INDEX

A) CONSOLE BASED

B) JDBC-MYSQL

A) CONSOLE BASED

ABSTRACT

The "Hospital Management System" is a console-based application developed using Java to manage hospital operations efficiently. It provides core functionalities such as adding, updating, deleting, and viewing both patient and doctor records. Each patient's information includes a unique ID, name, age, gender, ailment, and the doctor assigned, while doctor records contain details like ID, name, department, and specialization. The system is designed using object-oriented programming principles, ensuring modularity and maintainability through well-structured classes such as Patient, Doctor, AppointmentManager, and RecordManager.

One of the key modules is the appointment scheduler, which allows linking patients with doctors based on their availability and specialization. The system also supports tracking patient history and managing doctor schedules. Data is temporarily handled using Java's ArrayList, and data persistence is achieved through object serialization, enabling the saving and loading of records from files.

This project demonstrates effective use of Java concepts such as file I/O, exception handling, serialization, and collection frameworks. It operates through a text-based menu interface, making it simple and user-friendly. Additional features such as billing and prescription management can also be integrated to extend functionality, allowing the system to generate invoices based on treatments and maintain prescribed medications for each patient. Overall, it serves as a solid foundational project for beginners aiming to develop small-scale, real-world management systems using Java, while gaining hands-on experience in handling structured data and building interactive, modular applications.

.

2.1SCOPE OF PROJECT

- Storing and managing details of patients and doctors.
- Scheduling and tracking appointments.
- Recording patient history and assigned treatments.
- Generating basic billing based on consultation or treatment.
- Managing prescriptions issued to patients.
- Persisting data using Java serialization for long-term storage.
- Providing a console-based user interface for ease of access.

2.2 TECHNOLOGIES USED

1. Java Programming Language

The entire system is developed using Java, a widely-used, object-oriented programming language known for its portability and robustness. Java provides built-in libraries and APIs that simplify file handling, data storage, and user input processing. Its strong OOP foundation allows for clean and modular code design.

2. Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming methodology that organizes software design around data, or objects, rather than functions and logic. In OOP, a class serves as a blueprint for creating objects, which are individual instances containing both data and methods that operate on that data. The key principles of OOP include encapsulation, abstraction, inheritance, and polymorphism. Encapsulation involves bundling data and methods together while restricting direct access to some components. Abstraction hides complex internal details and only exposes the necessary functionalities, making it easier to use and manage. Inheritance allows one class to inherit properties and behaviors from another

promoting code reuse and scalability. Polymorphism enables a single interface to represent different types of behavior, allowing flexibility in code execution. Overall, OOP helps build modular, maintainable, and reusable software by mimicking real-world entities and promoting a structured, organized coding style.

3. Collection Framework

The Collection Framework in Java is a set of classes and interfaces that provide a standardized architecture to store, manipulate, and retrieve groups of objects efficiently. It is part of the java.util package and includes data structures such as lists, sets, queues, and maps along with algorithms like sorting and searching. The framework is designed to handle collections of objects with ease, offering powerful and flexible tools for developers.

- Collection – The root interface of the collection hierarchy.
- List – An ordered collection that allows duplicates (e.g., ArrayList, LinkedList).
- Set – A collection that does not allow duplicates (e.g., HashSet, TreeSet).
- Queue – A collection for holding elements prior to processing (e.g., PriorityQueue, ArrayDeque).
- Map – An object that maps keys to value, with no duplicate keys (e.g., HashMap, TreeMap).

4.File Handling

File handling in Java refers to the process of creating, reading, writing, and deleting files on the file system using Java code. It is a fundamental part of Java programming used for persistent data storage, meaning data remains available even after the program has ended. Java provides several classes and methods in the java.io and java.nio.file packages to perform file operations. Commonly used classes include:

- **File** – Represents a file or directory path. Used to create, delete, and check properties of files or directories.

• **FileReader / BufferedReader** – Used to read data from a file (character by character or line by line).

• **FileWriter / BufferedWrite**r – Used to write character data into a file.

• **FileInputStream / FileOutputStream** – Used for reading and writing binary data.

• **PrintWrite**r – Used to write formatted data to a file.

5. Serialization

Serialization in Java is the process of converting an object into a stream of bytes so that it can be easily saved to a file, sent over a network, or stored in memory. This allows objects to be persisted or transferred and then reconstructed (deserialized) later into the same state.

• Java provides built-in support through the java.io.Serializable interface.

• A class must implement Serializable to be eligible for serialization.

• Use ObjectOutputStream to write the object and ObjectInputStream to read it back.

• Fields marked with the transient keyword are not serialized.

• Serialization maintains the object structure and data, not the class methods.

6. Exception Handling

The project makes use of Java's built-in exception handling mechanism to manage errors gracefully. This includes wrapping risky operations, such as file I/O and user input, in try-catch blocks. By doing so, the system can catch and respond to runtime errors without crashing. For example, if a file is not found during loading, the program will display a friendly message and continue working, instead of terminating unexpectedly. This improves the reliability and user friendliness of the application.

7. Development Environment

The project is platform-independent and can be developed and run using any Java Development Kit (JDK) version 8 or above. It is suitable for both command-line execution and development within Integrated Development Environments (IDEs) like IntelliJ IDEA, Eclipse, or NetBeans. These environments provide additional tools like debugging, code completion, and project organization, which help speed up the development process. Since the project does not require any external libraries, it remains lightweight and easy to set up for learners and developers alike.

MODULE DESCRIPTION

1. Patient Management Module

Purpose:

To manage all operations related to patients, including adding, updating, viewing, and deleting patient records.

Class : Patient

Features:

- Add new patient with details (ID, name, age, gender, ailment, assigned doctor).
- Edit or delete existing patient records.
- Search and view patient information.
- Store patient history for future reference.

2. Doctor Management Module

Purpose:

To maintain doctor records and manage their availability for appointments.

Class : Doctor

Features:

- Add and manage doctor profiles (ID, name, specialization, department).
- Update doctor availability.
- View list of doctors and their specializations.

3. Appointment Management Module

Purpose:

To handle the scheduling, updating, and cancellation of appointments between doctors and patients.

Class : Appointment

Features:

- Schedule new appointments based on doctor availability.
- Cancel or reschedule existing appointments.
- View upcoming appointments by doctor or patient.

