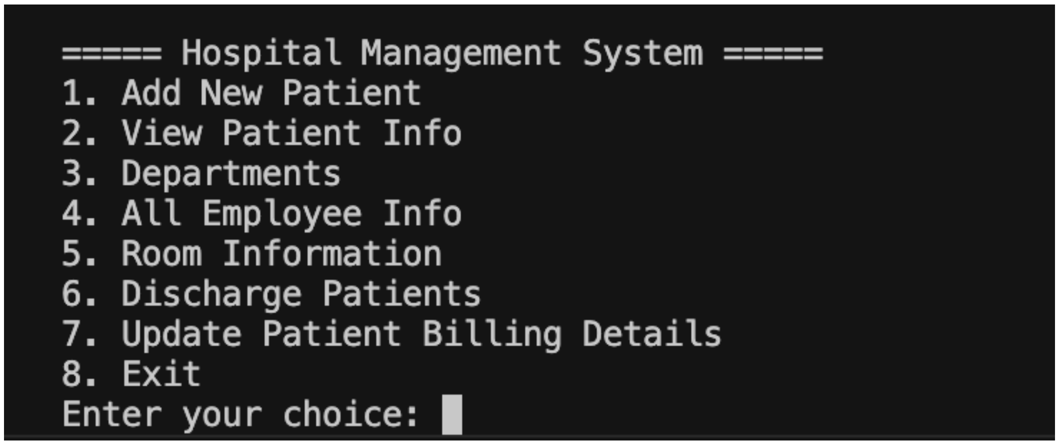
To deal with unexpected input (like entering a letter when a number is expected), a try-catch block is employed. This tests whether the input is an integer. If not, the program catches the NumberFormatException and lets the user know with a nice error message, then asks them again without crashing.

The essential functionality is provided by using a switch-case approach. Based on the numeric input of the user, a different case is invoked. Certain cases invoke static methods directly (such as AddPatient.addPatient(scanner)), whereas others instantiate new objects and then invoke their respective displayData() methods. This approach encourages modularity and neat organization.

Every menu choice relates to a unique function within the system—new patient additions, viewing patient or departmental data, managing employee or room data, discharging patients, or updating billing information. If the user selects option 8, the system prompts a thank-you message, shuts down the Scanner, and breaks out of the menu loop normally.

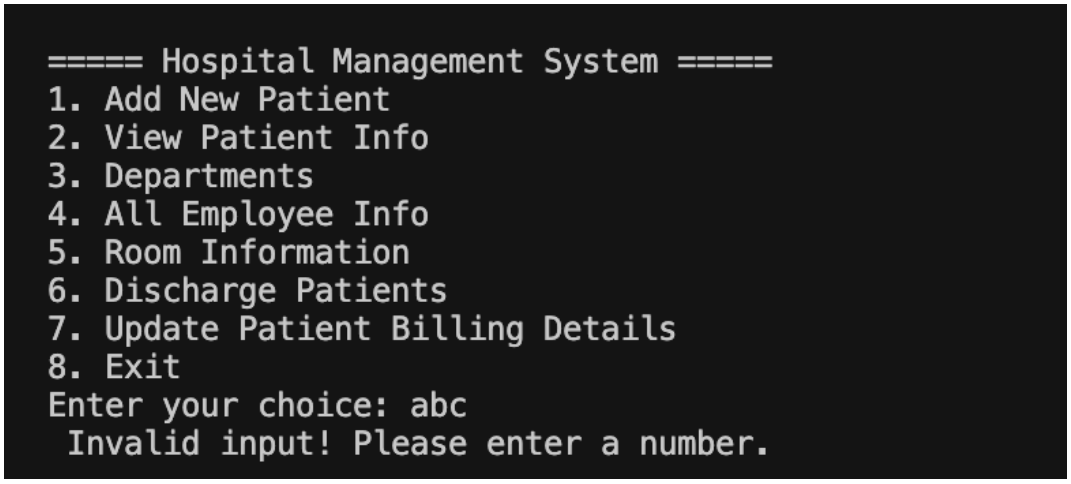
Finally, any other input that does not fall within the valid range (1–8) will invoke the default case, which merely informs the user that their input is not valid and loops back to show the menu again.

Output:



```
===== Hospital Management System =====
1. Add New Patient
2. View Patient Info
3. Departments
4. All Employee Info
5. Room Information
6. Discharge Patients
7. Update Patient Billing Details
8. Exit
Enter your choice: 
```

Output : Invalid Input:



```
===== Hospital Management System =====
1. Add New Patient
2. View Patient Info
3. Departments
4. All Employee Info
5. Room Information
6. Discharge Patients
7. Update Patient Billing Details
8. Exit
Enter your choice: abc
 Invalid input! Please enter a number.
```

3. Add Patient Module

Code:

```
import java.io.*; import java.util.Scanner; import java.time.LocalDateTime; import
java.time.format.DateTimeFormatter;
public class AddPatient { private static final String PATIENT_FILE = "/Users/ayushgupta/
Desktop/Hospital Management System_Text based /data/patients.txt";
 private static final String ROOM_FILE = "/Users/ayushgupta/Desktop/Hospital Management
System_Text based /data/rooms.txt";
public static void addPatient(Scanner scanner) {
    System.out.println("\n===== Add New Patient =====");

    String[] checkInDateTime = getCurrentDateTime();

    String totalBill = "0";
    String remainingAmount = "0";

    String id;
    while (true) {
        System.out.print("Enter Patient ID(Aadhar): ");
        id = scanner.nextLine().trim();
        if (!isPatientIdTaken(id)) {
            break;
        } else {
            System.out.println(" This Patient ID is already taken! Please enter a different ID.");
        }
    }

    System.out.print("Enter Name: ");
    String name = scanner.nextLine().trim();
    System.out.print("Enter Age: ");
    String age = scanner.nextLine().trim();
    System.out.print("Enter Gender (Male/Female): ");
    String gender = scanner.nextLine().trim();
    System.out.print("Enter Disease: ");
    String disease = scanner.nextLine().trim();
    System.out.print("Enter Deposit Amount: ");
    String deposit = scanner.nextLine().trim();

    String roomNumber;
    while (true) {
        System.out.print("Enter Room Number: ");
        roomNumber = scanner.nextLine().trim();
        if (isRoomAvailable(roomNumber)) {
            updateRoomStatus(roomNumber, "Occupied");
            break;
        } else {
            System.out.println(" This room is already occupied. Choose another.");
        }
    }

    String patientData = id + "," + name + "," + age + "," + gender + "," + disease + "," + deposit +
"," + roomNumber + "," + checkInDateTime[0] + "," + checkInDateTime[1] + "," + totalBill +
"," + remainingAmount;

    try (FileWriter writer = new FileWriter(PATIENT_FILE, true)) {
        writer.write(patientData + "\n");
        System.out.println(" Patient added successfully!\n");
    } catch (IOException e) {
        System.out.println(" Error saving patient data: " + e.getMessage());
    }
```

```
}

private static String[] getCurrentDateTime() {
    LocalDateTime now = LocalDateTime.now();
    DateTimeFormatter formatterDate = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    DateTimeFormatter formatterTime = DateTimeFormatter.ofPattern("HH:mm");

    String[] dateTime = new String[2];
    dateTime[0] = now.format(formatterDate);
    dateTime[1] = now.format(formatterTime);

    return dateTime;
}

private static boolean isPatientIdTaken(String id) {
    try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length == 10 && details[0].trim().equals(id)) {
                return true;
            }
        }
    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
    }
    return false;
}

private static boolean isRoomAvailable(String roomNumber) {
    try (BufferedReader reader = new BufferedReader(new FileReader(ROOM_FILE))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length >= 2 && details[0].trim().equals(roomNumber) &&
details[1].trim().equalsIgnoreCase("Available")) {
                return true;
            }
        }
    } catch (IOException e) {
        System.out.println("Error reading room data: " + e.getMessage());
    }
    return false;
}

private static void updateRoomStatus(String roomNumber, String status) {
    File file = new File(ROOM_FILE);
    File tempFile = new File("temp_rooms.txt");

    try (BufferedReader reader = new BufferedReader(new FileReader(file));
         FileWriter writer = new FileWriter(tempFile)) {

        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length >= 2 && details[0].trim().equals(roomNumber)) {
                writer.write(roomNumber + "," + status + "," + details[2] + "," + details[3] + "\n");
            } else {
                writer.write(line + "\n");
            }
        }
    } catch (IOException e) {
        System.out.println(" Error updating room status: " + e.getMessage());
        return;
    }

    if (file.delete() && tempFile.renameTo(file)) {
        System.out.println(" Room status updated.");
    } else {
        System.out.println(" Error updating room file.");
    }
}

}
```

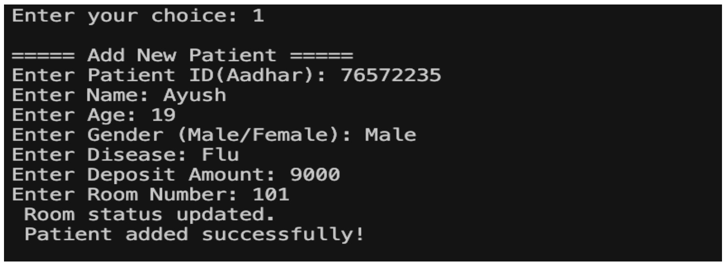
Concepts Used:

- File Handling
- Scanner
- String Manipulation and formatting
- Date and time Handling
- Data storage in structured files (.txt)
- Modular Methods (helper functions like isRoomAvailable, updateRoomStatus, etc.)
- Exception Handling
- Constructor

Explanation:

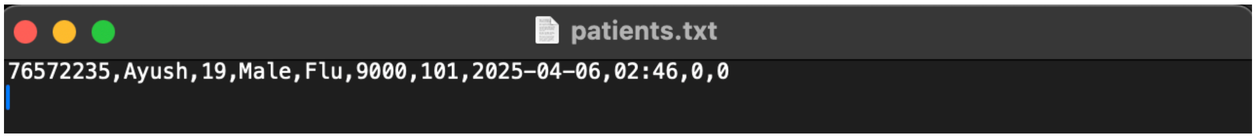
This module manages the process of registering a new patient to the hospital system. It begins by creating the present date and time through LocalDateTime and DateTimeFormatter, which are utilized in capturing the patient's check-in information. The system utilizes a Scanner to capture patients' data including ID, name, age, gender, disease, deposit amount, and preferred room number.
Prior to the acceptance of a patient ID, the system will check if the ID already exists through the isPatientIdTaken() method so that no duplicate IDs are submitted. After being validated, it proceeds to ask for other patient information. When it comes to room selection, the system will check availability through the isRoomAvailable() method. If the chosen room is available, it is flagged as "Occupied" through the updateRoomStatus() method.
Once all the inputs are fetched and verified, the data is formatted as a line of comma-separated values and saved to a text file in FileWriter. This record contains bill placeholders such as total bill and amount remaining, which are assigned an initial value of zero. In case there is any sort of error when writing to the file, this is caught and output using a try-catch block.
The whole logic is structured with modular helper methods, which makes the code readable and maintainable. These are room availability methods, room status update methods, duplicate ID methods, and date-time methods.

Output:



```
Enter your choice: 1

===== Add New Patient =====
Enter Patient ID(Aadhar): 76572235
Enter Name: Ayush
Enter Age: 19
Enter Gender (Male/Female): Male
Enter Disease: Flu
Enter Deposit Amount: 9000
Enter Room Number: 101
 Room status updated.
 Patient added successfully!
```

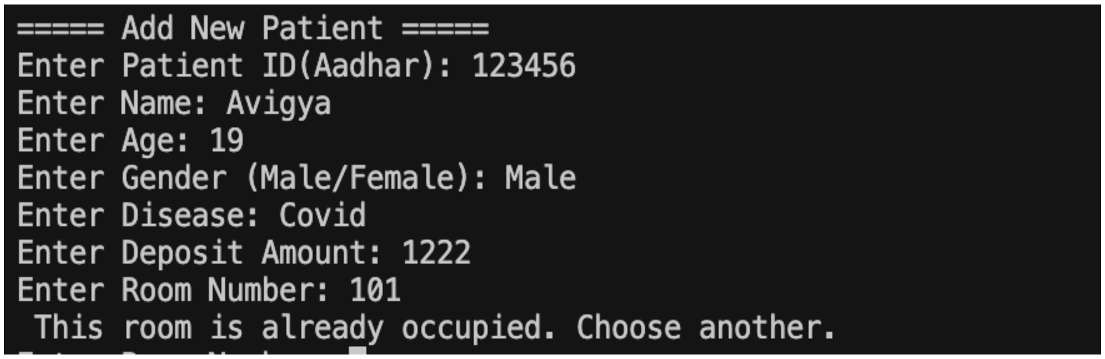
Data Stored in Patient.txt :



```
76572235,Ayush,19,Male,Flu,9000,101,2025-04-06,02:46,0,0
```

Output:

- If the room is already occupied



```
===== Add New Patient =====
Enter Patient ID(Aadhar): 123456
Enter Name: Avigya
Enter Age: 19
Enter Gender (Male/Female): Male
Enter Disease: Covid
Enter Deposit Amount: 1222
Enter Room Number: 101
 This room is already occupied. Choose another.
```