4. Data Persistence Module

Purpose:

To save and load data from files to ensure continuity across sessions.

Class : DataHandler

Features:

- Serialize and deserialize data for patients, doctors, and appointments.
- Load data on program start and save before exit.
- Ensure data integrity during I/O operations.

5. User Interface Module

Purpose:

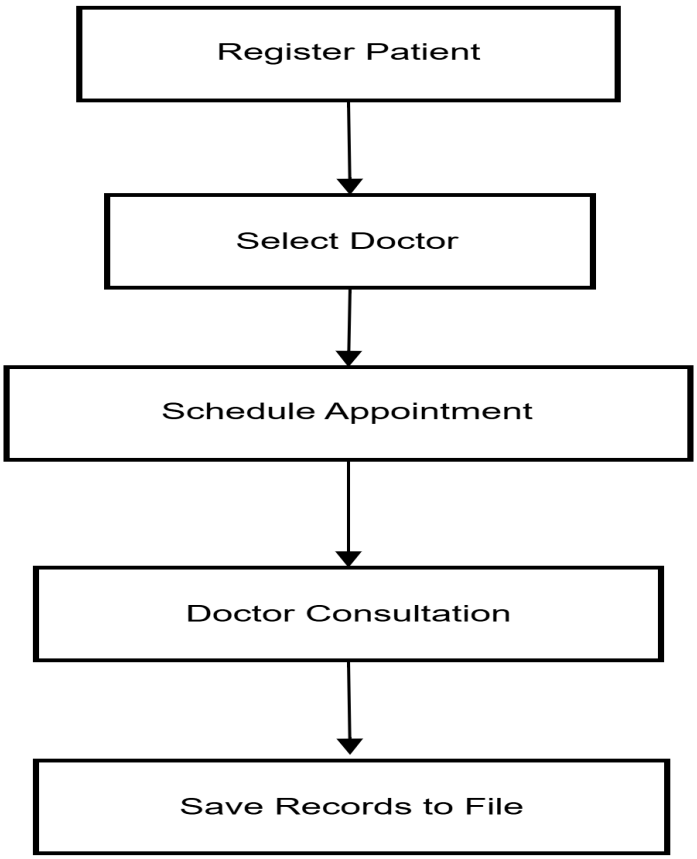
To provide an interactive text-based menu system for navigating through the application.

Class : HospitalSystem (main class)

Features:

- Display menu-driven options for all modules.
- Take user input and call relevant methods.
- Handle invalid inputs gracefully using exception handling.

FLOW DIAGRAM



DESCRIPTION:

1. **Register Patient:**

   The process begins with registering a new patient in the system. This includes capturing essential details such as name, age, gender, medical history, and contact information.

2. **Select Doctor:**

   After the patient is registered, the next step involves selecting a suitable doctor based on the patient's medical condition and available specialties. The system identifies available doctors from the list and assigns one based on the patient's needs.

3. **Schedule Appointment:**

   Once the doctor is selected, the system schedules an appointment at a mutually available time. This ensures that both the doctor and the patient have time slots aligned for consultation.

4. **Doctor Consultation:**

   The patient meets with the doctor for consultation. During this session, the doctor assesses the patient's condition, makes a diagnosis, and discusses treatment options. The doctor may also order additional tests or prescribe medication.

5. **Save Records to File:**

   Finally, all data—patient records, doctor details, appointments, prescriptions, and billing information—are saved to a file. This ensures that the hospital's data is securely stored and can be accessed in future sessions.

DESCRIPTION ON JAVA

Java is a powerful, versatile, and widely-used object-oriented programming language initially developed by Sun Microsystems in 1995 and now owned by Oracle Corporation. It was designed with the concept of "Write Once, Run Anywhere" (WORA), which means that once a Java program is compiled, it can run on any platform without modification, as long as it has a Java Virtual Machine (JVM). This platform independence is one of Java's most significant advantages and makes it ideal for cross-platform development.

Java is a compiled and interpreted language. The source code written in .java files is compiled by the Java compiler into bytecode, which is then interpreted and executed by the JVM. This layered approach ensures that Java applications are both secure and portable.

Additionally, Java provides features like automatic memory management (garbage collection), exception handling, and multi-threading, which help developers create efficient, reliable, and robust applications.

Key Features of Java:

1. Platform Independent

- Java follows the principle of "Write Once, Run Anywhere" (WORA).
- Compiled Java code (bytecode) can run on any system with a Java Virtual Machine (JVM).

2. Object-Oriented

- Java is built on object-oriented principles like inheritance, encapsulation, polymorphism, and abstraction, which help in modular and reusable code.

3. Simple and Easy to Learn

- Java has a clean and readable syntax similar to C/C++, but without complex features like pointers and operator overloading.

4. Secure

- Java has built-in security features like bytecode verification, sandboxing, and the absence of direct memory access, making it ideal for networked and web-based applications.

5. Robust

- Java has strong memory management, exception handling, and type checking, reducing the risk of crashes or memory leaks.

6. Multithreaded

- Java allows execution of multiple threads simultaneously using built-in support for multithreading, making programs faster and more efficient.

7. Distributed

• Java provides built-in libraries like RMI (Remote Method Invocation) and socket programming, allowing the development of distributed applications.

8. High Performance

• Although Java is interpreted, the use of Just-In-Time (JIT) compilers makes execution fast and efficient.

9. Dynamic and Extensible

• Java supports dynamic linking of classes and includes tools to dynamically load new code and update applications on the fly.

10. Rich Standard Library

• Java offers an extensive set of APIs and libraries for everything from data structures to networking, GUIs, and database connections.

SOURCE CODE

Application.java

```
package hospitalManagementSystem;
import java.util.*;
public class Application {
    static Scanner sc = new Scanner(System.in);
    static List<Patient> patients = new ArrayList<>();
    static List<Doctor> doctors = new ArrayList<>();
    static List<Appointment> appointments = new ArrayList<>();
```

```
public static void main(String[] args) {
    while (true) {
        System.out.println("\n--- HOSPITAL MANAGEMENT SYSTEM ---");
        System.out.println("1. Manage Patients");
        System.out.println("2. Manage Doctors");
        System.out.println("3. Manage Appointments");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine(); // consume leftover newline

        switch (choice) {
            case 1:
                managePatients();
                break;
            case 2:
                manageDoctors();
                break;
            case 3:
                manageAppointments();
                break;
            case 4:
                System.out.println("Exiting... Thank you!");
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice!");
        }
    }
}
```

```
static void managePatients() {
    while (true) {
        System.out.println("\n--- PATIENT MANAGEMENT ---");
        System.out.println("1. Add Patient");
        System.out.println("2. View Patients");
        System.out.println("3. Go Back");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine();

        switch (choice) {
            case 1:
                System.out.print("Enter patient name: ");
                String pname = sc.nextLine();
                System.out.print("Enter patient age: ");
                int age = sc.nextInt();
                sc.nextLine();
                patients.add(new Patient(pname, age));
                System.out.println("Patient added successfully!");
                break;
            case 2:
                System.out.println("\n--- List of Patients ---");
                for (Patient p : patients) {
                    System.out.println(p);
                }
                break;
            case 3:
                Return;
```

```
            default:
                System.out.println("Invalid choice!");
        }
    }
}

static void manageDoctors() {
    while (true) {
        System.out.println("\n--- DOCTOR MANAGEMENT ---");
        System.out.println("1. Add Doctor");
        System.out.println("2. View Doctors");
        System.out.println("3. Go Back");
        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine();

        switch (choice) {
            case 1:
                System.out.print("Enter doctor name: ");
                String dname = sc.nextLine();
                System.out.print("Enter specialization: ");
                String specialization = sc.nextLine();
                doctors.add(new Doctor(dname, specialization));
                System.out.println("Doctor added successfully!");
                break;
            case 2:
                System.out.println("\n--- List of Doctors ---");
                for (Doctor d : doctors) {
                    System.out.println(d);
```

```
            }
            break;
        case 3:
            return;
        default:
            System.out.println("Invalid choice!");
      }
   }
}

static void manageAppointments() {
   while (true) {
      System.out.println("\n--- APPOINTMENT MANAGEMENT ---");
      System.out.println("1. Book Appointment");
      System.out.println("2. View Appointments");
      System.out.println("3. Go Back");
      System.out.print("Enter your choice: ");
      int choice = sc.nextInt();
      sc.nextLine();

      switch (choice) {
         case 1:
            if (patients.isEmpty() || doctors.isEmpty()) {
               System.out.println("Add patients and doctors first!");
               break;
            }
            System.out.println("Select Patient:");
            for (int i = 0; i < patients.size(); i++) {
```

```
            System.out.println((i + 1) + ". " + patients.get(i));
        }
        int pIndex = sc.nextInt() - 1;
        sc.nextLine();
        if (pIndex < 0 || pIndex >= patients.size()) {
            System.out.println("Invalid patient selected.");
            break;
        }

        System.out.println("Select Doctor:");
        for (int i = 0; i < doctors.size(); i++) {
            System.out.println((i + 1) + ". " + doctors.get(i));
        }
        int dIndex = sc.nextInt() - 1;
        sc.nextLine();
        if (dIndex < 0 || dIndex >= doctors.size()) {
            System.out.println("Invalid doctor selected.");
            break;
        }

        System.out.print("Enter appointment date (YYYY-MM-DD): ");
        String date = sc.nextLine();
        appointments.add(new Appointment(patients.get(pIndex), doctors.get(dIndex), date));
        System.out.println("Appointment booked successfully!");
        break;

    case 2:
        System.out.println("\n--- List of Appointments ---");
        for (Appointment a : appointments) {
```

```
                System.out.println(a);
            }
            break;

        case 3:
            return;

        default:
            System.out.println("Invalid choice!");
      }
    }
  }
}
```

Patient.java

```
package hospitalManagementSystem;
public class Patient {
  private String name;
  private int age;

  public Patient(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public String toString() {
    return "Patient Name: " + name + ", Age: " + age;
  }
}
```

Doctor.java

```
package hospitalManagementSystem;
public class Doctor {
    private String name;
    private String specialization;
    public Doctor(String name, String specialization) {
        this.name = name;
        this.specialization = specialization;
    }
    public String toString() {
        return "Doctor Name: " + name + ", Specialization: " + specialization;
    }
}
```

Appointment.java

```
package hospitalManagementSystem;

public class Appointment {
    private Patient patient;
    private Doctor doctor;
    private String date;

    public Appointment(Patient patient, Doctor doctor, String date) {
        this.patient = patient;
        this.doctor = doctor;
        this.date = date;
    }
```

```
    public String toString() {
        return "Appointment: " + patient + " with " + doctor + " on " + date;
    }
}
```

OUTPUT :

```
--- HOSPITAL MANAGEMENT SYSTEM ---
1. Manage Patients
2. Manage Doctors
3. Manage Appointments
4. Exit
Enter your choice: 1

--- PATIENT MANAGEMENT ---
1. Add Patient
2. View Patients
3. Go Back
Enter your choice: 1
Enter patient name: Vinoth
Enter patient age: 34
Patient added successfully!

--- PATIENT MANAGEMENT ---
1. Add Patient
2. View Patients
3. Go Back
Enter your choice: 2

--- List of Patients ---
Patient Name: Vinoth, Age: 34
```

```
--- HOSPITAL MANAGEMENT SYSTEM ---
1. Manage Patients
2. Manage Doctors
3. Manage Appointments
4. Exit
Enter your choice: 2

--- DOCTOR MANAGEMENT ---
1. Add Doctor
2. View Doctors
3. Go Back
Enter your choice: 1
Enter doctor name: Kaviya
Enter specialization: Neurologist
Doctor added successfully!

--- DOCTOR MANAGEMENT ---
1. Add Doctor
2. View Doctors
3. Go Back
Enter your choice: 2

--- List of Doctors ---
Doctor Name: Kaviya, Specialization: Neurologist
```

```
Enter your choice: 3

--- APPOINTMENT MANAGEMENT ---
1. Book Appointment
2. View Appointments
3. Go Back
Enter your choice: 1
Select Patient:
1. Patient Name: Vinoth, Age: 34
2. Patient Name: Suba, Age: 56
2
Select Doctor:
1. Doctor Name: Kaviya, Specialization: Neurologist
1
Enter appointment date (YYYY-MM-DD): 2025-05-15
Appointment booked successfully!

--- APPOINTMENT MANAGEMENT ---
1. Book Appointment
2. View Appointments
3. Go Back
Enter your choice: 2

--- List of Appointments ---
Appointment: Patient Name: Vinoth, Age: 34 with Doctor Name: Kaviya, Specialization: Neurologist on 2025-05-13
Appointment: Patient Name: Suba, Age: 56 with Doctor Name: Kaviya, Specialization: Neurologist on 2025-05-15
```

CONCLUSION

The Hospital Management System is a Java-based console application designed to streamline the management of patients, doctors, and appointments in healthcare facilities. By automating key administrative tasks such as maintaining patient records, managing doctor profiles, and scheduling appointments, the system enhances efficiency and reduces human error.