4. Departments Module

Code:

```
import java.io.*;
public class Department implements Displayable { private static final String FILE_PATH = "/
Users/ayushgupta/Desktop/Hospital Management System_Text based /data/departments.txt";
public void displayData() {
   File file = new File(FILE_PATH);

   if (!file.exists()) {
      System.out.println(" Error: Departments file does not exist at " + FILE_PATH);
      return;
   }

   if (file.length() == 0) {
      System.out.println(" No department records found (File is empty).");
      return;
   }

   System.out.println("\n===== Hospital Departments =====");
```

```
    System.out.printf("%-15s %-25s %-20s%n", "Dept ID", "Department Name", "Head of
Department");
    System.out.println("-----------------------------------------------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        boolean hasRecords = false;

        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length == 3) {
                hasRecords = true;
                System.out.printf("%-15s %-25s %-20s%n", details[0], details[1], details[2]);
            } else {
                System.out.println(" Skipping invalid record: " + line);
            }
        }

        if (!hasRecords) {
            System.out.println("\n No valid department records found.");
        }

    } catch (IOException e) {
        System.out.println(" Error reading department data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    Department department = new Department();
    department.displayData();
}

}
```

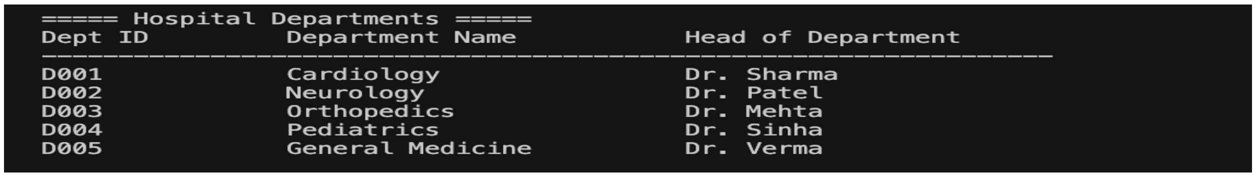
Concepts Used:

- File Handling
- Strings
- Interface Implementation (Displayable)
- Polymorphism via Method Overriding
- Modular Design (encapsulating logic inside displayData() method)
- Exception Handling
- Constructor

Explanation:

The Department Module has the functionality of displaying the list of hospital departments by reading data from a text file. It provides the functionality to the users to view department-related information such as Department ID, Department Name, and Head of Department in a well-formatted way.
The program initially checks if the specified file exists or not. If not, it displays an error message to inform the user that the department file is not found. This avoids any file-not-found errors while running. It also checks whether the file is empty or not to prevent blank data from being displayed.
It then applies a BufferedReader to read the file line by line. It breaks each line, using commas as delimiters for values, separating them into columns. The form expected is three pieces of information: the ID of the department, name, and head. If the line has exactly three of these pieces of information, they are then printed out tabularly in System.out.printf().
If an incorrect number of elements is present in a line of the file, the program safely skips over it and alerts the user by printing a message. This introduces robustness into the system as it gracefully processes corrupt or faulty records.
The Displayable interface is implemented by the Department class, and the displayData() method is a case of method overriding. This is an example of polymorphism, in which the same method signature behaves differently based on the implementing class. This is a design choice that enables scalable architecture, where additional modules can be designed with the same pattern.

Output:



```
===== Hospital Departments =====
Dept ID         Department Name          Head of Department
------------------------------------------------------------------
D001            Cardiology               Dr. Sharma
D002            Neurology                Dr. Patel
D003            Orthopedics              Dr. Mehta
D004            Pediatrics               Dr. Sinha
D005            General Medicine         Dr. Verma
```

5. Employee Module

Code

```
public class Employee {
   protected String empId, name, deptId, email, specialization, shift, contact;
   protected double salary;

   public Employee(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
       this.empId = empId;
       this.name = name;
       this.deptId = deptId;
       this.salary = salary;
        this.email = email;
       this.specialization = specialization;
       this.shift = shift;
       this.contact = contact;
   }

   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s%n",
               empId, name, "General Employee", deptId, salary, email, specialization, shift, contact);
   }
}

// Doctor subclass
class Doctor extends Employee {
   public Doctor(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }
   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s%n",
               empId, name, "Doctor", deptId, salary, email, specialization, shift, contact);
   }
}
```

```
class Nurse extends Employee {
   public Nurse(String empId, String name, String deptId, double salary, String email, String
specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }

   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s%n",
               empId, name, "Nurse", deptId, salary, email, specialization, shift, contact);
   }
}
 class AdminStaff extends Employee {
   public AdminStaff(String empId, String name, String deptId, double salary, String email,
String specialization, String shift, String contact) {
       super(empId, name, deptId, salary, email, specialization, shift, contact);
   }

   public void displayInfo() {
       System.out.printf("%-10s %-25s %-15s %-10s %-10.2f %-20s %-15s %-15s%n",
               empId, name, "Administrator", deptId, salary, email, specialization, shift, contact);
   }
}
```

Concepts Used

- Class and Object-Oriented Design
- This keyword
- Super keyword
- Inheritance (subclasses: Doctor, Nurse, AdminStaff extending Employee)
- Polymorphism (method overriding using displayInfo() in each subclass)
- Constructor Chaining using super()
- Method Overriding to customize employee role display
- Protected Access Modifier for shared access within package/subclasses
- Formatted Output using System.out.printf()
- Exception Handling

6. **Employee Management Module**

Code:

```
import java.io.*;
 Displayable { private static final String FILE_PATH = "/Users/ayushgupta/Desktop/Hospital
Management System_Text based /data/employees.txt"; private Employee[] employees; // Array
to store employees private int employeeCount = 0; // To keep track of employees added
public AllEmployeeInfo() {
   employees = new Employee[50]; // Assuming max 50 employees
   loadEmployeesFromFile();
}

public void displayData() {
   if (employeeCount == 0) {
      System.out.println(" No employee records found.");
      return;
   }

   System.out.println("\n===== Employee Records =====");
   System.out.printf("%-10s %-25s %-15s %-10s %-10s %-20s %-15s %-15s %-15s%n",
         "Emp ID", "Name", "Designation", "Dept ID", "Salary", "Email", "Specialization",
"Shift", "Contact");

System.out.println("-------------------------------------------------------------------------------------
----------------------------------");

   for (int i = 0; i < employeeCount; i++) {
      employees[i].displayInfo();
   }
}

private void loadEmployeesFromFile() {
   File file = new File(FILE_PATH);

   if (!file.exists() || file.length() == 0) {
      System.out.println(" No employees found (File missing or empty).");
      return;
```

```
    }

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;

        while ((line = reader.readLine()) != null && employeeCount < employees.length) {
            String[] details = line.split(",");
            if (details.length == 9) {
                String empId = details[0].trim();
                String name = details[1].trim();
                String designation = details[2].trim();
                String deptId = details[3].trim();
                double salary = Double.parseDouble(details[4].trim());
                String email = details[5].trim();
                String specialization = details[6].trim();
                String shift = details[7].trim();
                String contact = details[8].trim();

                if (designation.equalsIgnoreCase("Doctor")) {
                    employees[employeeCount++] = new Doctor(empId, name, deptId, salary, email,
specialization, shift, contact);
                } else if (designation.equalsIgnoreCase("Nurse")) {
                    employees[employeeCount++] = new Nurse(empId, name, deptId, salary, email,
specialization, shift, contact);
                } else {
                    employees[employeeCount++] = new AdminStaff(empId, name, deptId, salary,
email, specialization, shift, contact);
                }
            }
        }
    } catch (IOException | NumberFormatException e) {
        System.out.println(" Error loading employee data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    AllEmployeeInfo allEmployees = new AllEmployeeInfo();
    allEmployees.displayData();  // Display employee details
}

}
```