The project adopts a modular, object-oriented approach, with dedicated classes for each core function—Patient, Doctor, Appointment, and DataHandler—ensuring maintainability and scalability. It leverages Java's JDBC for seamless interaction with a MySQL database, enabling reliable data storage and retrieval. Additionally, features like exception handling and user input validation improve system robustness and usability.

Designed for small to medium healthcare facilities, the system provides a solid foundation that can be extended with features like billing, inventory, or advanced scheduling in future versions. This project exemplifies how Java can be effectively used to build structured and functional management application

B) JDBC-MySQL

ABSTRACT

The "Hospital Management System" is a console-based application developed using Java to manage hospital operations efficiently. It provides core functionalities such as adding, updating, deleting, and viewing both patient and doctor records. Each patient's information includes a unique ID, name, age, gender, ailment, and the doctor assigned, while doctor records contain details like ID, name, department, and specialization. The system is designed using object-oriented programming principles, ensuring modularity and maintainability through well-structured classes such as Patient, Doctor, AppointmentManager, and RecordManager.

One of the key modules is the appointment scheduler, which allows linking patients with doctors based on their availability and specialization. The system also supports tracking patient history and managing doctor schedules. Data is temporarily handled using Java's ArrayList, and data persistence is achieved through object serialization, enabling the saving and loading of records from files.

This project demonstrates effective use of Java concepts such as file I/O, exception handling, serialization, and collection frameworks. It operates through a text-based menu interface, making it simple and user-friendly. Additional features such as billing and prescription management can also be integrated to extend functionality, allowing the system to generate invoices based on treatments and maintain prescribed medications for each patient. Overall, it serves as a solid foundational project for beginners aiming to develop small-scale, real-world management systems using Java, while gaining hands-on experience in handling structured data and building interactive, modular applications.

INTRODUCTION

A Hospital Management System (HMS) is a comprehensive software solution designed to manage the daily activities of a hospital efficiently.It handles patient information, doctor schedules, appointments, billing, and medical records in a centralized manner.HMS helps streamline operations and reduces the need for manual data entry, minimizing errors.It ensures that patient data is stored securely and can be retrieved quickly when needed.

Doctors and staff can access real-time information, improving the quality of healthcare services.The system supports appointment scheduling, making it easier for patients to consult doctors on time.Billing and payment processes become faster and more accurate with automated calculations.HMS improves communication between different hospital departments.It enhances resource management by tracking bed occupancy, staff availability, and inventory.Overall, HMS boosts efficiency, reduces administrative workload, and supports better patient care.

2.1 Scope of the Project

The scope of the project includes:

Patient Management:

- Adding new patients to the system.
- Updating patient details (e.g., contact information, medical history).
- Deleting patient records when necessary.
- Viewing details of all patients or a specific patient.

Doctor Management:

- Adding, updating, and deleting doctor records.
- Managing doctor specialization and contact details.
- Viewing details of all doctors or a specific doctor.

Appointment Management:

- Scheduling appointments for patients with doctors.
- Updating appointment details.
- Canceling appointments when necessary.
- Viewing scheduled appointments and their status.

Database Connectivity:

- Connecting to a MySQL database to store and retrieve patient, doctor, and appointment information.
- Using JDBC (Java Database Connectivity) for secure and efficient database transactions.

Console-based User Interface:

The system provides a text-based user interface that allows interaction through the console, enabling users to manage hospital records and appointments.

The system is designed to manage operations in small to medium-sized hospitals and can be expanded to accommodate more complex features as needed.

2.2 Technologies Used

Java: The application is developed using the Java programming language, chosen for its platform independence and object-oriented approach, ensuring scalability and flexibility in system development.

MySQL: A relational database management system used for storing hospital data securely. MySQL ensures high-performance data retrieval and scalability.

JDBC (Java Database Connectivity): A Java API that enables the application to interact with the MySQL database, allowing the system to store, retrieve, and manipulate patient, doctor, and appointment records.

Scanner Class: Utilized for capturing user input from the console to perform operations such as adding, updating, or deleting records and scheduling appointments.

MODULE DESCRIPTION

1. Patient Management Module

Purpose:

To manage all operations related to patients, including adding, updating, viewing, and deleting patient records.

Class : Patient

Features:

- Add new patient with details (ID, name, age, gender, ailment, assigned doctor).
- Edit or delete existing patient records.
- Search and view patient information.
- Store patient history for future reference.

2. Doctor Management Module

Purpose:

To maintain doctor records and manage their availability for appointments.

Class : Doctor

Features:

- Add and manage doctor profiles (ID, name, specialization, department).
- Update doctor availability.
- View list of doctors and their specializations.

3. Appointment Management Module

Purpose:

To handle the scheduling, updating, and cancellation of appointments between doctors and patients.

Class : Appointment

Features:

- Schedule new appointments based on doctor availability.
- Cancel or reschedule existing appointments.
- View upcoming appointments by doctor or patient.

4. Data Persistence Module

Purpose:

To save and load data from files to ensure continuity across sessions.

Class : DataHandler

Features:

- Serialize and deserialize data for patients, doctors, and appointments.
- Load data on program start and save before exit.
- Ensure data integrity during I/O operations.