Concepts Used:

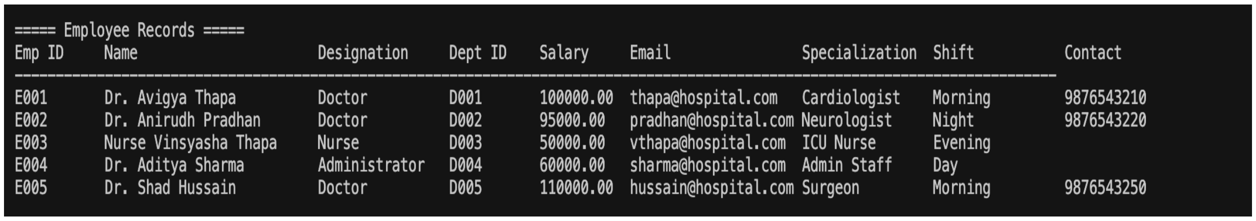
- Classes and Objects
- Constructors
- Inheritance
- Method Overriding
- Interfaces
- Access Modifiers (public, private)
- Arrays
- File Handling using Java I/O Streams (FileReader, BufferedReader)
- Exception Handling (try-catch)
- Program Control Statements (if-else, while, for)
- Data Types and Variables
- Input and Output Statements (System.out.println)

Explanation:

The class AllEmployeeInfo uses a constructor to initialize an array of Employee objects and calls a method to load data from a file. The displayData() method showcases method overriding as it calls the child class's displayInfo() method for each employee. The class implements a custom interface (Displayable), demonstrating how interfaces are used in Java.

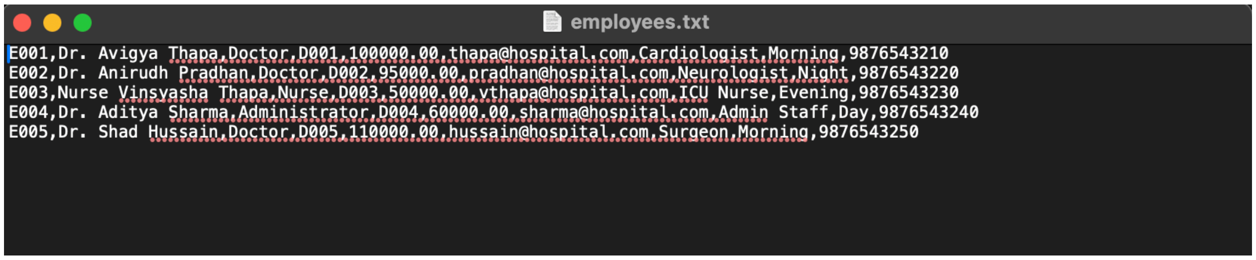
The code also makes use of access modifiers (private, public) to enforce encapsulation. A fixed-size array is used to store employee objects. Java File I/O streams (FileReader, BufferedReader) are used to read data from a text file, and exception handling (try-catch) ensures that errors such as file not found or number format issues are caught gracefully. Additionally, standard control statements like if, else, and while are used for logic flow. The use of data types, variables, and output statements like System.out.println completes the overall implementation of an OOP-based employee information system.

Output:



```
===== Employee Records =====
Emp ID     Name                  Designation    Dept ID    Salary     Email                Specialization  Shift          Contact
-----------------------------------------------------------------------------------------------------------------------------
E001       Dr. Avigya Thapa      Doctor         D001       100000.00  thapa@hospital.com   Cardiologist    Morning        9876543210
E002       Dr. Anirudh Pradhan   Doctor         D002       95000.00   pradhan@hospital.com Neurologist     Night          9876543220
E003       Nurse Vinsyasha Thapa Nurse          D003       50000.00   vthapa@hospital.com  ICU Nurse       Evening
E004       Dr. Aditya Sharma     Administrator  D004       60000.00   sharma@hospital.com  Admin Staff     Day
E005       Dr. Shad Hussain      Doctor         D005       110000.00  hussain@hospital.com Surgeon         Morning        9876543250
```

Records in the employees.txt



```
E001,Dr. Avigya Thapa,Doctor,D001,100000.00,thapa@hospital.com,Cardiologist,Morning,9876543210
E002,Dr. Anirudh Pradhan,Doctor,D002,95000.00,pradhan@hospital.com,Neurologist,Night,9876543220
E003,Nurse Vinsyasha Thapa,Nurse,D003,50000.00,vthapa@hospital.com,ICU Nurse,Evening,9876543230
E004,Dr. Aditya Sharma,Administrator,D004,60000.00,sharma@hospital.com,Admin Staff,Day,9876543240
E005,Dr. Shad Hussain,Doctor,D005,110000.00,hussain@hospital.com,Surgeon,Morning,9876543250
```