5. User Interface Module

Purpose:

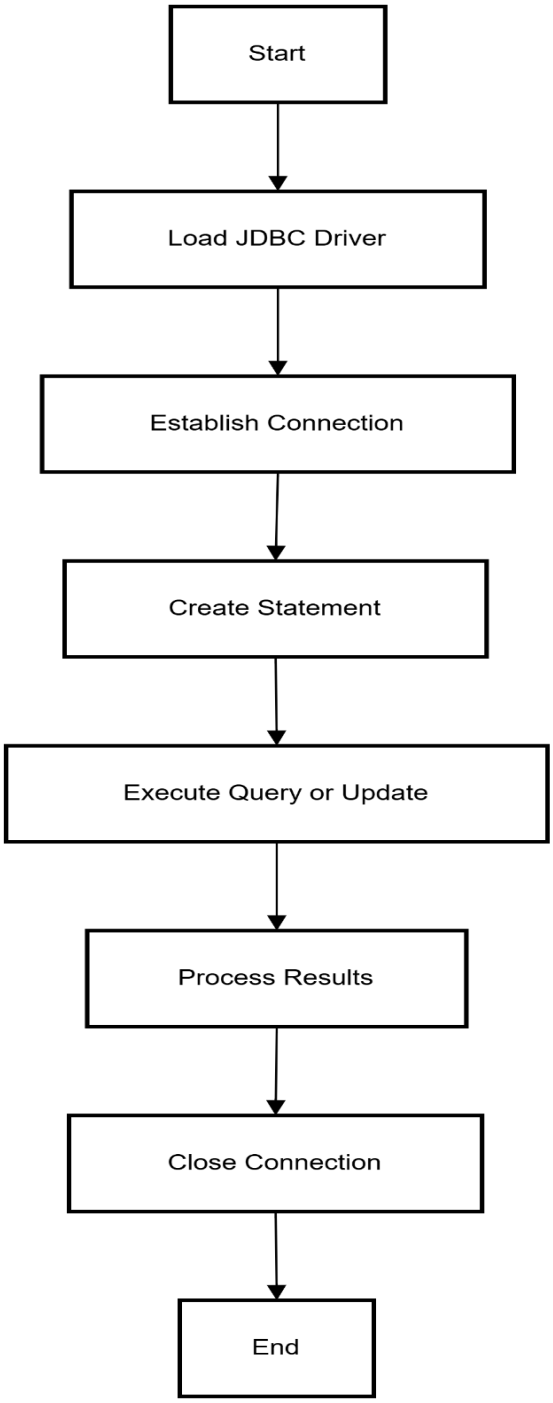
To provide an interactive text-based menu system for navigating through the application.

Class : HospitalSystem (main class)

Features:

- Display menu-driven options for all modules.
- Take user input and call relevant methods.
- Handle invalid inputs gracefully using exception handling.

FLOW DIAGRAM



DESCRIPTION ON JAVA

Java is the core programming language chosen for the development of the **Employee Payroll Management System** due to its numerous advantages, such as platform independence, robustness, ease of integration with databases, and strong support for object-oriented programming (OOP). Below, we delve deeper into the key features of Java that were utilized in the development of this system:

1. JDBC (Java Database Connectivity)

Java provides a powerful API known as **JDBC** (Java Database Connectivity) that allows developers to connect Java applications with databases like MySQL. This feature is critical for the **Employee Payroll Management System** since it relies on MySQL for storing and retrieving employee records and payroll data.

How JDBC Works:

• **Establishing Connection**: The system establishes a connection to the MySQL database using DriverManager.getConnection() by providing the necessary credentials such as the database URL, username, and password.

- **Executing SQL Queries**: JDBC allows the system to send SQL commands (such as SELECT, INSERT, UPDATE, and DELETE) to the database. The queries are executed via the Statement or PreparedStatement interfaces, which protect against SQL injection attacks and allow for parameterized queries.

• **ResultSet**: After executing queries like SELECT, a ResultSet object is used to retrieve the results of the query. The data can then be displayed to the user or processed further. • **Exception Handling**: The system uses JDBC's exception handling capabilities to catch SQL exceptions (e.g., connection errors, syntax errors) and manage them gracefully.

Key JDBC Components Used in the System:

- Connection: Establishes a connection to the database.
- Statement / PreparedStatement: Allows for the execution of SQL queries. • ResultSet: Retrieves the result of a SELECT query.
- SQLException: Handles any database-related errors that occur during interaction with the database.

2. Exception Handling

Java's **exception handling** mechanism plays a vital role in ensuring that errors and exceptions during runtime, particularly those related to database interactions, are caught and handled appropriately. The system uses Java's built-in try-catch blocks to manage potential exceptions, ensuring the program doesn't crash unexpectedly.

Why Exception Handling is Crucial:

- **Database Connectivity Errors**: If the connection to the MySQL database fails (e.g., incorrect credentials, server down), an exception is thrown, which is caught and logged to provide the user with meaningful feedback.
- **SQL Errors**: Errors like syntax mistakes in SQL queries, missing fields, or data type mismatches are handled by catching SQLException and displaying an appropriate error message to the user.
- **User Input Errors**: The system catches potential input errors (e.g., invalid employee data) and ensures that the application does not crash.

Implementation:

```
try {
    Connection conn = DriverManager.getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);
 // Execute queries
} catch (SQLException e) {
 System.err.println("Error: " + e.getMessage());
 // Log error or notify user
```

```
} finally {
// Close connections/resources
}
```

The finally block ensures that the database connection is closed after the operations, preventing resource leaks.

3. Object-Oriented Programming (OOP) Principles

Java is an **object-oriented programming (OOP)** language, meaning that the application is designed using classes and objects that represent real-world entities. This approach helps in organizing code into reusable and maintainable modules, which is particularly useful in a complex system like the **Hospital Management System**.

OOP Features Used in the System:

- **Encapsulation**: The system uses encapsulation to hide the internal workings of the system and expose only necessary details to the outside world. Each class has its attributes (fields) and methods that manipulate them. For instance, the Employee class encapsulates employee details, and the Payroll class encapsulates payroll details.
- **Abstraction**: The system uses abstraction to focus on high-level functionalities without exposing unnecessary implementation details. For example, classes like EmployeeManager and PayrollManager abstract the underlying logic for managing employee records and payroll generation.
- **Inheritance**: While inheritance is not explicitly used in the current system, Java's inheritance capabilities allow subclasses to extend base classes to inherit common functionality. This can be useful if the system needs to be extended with more specialized employee types in the future.
- **Polymorphism**: Methods like calculateNetSalary() can be overridden in subclasses to provide specific implementations if needed, such as applying special rules for different types of employees (e.g., part-time or full-time).

Benefits of OOP in this System:

- **Modularity**: Each class handles specific functionality, making the system modular. For instance, EmployeeManager handles employee records, while PayrollManager manages payroll-related operations.
- **Reusability**: Classes like Employee and Payroll can be reused across different parts of the system or even in future applications.
- **Maintainability**: Changes in business logic can be isolated within specific classes, reducing the risk of errors and making the system easier to update and maintain.
- **Scalability**: The modular structure allows for the addition of new features with minimal disruption to existing code.

4. Multithreading (Optional Future Expansion)

Though multithreading is not explicitly used in this basic version of the project, Java supports multithreading which could be used for more advanced scenarios, such as handling multiple payroll requests simultaneously or performing data processing tasks in the background.

SOURCE CODE :

JAVA JDBC APPLICATION

APP

```
package com.hospitalmanagement.hospital;
import java.sql.Connection;
import java.util.Scanner;
public class App {
   public static void main(String[] args) {
      try {
         Connection con = DBConnection.getConnection();
         if (con == null) {
            System.out.println("Exiting due to database error.");
            return;
         }
```

```
Scanner sc = new Scanner(System.in);
Patient patient = new Patient();
Doctor doctor = new Doctor();
Appointment appointment = new Appointment();

while (true) {
    System.out.println("\nHOSPITAL MANAGEMENT SYSTEM");
    System.out.println("1. Manage Patient");
    System.out.println("2. Manage Doctor");
    System.out.println("3. Manage Appointment");
    System.out.println("0. Exit");
    System.out.print("Enter your choice: ");
    int choice = Integer.parseInt(sc.nextLine());

    switch (choice) {
        case 1:
            managePatient(patient, con, sc);
            break;
        case 2:
            manageDoctor(doctor, con, sc);
            break;
        case 3:
            manageAppointment(appointment, con, sc);
            break;
        case 0:
            System.out.println("Exiting system...");
            try {
                con.close();
            } catch (Exception e) {
                System.out.println("Error closing connection: " + e.getMessage());}
```

```
                sc.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Try again!");
        }
      }
    } catch (Exception e) {
      System.out.println("Unexpected error occurred: " + e.getMessage());
      e.printStackTrace();
    }
}

private static void managePatient(Patient patient, Connection con, Scanner sc) {
    System.out.println("\n--- PATIENT MANAGEMENT ---");
    System.out.println("1. Add Patient");
    System.out.println("2. Update Patient");
    System.out.println("3. Delete Patient");
    System.out.println("4. View Patient by ID");
    System.out.print("Enter your choice: ");
    int ch = Integer.parseInt(sc.nextLine());

    switch (ch) {
      case 1:
        patient.addPatient(con, sc);
        break;
      case 2:
        patient.updatePatient(con, sc);
        Break;
```

```
            case 3:
                patient.deletePatient(con, sc);
                break;
            case 4:
                patient.viewPatientById(con, sc);
                break;
            default:
                System.out.println("Invalid option!");
        }
    }

    private static void manageDoctor(Doctor doctor, Connection con, Scanner sc) {
        System.out.println("\n--- DOCTOR MANAGEMENT ---");
        System.out.println("1. Add Doctor");
        System.out.println("2. Update Doctor");
        System.out.println("3. Delete Doctor");
        System.out.println("4. View Doctor by ID");
        System.out.print("Enter your choice: ");
        int ch = Integer.parseInt(sc.nextLine());

        switch (ch) {
            case 1:
                doctor.addDoctor(con, sc);
                break;
            case 2:
                doctor.updateDoctor(con, sc);
                break;
            case 3:
                doctor.deleteDoctor(con, sc);
```

```
            break;
        case 4:
            doctor.viewDoctorById(con, sc);
            break;
        default:
            System.out.println("Invalid option!");
    }
}
    private static void manageAppointment(Appointment appointment, Connection con, Scanner
sc) {
        System.out.println("\n--- APPOINTMENT MANAGEMENT ---");
        System.out.println("1. Add Appointment");
        System.out.println("2. Update Appointment");
        System.out.println("3. Delete Appointment");
        System.out.println("4. View Appointment by ID");
        System.out.print("Enter your choice: ");
        int ch = Integer.parseInt(sc.nextLine());

        switch (ch) {
            case 1:
                appointment.addAppointment(con, sc);
            break;
        case 2:
            appointment.updateAppointment(con, sc);
            break;
        case 3:
            appointment.deleteAppointment(con, sc);
            Break;
```

```
            case 4:
                appointment.viewAppointmentById(con, sc);
                break;
            default:
                System.out.println("Invalid option!");
        }
    }
}
```