7. Discharge Patient module

Code:

```
import java.io.*; import java.util.Scanner;
public class DischargePatient { private static final String PATIENT_FILE = "/Users/ayushgupta/
Desktop/Hospital Management System_Text based /data/patients.txt"; private static final String
ROOM_FILE = "/Users/ayushgupta/Desktop/Hospital Management System_Text based /data/
rooms.txt";
public static void dischargePatient(Scanner scanner) {
    String[] patients = new String[100];
    int patientCount = 0;
    String dischargedPatientRoom = null;
     System.out.println("\n===== Patient List =====");
    System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
            "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining Amount");

System.out.println("--------------------------------------------------------------------------------------------
------------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
        String line;
        while ((line = reader.readLine()) != null && patientCount < patients.length) {
            line = line.trim();
            if (line.isEmpty()) continue;
            String[] details = line.split(",", -1);
            if (details.length == 11) {
                System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
                        details[0].trim(), details[1].trim(), details[2].trim(), details[3].trim(),
                        details[4].trim(), details[5].trim(), details[6].trim(),
                        details[7].trim(), details[8].trim(), details[9].trim(), details[10].trim());
                patients[patientCount++] = line;
            } else {
                System.out.println(" Skipping invalid record (incorrect fields): " + line);
            }
        }
        if (patientCount == 0) {
            System.out.println(" No patients found.");
            return;
        }
    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
        return;
    }

    System.out.print("\nEnter Patient ID to discharge: ");
    String patientId = scanner.nextLine().trim();
    boolean found = false;
    String[] updatedPatients = new String[patientCount];
    int updatedCount = 0;

    for (int i = 0; i < patientCount; i++) {
        String[] details = patients[i].split(",");
        if (details.length == 11 && details[0].trim().equals(patientId)) {
            dischargedPatientRoom = details[6].trim();
            found = true;
        } else {
            updatedPatients[updatedCount++] = patients[i];
        }
    }

    if (!found) {
        System.out.println(" No patient found with ID: " + patientId);
        return;
    }

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(PATIENT_FILE))) {
        for (int i = 0; i < updatedCount; i++) {
            writer.write(updatedPatients[i]);
            writer.newLine();
        }
    } catch (IOException e) {
        System.out.println(" Error updating patient records: " + e.getMessage());
    }

    if (dischargedPatientRoom != null) {
        updateRoomStatus(dischargedPatientRoom);
    }

    System.out.println(" Patient with ID " + patientId + " has been discharged successfully!");
}

private static void updateRoomStatus(String roomNumber) {
    String[] rooms = new String[100];
    int roomCount = 0;

    try (BufferedReader reader = new BufferedReader(new FileReader(ROOM_FILE))) {
        String line;
        while ((line = reader.readLine()) != null && roomCount < rooms.length) {
            String[] details = line.split(",");
            if (details.length == 4) {
                if (details[0].trim().equals(roomNumber)) {
                    details[1] = "Available";
                    line = String.join(",", details);
                }
            }
            rooms[roomCount++] = line;
        }
    } catch (IOException e) {
        System.out.println(" Error reading room data: " + e.getMessage());
        return;
    }

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(ROOM_FILE))) {
        for (int i = 0; i < roomCount; i++) {
            writer.write(rooms[i]);
            writer.newLine();
        }
    } catch (IOException e) {
        System.out.println("Error updating room records: " + e.getMessage());
    }
}

}
```

Concepts Used

- File Handling (FileReader, BufferedReader, FileWriter, BufferedWriter)
- Exception Handling (try-catch)
- Array Operations
- Conditional Statements
- Looping Constructs (for, while)
- String Manipulation (split(), trim(), equals())
- Modular Programming (updateRoomStatus() method)
- Formatted Output (System.out.printf())

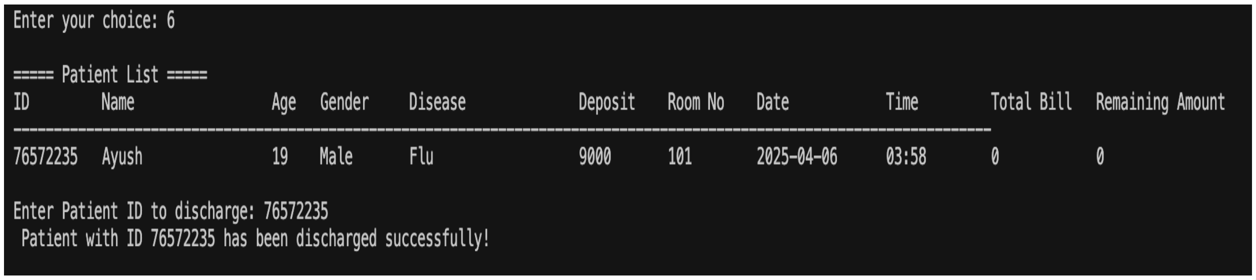
Explanation:

This module is responsible for discharging a patient out of the hospital system. It begins by reading all patient records from a text file and printing them in tabular form using formatted printing. One line from the file corresponds to one patient record containing properties like ID, name, age, gender, disease, deposit, room number, admission date and time, total bill, and remaining amount.

The program requires the user to input the ID of the patient that they need to discharge. When the given ID is present, the individual record of the patient in question is deleted from the data and the file rewritten excluding that one entry. It also fetches the corresponding room number and tags it as "Available" inside the rooms.txt file through updateRoomStatus(). This way, the room may be allocated again to upcoming patients.

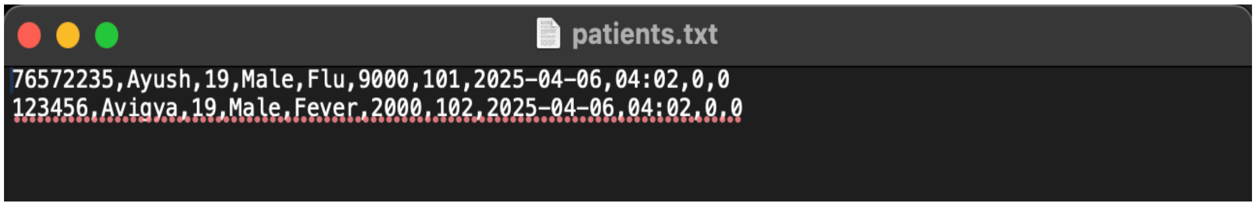
The module employs fundamental Java concepts such as file operations, exception handling, string manipulation, conditional statements, and modular programming to carry out this operation with efficiency and simplicity.

Output:



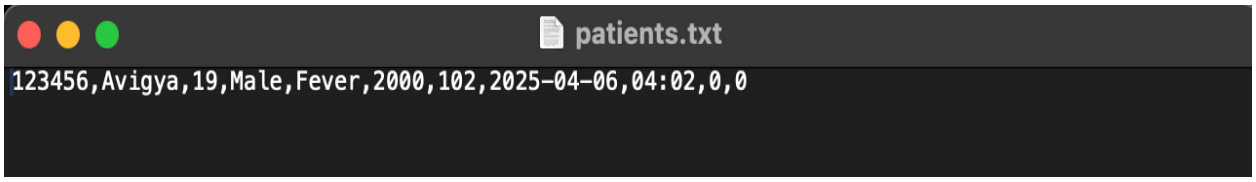
```
Enter your choice: 6

===== Patient List =====
ID          Name                  Age   Gender    Disease               Deposit    Room No    Date            Time        Total Bill   Remaining Amount
--------------------------------------------------------------------------------------------------------------------------------
76572235    Ayush                 19    Male      Flu                   9000       101        2025-04-06      03:58       0            0

Enter Patient ID to discharge: 76572235
 Patient with ID 76572235 has been discharged successfully!
```

Sample Patient.txt(before discharge):



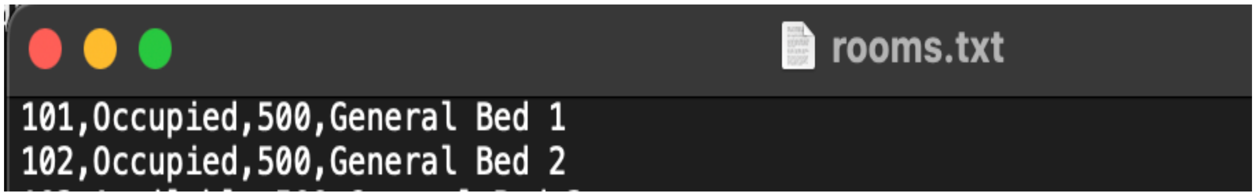
```
76572235,Ayush,19,Male,Flu,9000,101,2025-04-06,04:02,0,0
123456,Avigya,19,Male,Fever,2000,102,2025-04-06,04:02,0,0
```

Sample Patient.txt (after discharge)



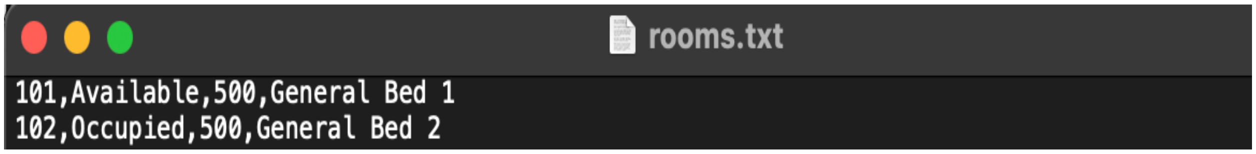
```
123456,Avigya,19,Male,Fever,2000,102,2025-04-06,04:02,0,0
```

Sample Rooms .txt (Before)



```
101,Occupied,500,General Bed 1
102,Occupied,500,General Bed 2
```

Sample Rooms.txt (After)



```
101,Available,500,General Bed 1
102,Occupied,500,General Bed 2
```

8. Displayable Module

Code

```
public interface Displayable {
    void displayData();
}
```

Explanation:

Method overriding is implemented in this class in this module through the implementation of the interface Displayable. The interface has an abstract method called displayData(), which serves as a contract for any class implementing it. The class AllEmployeeInfo implements the interface, and its own version of the displayData() method overrides the method declared in the interface. This shows runtime polymorphism, in which various classes can override the same method in their own manner. Overriding in this case provides uniform behavior while still providing flexibility in the way data is presented for various entities such as employees or patients.

9. Room Module

Code:

```
import java.io.*;
public class Room implements Displayable { private static final String ROOM_FILE = "/Users/
ayushgupta/Desktop/Hospital Management System_Text based /data/rooms.txt";
public void displayData() {
    File file = new File(ROOM_FILE);
    if (!file.exists() || file.length() == 0) {
        System.out.println(" No room records found.");
        return;
    }

    System.out.println("\n===== Room Information =====");
    System.out.printf("%-10s %-15s %-10s %-20s%n", "Room No", "Availability", "Price",
"Room Type");
    System.out.println("-------------------------------------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] details = line.split(",");
            if (details.length == 4) {
                String roomNo = details[0].trim();
                String availability = details[1].trim();
                String price = details[2].trim();
                String roomType = details[3].trim();

                if (availability.equalsIgnoreCase("Occupied")) {
                    System.out.printf("%-10s %-15s %-10s %-20s  Room is Occupied!%n", roomNo,
availability, price, roomType);
                } else {
                    System.out.printf("%-10s %-15s %-10s %-20s%n", roomNo, availability, price,
roomType);
                }
            }
        }
    } catch (IOException e) {
        System.out.println(" Error reading room data: " + e.getMessage());
    }
}

public static void main(String[] args) {
    Room room = new Room();
    room.displayData();
}

}
```

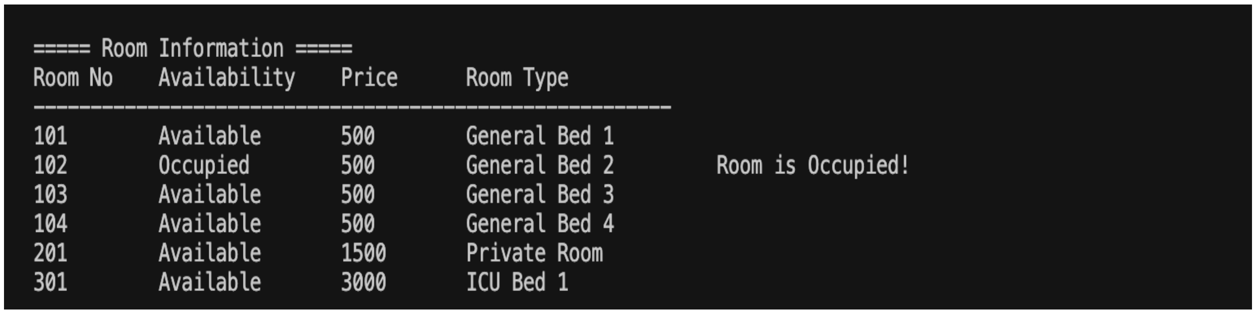
Concepts used:

- Interface Implementation
- Method Overloading
- File Handling
- Exception Handing
- String Manipulation

Explanation:

This module introduces a Room class that extends the Displayable interface and overrides the displayData() method to display room information that is saved in a text file. The method reads the room records file and prints each room's number, availability, price, and type in a well-formatted table. If the room is tagged as "Occupied," a note is also printed alongside it. The application of method overriding guarantees that the displayData() method exhibits certain functionality in rooms while preserving consistency with the other classes supporting the Displayable interface. It also applies file handling to load room data, conditional statements to verify occupancy, and formatted print to improve readability. This indicates runtime polymorphism and correct separation of responsibilities in object-oriented programming.