PATIENT

```
package com.hospitalmanagement.hospital;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class Patient {
    public void addPatient(Connection con, Scanner sc) {
        try {
            System.out.print("Enter First Name: ");
            String firstName = sc.nextLine();

            System.out.print("Enter Last Name: ");
            String lastName = sc.nextLine();

            System.out.print("Enter Gender: ");
            String gender = sc.nextLine();

            System.out.print("Enter Date of Birth (yyyy-mm-dd): ");
            String dob = sc.nextLine();
```

```
        System.out.print("Enter Phone Number: ");
        String phone = sc.nextLine();

        System.out.print("Enter Email: ");
        String email = sc.nextLine();

        System.out.print("Enter Address: ");
        String address = sc.nextLine();

        System.out.print("Enter Blood Group: ");
        String bloodGroup = sc.nextLine();

        String sql = "INSERT INTO patient (first_name, last_name, gender, date_of_birth, phone_number,
email, address, blood_group, registration_date) VALUES (?, ?, ?, ?, ?, ?, ?, ?, NOW())";
        PreparedStatement stmt = con.prepareStatement(sql);

        stmt.setString(1, firstName);
        stmt.setString(2, lastName);
        stmt.setString(3, gender);
        stmt.setString(4, dob);
        stmt.setString(5, phone);
        stmt.setString(6, email);
        stmt.setString(7, address);
        stmt.setString(8, bloodGroup);

        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Patient added successfully!");
        }
```

```
  } catch (SQLException e) {
      System.out.println("Error while adding patient: " + e.getMessage());
   }
}

public void updatePatient(Connection con, Scanner sc) {
   try {
      System.out.print("Enter Patient ID to update: ");
      int patientId = Integer.parseInt(sc.nextLine());

      System.out.print("Enter New Phone Number: ");
      String phone = sc.nextLine();

      System.out.print("Enter New Address: ");
      String address = sc.nextLine();

      String sql = "UPDATE patient SET phone_number = ?, address = ? WHERE patient_id = ?";
      PreparedStatement stmt = con.prepareStatement(sql);

      stmt.setString(1, phone);
      stmt.setString(2, address);
      stmt.setInt(3, patientId);

      int rows = stmt.executeUpdate();
      if (rows > 0) {
         System.out.println("Patient updated successfully!");
      } else {
         System.out.println("Patient ID not found!");
      }
```

```
    } catch (SQLException e) {
        System.out.println("Error while updating patient: " + e.getMessage());
    }
}

public void deletePatient(Connection con, Scanner sc) {
    try {
        System.out.print("Enter Patient ID to delete: ");
        int patientId = Integer.parseInt(sc.nextLine());

        String sql = "DELETE FROM patient WHERE patient_id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, patientId);

        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Patient deleted successfully!");
        } else {
            System.out.println("Patient ID not found!");
        }
    } catch (SQLException e) {
        System.out.println("Error while deleting patient: " + e.getMessage());
    }
}

public void viewPatientById(Connection con, Scanner sc) {
    try {
        System.out.print("Enter Patient ID to view: ");
        int id = Integer.parseInt(sc.nextLine());
```

```
            String sql = "SELECT * FROM patient WHERE patient_id = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, id);

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                System.out.println("Patient Details:");
                System.out.println("Name: " + rs.getString("first_name") + " " + rs.getString("last_name"));
                System.out.println("Gender: " + rs.getString("gender"));
                System.out.println("DOB: " + rs.getDate("date_of_birth"));
                System.out.println("Phone: " + rs.getString("phone_number"));
                System.out.println("Email: " + rs.getString("email"));
                System.out.println("Address: " + rs.getString("address"));
                System.out.println("Blood Group: " + rs.getString("blood_group"));
                System.out.println("Registered On: " + rs.getTimestamp("registration_date"));
            } else {
                System.out.println("Patient not found!");
            }
        } catch (SQLException e) {
            System.out.println("Error while viewing patient: " + e.getMessage());
        }
    }
}
```

DOCTOR

```
package com.hospitalmanagement.hospital;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.util.Scanner;
public class Doctor {
    public void addDoctor(Connection con, Scanner sc) {
        try {
            System.out.print("Enter First Name: ");
            String firstName = sc.nextLine();

            System.out.print("Enter Last Name: ");
            String lastName = sc.nextLine();

            System.out.print("Enter Specialty: ");
            String specialty = sc.nextLine();

            System.out.print("Enter Phone Number: ");
            String phone = sc.nextLine();

            System.out.print("Enter Email: ");
            String email = sc.nextLine();

            String sql = "INSERT INTO doctor (first_name, last_name, specialty, phone_number, email,
hire_date) VALUES (?, ?, ?, ?, ?, NOW())";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setString(1, firstName);
            stmt.setString(2, lastName);
            stmt.setString(3, specialty);
            stmt.setString(4, phone);
            stmt.setString(5, email);
```

```
            int rows = stmt.executeUpdate();
            if (rows > 0) {
                System.out.println("Doctor added successfully!");
            }
        } catch (SQLException e) {
            System.out.println("Error while adding doctor: " + e.getMessage());
        }
    }
    public void updateDoctor(Connection con, Scanner sc) {
        try {
            System.out.print("Enter Doctor ID to Update: ");
            int doctorId = Integer.parseInt(sc.nextLine());

            System.out.print("Enter New Phone Number: ");
            String phone = sc.nextLine();

            System.out.print("Enter New Specialty: ");
            String specialty = sc.nextLine();

            System.out.print("Enter New Email: ");
            String email = sc.nextLine();

            String sql = "UPDATE doctor SET phone_number = ?, specialty = ?, email = ? WHERE doctor_id
= ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setString(1, phone);
            stmt.setString(2, specialty);
            stmt.setString(3, email);
            stmt.setInt(4, doctorId);
```

```
        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Doctor updated successfully!");
        } else {
            System.out.println("Doctor ID not found!");
        }
    } catch (SQLException e) {
        System.out.println("Error while updating doctor: " + e.getMessage());
    }
}

public void deleteDoctor(Connection con, Scanner sc) {
    try {
        System.out.print("Enter Doctor ID to Delete: ");
        int doctorId = Integer.parseInt(sc.nextLine());

        String sql = "DELETE FROM doctor WHERE doctor_id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, doctorId);

        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Doctor deleted successfully!");
        } else {
            System.out.println("Doctor ID not found!");
        }
    } catch (SQLException e) {
```

```
            System.out.println("Error while deleting doctor: " + e.getMessage());
        }
    }

    public void viewDoctorById(Connection con, Scanner sc) {
        try {
            System.out.print("Enter Doctor ID to View: ");
            int id = Integer.parseInt(sc.nextLine());

            String sql = "SELECT * FROM doctor WHERE doctor_id = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, id);

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                System.out.println("Doctor Details:");
                System.out.println("Name: " + rs.getString("first_name") + " " + rs.getString("last_name"));
                System.out.println("Specialty: " + rs.getString("specialty"));
                System.out.println("Phone: " + rs.getString("phone_number"));
                System.out.println("Email: " + rs.getString("email"));
                System.out.println("Hire Date: " + rs.getDate("hire_date"));
            } else {
                System.out.println("Doctor not found!");
            }
        } catch (SQLException e) {
            System.out.println("Error while viewing doctor: " + e.getMessage());
        }
    }
}
```