Output:



```
===== Room Information =====
Room No    Availability    Price      Room Type
--------------------------------------------------------
101        Available       500        General Bed 1
102        Occupied        500        General Bed 2        Room is Occupied!
103        Available       500        General Bed 3
104        Available       500        General Bed 4
201        Available       1500       Private Room
301        Available       3000       ICU Bed 1
```

10. Billing Detail Module

Code:

```
import java.io.*; import java.util.Scanner;
public class BillingDetails { private static final String PATIENT_FILE = "/Users/ayushgupta/
Desktop/Hospital Management System_Text based /data/patients.txt";
public static void updateBillingDetails(Scanner scanner) {
   String[] patients = new String[100];
   int patientCount = 0;

 System.out.println("\n===== Patient Billing Details =====");
   System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-15s %-15s
%-15s%n",
         "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining Amount");

System.out.println("------------------------------------------------------------------------------------------
---------------------------");

   try (BufferedReader reader = new BufferedReader(new FileReader(PATIENT_FILE))) {
      String line;
      while ((line = reader.readLine()) != null && patientCount < patients.length) {
         String[] details = line.split(",", -1);
         if (details.length == 11) {
            System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-15s %-15s
%-15s%n",
                  details[0], details[1], details[2], details[3], details[4], details[5], details[6],
                  details[7], details[8], details[9], details[10]);
            patients[patientCount++] = line;
         } else {
            System.out.println("invalid record: " + line);
         }
      }
   } catch (IOException e) {
      System.out.println(" Error reading patient data: " + e.getMessage());
      return;
```

```
}

System.out.print("\nEnter Patient ID to update billing details: ");
String patientId = scanner.nextLine().trim();
boolean patientFound = false;

for (int i = 0; i < patientCount; i++) {
   String[] details = patients[i].split(",");

   if (details[0].equals(patientId)) {
      patientFound = true;
      System.out.println("\n Updating billing details for: " + details[1]);

      double depositAmount = Double.parseDouble(details[5]);
      double totalBill = Double.parseDouble(details[9]);
      double remainingAmount = Double.parseDouble(details[10]);

      while (true) {
         System.out.println("\nChoose an option:");
         System.out.println("1. Update Total Bill");
         System.out.println("2. Pay Remaining Amount");
         System.out.println("3. Exit");
         System.out.print("Enter your choice: ");
         String choice = scanner.nextLine().trim();

          if (choice.equals("1")) {
            System.out.print("Enter new Total Bill Amount: ");
            double newTotalBill = Double.parseDouble(scanner.nextLine().trim());

            if (totalBill == 0) {
               remainingAmount = newTotalBill - depositAmount;
            } else {
               remainingAmount += (newTotalBill - totalBill);
            }

            totalBill = newTotalBill;
            System.out.println(" Updated Remaining Amount: " + remainingAmount);
            } else if (choice.equals("2")) {
                System.out.print("Enter amount to pay: ");
                double payment = Double.parseDouble(scanner.nextLine().trim());

                if (payment > remainingAmount) {
                    System.out.println(" Payment exceeds remaining amount! Enter a valid
amount.");
                    continue;
                }

                remainingAmount -= payment;
                System.out.println(" Payment successful! New Remaining Amount: " +
remainingAmount);

            } else if (choice.equals("3")) {
                break;
            } else {
                System.out.println(" Invalid choice. Please enter 1, 2, or 3.");
                continue;
            }

            details[9] = String.valueOf(totalBill);
            details[10] = String.valueOf(remainingAmount);
            patients[i] = String.join(",", details);
        }
        break;
    }
}

if (!patientFound) {
    System.out.println(" No patient found with ID: " + patientId);
    return;
}

try (BufferedWriter writer = new BufferedWriter(new FileWriter(PATIENT_FILE))) {
    for (int i = 0; i < patientCount; i++) {
        writer.write(patients[i]);
        writer.newLine();
    }
    System.out.println(" Billing details updated successfully!");
} catch (IOException e) {
    System.out.println("Error updating patient data: " + e.getMessage());
}
}

}
```

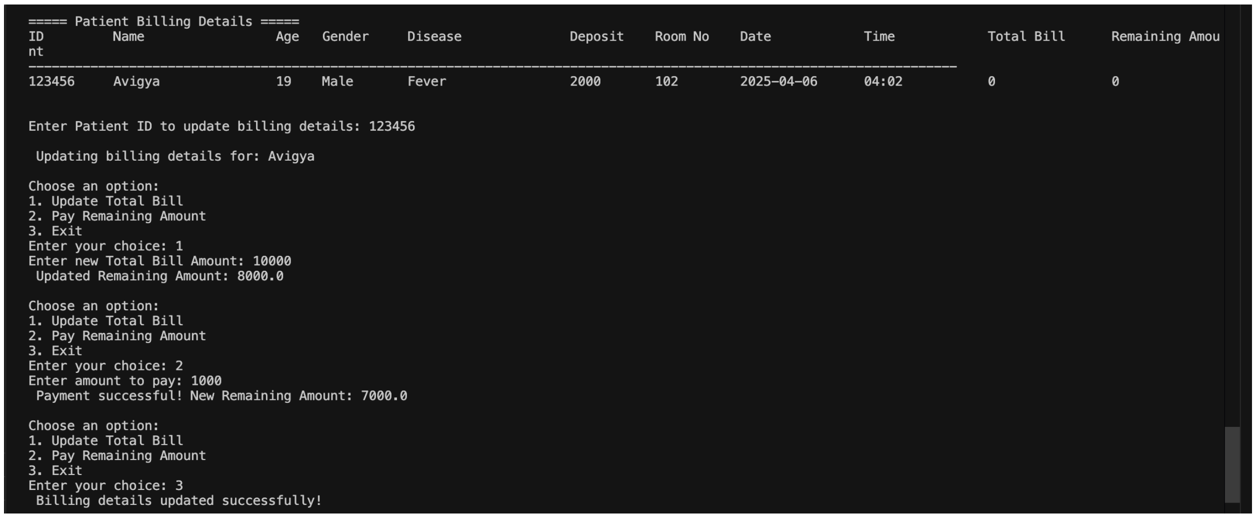
Concepts Used:

- Static Method
- File Reading and Writing
- Formatted Output
- Array and String Manipulation
- Exception Handling
- Scanner

Explanation:

The BillingDetails class handles the financial transactions related to patients in the hospital. It reads patient records from a file and displays them in a structured format. Users can update the total bill amount or make payments towards the remaining balance. The system ensures that changes to billing are reflected in memory and saved back to the file. Input validation is included to prevent overpayment. This module uses file I/O operations, arrays, loops, and parsing numeric values from strings to manage billing efficiently. It promotes robust input handling and improves user interaction through menu-driven options.

Output:



```
===== Patient Billing Details =====
ID         Name                  Age  Gender     Disease              Deposit    Room No    Date            Time            Total Bill      Remaining Amou
nt
------------------------------------------------------------------------------------------------------------------------
123456     Avigya                19   Male       Fever                2000       102        2025-04-06      04:02           0               0


Enter Patient ID to update billing details: 123456

 Updating billing details for: Avigya

Choose an option:
1. Update Total Bill
2. Pay Remaining Amount
3. Exit
Enter your choice: 1
Enter new Total Bill Amount: 10000
 Updated Remaining Amount: 8000.0

Choose an option:
1. Update Total Bill
2. Pay Remaining Amount
3. Exit
Enter your choice: 2
Enter amount to pay: 1000
 Payment successful! New Remaining Amount: 7000.0

Choose an option:
1. Update Total Bill
2. Pay Remaining Amount
3. Exit
Enter your choice: 3
 Billing details updated successfully!
```

11. View Patient Module

Code:

```
import java.io.*;
public class ViewPatients implements Displayable { private static final String FILE_PATH = "/
Users/ayushgupta/Desktop/Hospital Management System_Text based /data/patients.txt";
public void displayData() {
    File file = new File(FILE_PATH);

    if (!file.exists()) {
        System.out.println(" Error: File does not exist at " + FILE_PATH);
        return;
    }
    if (file.length() == 0) {
        System.out.println(" No patient records found (File is empty).");
        return;
    }

    System.out.println("\n===== Patient Records =====");
    System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
            "ID", "Name", "Age", "Gender", "Disease", "Deposit", "Room No", "Date", "Time",
"Total Bill", "Remaining Amount");
System.out.println("------------------------------------------------------------------------------------------
-----------------------------------");

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        boolean hasRecords = false;

        while ((line = reader.readLine()) != null) {
            line = line.trim();
            if (line.isEmpty()) {
                continue;
            }

            String[] details = line.split(",", 11);

            if (details.length == 11) {
                hasRecords = true;
                System.out.printf("%-10s %-20s %-5s %-10s %-20s %-10s %-10s %-15s %-12s %-12s
%-12s%n",
                        details[0].trim(), details[1].trim(), details[2].trim(),
                        details[3].trim(), details[4].trim(), details[5].trim(), details[6].trim(),
                        details[7].trim(), details[8].trim(), details[9].trim(), details[10].trim());
            } else {
                System.out.println(" Skipping invalid record: " + line);
            }
        }

        if (!hasRecords) {
            System.out.println("\n No valid patient records found.");
        }

    } catch (IOException e) {
        System.out.println(" Error reading patient data: " + e.getMessage());
    }
}
```

```
public static void main(String[] args) {
   ViewPatients viewPatients = new ViewPatients();
   viewPatients.displayData();
}

}
```

Concepts used:

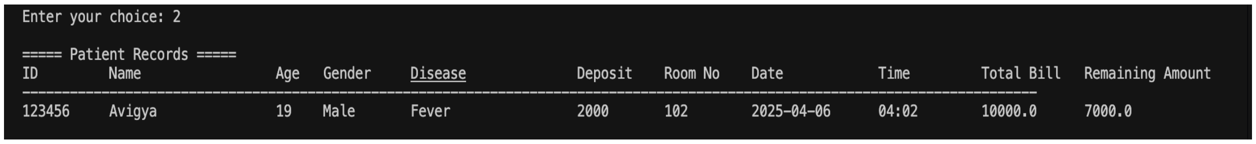
- Fle Handling
- Exception Handling
- Interface
- String Handling
- Constructor
- Overriding

Explanation:

The ViewPatients class is part of a hospital management system and is designed to display patient records stored in a text file. It implements the Displayable interface, which dictates that the class should provide a method for displaying data. The class defines a constant FILE_PATH which specifies the location of the file containing the patient records.
Upon execution, the displayData() method first checks if the file exists at the specified path. If the file doesn't exist, the program prints an error message indicating that the file cannot be found. If the file exists but is empty, it will output a message stating that no patient records are available.
If the file contains data, the program proceeds to read the file line by line using a BufferedReader. Each line of the file is expected to represent one patient's information, where the data is separated by commas. The class expects each record to contain 11 fields, corresponding to patient details such as patient ID, name, age, gender, disease, deposit amount, room number, admission date, admission time, total bill, and remaining amount. The program checks that each record has exactly 11 fields; if any record has fewer or more, it is considered invalid, and the program skips displaying that record, logging a message that the record is invalid.

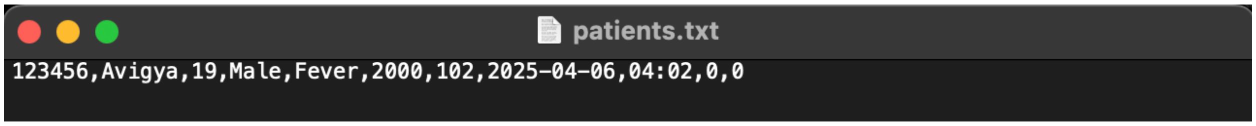
For valid records, the program formats and displays the patient details in a structured table with appropriate headers like "ID", "Name", "Age", "Disease", etc., ensuring each column is properly aligned for easy readability. If no valid records are found, a message is printed to inform the user that no valid patient data was available.

The class also handles potential IOException that may occur while reading the file, ensuring that any file access errors are caught and an appropriate error message is displayed.
In the main method, an instance of ViewPatients is created, and the displayData() method is called to initiate the reading and displaying process. This class offers an efficient way to display patient data while ensuring that invalid or missing data is handled gracefully.
Output:



```
Enter your choice: 2

===== Patient Records =====
ID        Name                 Age   Gender    Disease              Deposit   Room No   Date          Time       Total Bill   Remaining Amount
-------------------------------------------------------------------------------------------------------------------------------
123456    Avigya               19    Male      Fever                2000      102       2025-04-06    04:02      10000.0      7000.0
```

Records in patient.txt:



```
123456,Avigya,19,Male,Fever,2000,102,2025-04-06,04:02,0,0
```

Conclusion:

Hospital Employee and Room Management System efficiently implements Java programming for managing and optimizing the hospital employee and room data. Using concepts of Object-Oriented Programming (OOP), the system keeps different entities like employees (doctors, nurses, and admin staff) and room information well-organized. The information is stored effectively in text files, offering a lightweight and manageable solution without having to use extensive database systems.
The program's modular design, each having distinct classes for employees, rooms, and data handling, is more maintainable and scalable. For instance, varying types of employees are processed through the use of inheritance, thus the code is more adaptable to future enhancements or additions. The user interface of the system is minimal and efficient, yet meant for easy use so that users can view, insert, and modify employee and room details without hindrances.
The system also provides patient records and billing information support so that the hospital staff can monitor patient information and payments with ease. This further adds to the system's functionality to meet all the requirements of hospital management.
Furthermore, the file handling functionality guarantees information remains even when the program terminates. Through the reading from and writing to files, the system properly keeps hospital records for recall purposes, as straightforward algorithms provide effective data processing.
In summary, the project illustrates not only the applicability of Java in practical real-world applications but also emphasizes the need for OOP in the handling of sophisticated data systems. The system offers a lean, easy-to-use system for handling hospital personnel, room status, patient files, and billing information that can be a useful addition to small-scale healthcare operations.