APPOINMENT

```
package com.hospitalmanagement.hospital;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class Appointment {
   public void addAppointment(Connection con, Scanner sc) {
      try {
         System.out.print("Enter Patient ID: ");
         int patientId = Integer.parseInt(sc.nextLine());

         System.out.print("Enter Doctor ID: ");
         int doctorId = Integer.parseInt(sc.nextLine());

         System.out.print("Enter Appointment Date and Time (yyyy-mm-dd hh:mm:ss): ");
         String appDate = sc.nextLine();

         System.out.print("Enter Reason: ");
         String reason = sc.nextLine();

         System.out.print("Enter Status (Scheduled / Completed / Cancelled): ");
         String status = sc.nextLine();

           String sql = "INSERT INTO appointment (patient_id, doctor_id, appointment_date, reason, status)
VALUES (?, ?, ?, ?, ?)";
         PreparedStatement stmt = con.prepareStatement(sql);
```

```
        stmt.setInt(1, patientId);
        stmt.setInt(2, doctorId);
        stmt.setString(3, appDate);
        stmt.setString(4, reason);
        stmt.setString(5, status);

        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Appointment added successfully!");
        }
    } catch (SQLException e) {
        System.out.println("Error while adding appointment: " + e.getMessage());
    }
}
public void updateAppointment(Connection con, Scanner sc) {
    try {
        System.out.print("Enter Appointment ID to Update: ");
        int appointmentId = Integer.parseInt(sc.nextLine());

        System.out.print("Enter New Status (Scheduled / Completed / Cancelled): ");
        String status = sc.nextLine();

        String sql = "UPDATE appointment SET status = ? WHERE appointment_id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setString(1, status);
        stmt.setInt(2, appointmentId);
        int rows = stmt.executeUpdate();
        if (rows > 0) {
```

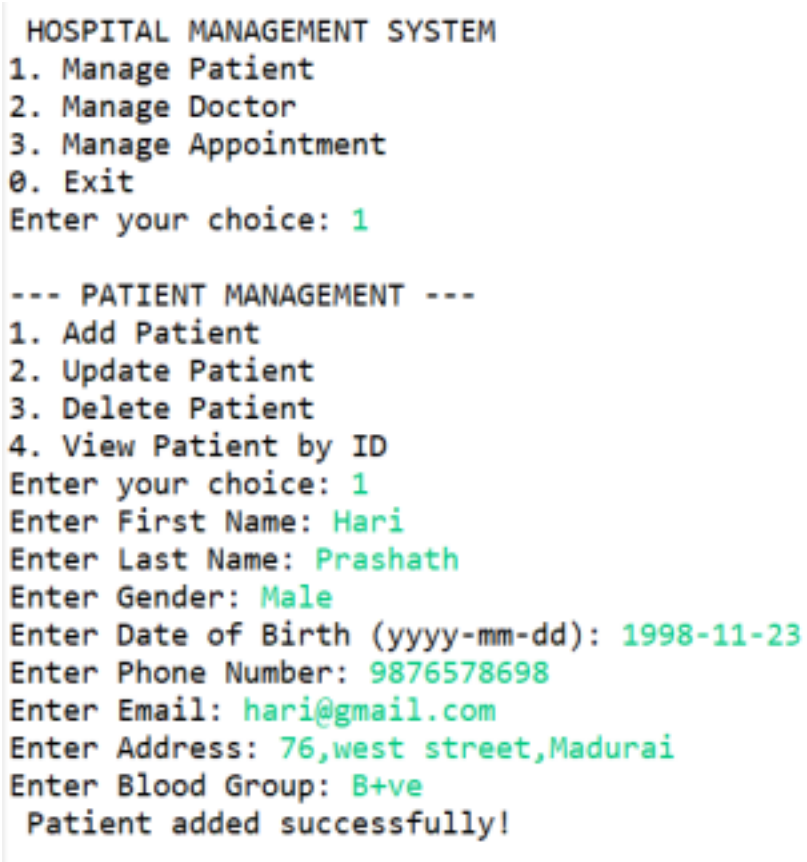
```
            System.out.println("Appointment updated successfully!");
        } else {
            System.out.println("Appointment ID not found!");
        }
    } catch (SQLException e) {
        System.out.println("Error while updating appointment: " + e.getMessage());
    }
}

public void deleteAppointment(Connection con, Scanner sc) {
    try {
        System.out.print("Enter Appointment ID to Delete: ");
        int appointmentId = Integer.parseInt(sc.nextLine());

        String sql = "DELETE FROM appointment WHERE appointment_id = ?";
        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, appointmentId)
        int rows = stmt.executeUpdate();
        if (rows > 0) {
            System.out.println("Appointment deleted successfully!");
        } else {
            System.out.println("Appointment ID not found!");
        }
    } catch (SQLException e) {
        System.out.println("Error while deleting appointment: " + e.getMessage());
    }
}
```

```
    public void viewAppointmentById(Connection con, Scanner sc) {
        try {
            System.out.print("Enter Appointment ID to View: ");
            int id = Integer.parseInt(sc.nextLine());

            String sql = "SELECT * FROM appointment WHERE appointment_id = ?";
            PreparedStatement stmt = con.prepareStatement(sql);
            stmt.setInt(1, id);

            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                System.out.println("Appointment Details:");
                System.out.println("Patient ID: " + rs.getInt("patient_id"));
                System.out.println("Doctor ID: " + rs.getInt("doctor_id"));
                System.out.println("Date: " + rs.getTimestamp("appointment_date"));
                System.out.println("Reason: " + rs.getString("reason"));
                System.out.println("Status: " + rs.getString("status"));
            } else {
                System.out.println("Appointment not found!");
            }
        } catch (SQLException e) {
            System.out.println("Error while viewing appointment: " + e.getMessage());
        }
    }
}
```

JDBConnection

```
package com.hospitalmanagement.hospital;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBConnection {
    public static Connection getConnection() throws Exception {
        Class.forName("com.mysql.cj.jdbc.Driver");
        return DriverManager.getConnection("jdbc:mysql://localhost:3306/hospital", "root", "password");
    }
}
```

SCREEN SHOTS



```
--- PATIENT MANAGEMENT ---
1. Add Patient
2. Update Patient
3. Delete Patient
4. View Patient by ID
Enter your choice: 4
Enter Patient ID to View: 3
 Patient Details:
Name: Mary Smith
Gender: Female
DOB: 1998-09-25
Phone: 9143657298
Email: mary@gmail.com
Address: 37,West Street,Bangalore
Blood Group: A+ve
Registered On: 2025-05-11 11:41:06.0


 HOSPITAL MANAGEMENT SYSTEM
1. Manage Patient
2. Manage Doctor
3. Manage Appointment
0. Exit
Enter your choice: 2

--- DOCTOR MANAGEMENT ---
1. Add Doctor
2. Update Doctor
3. Delete Doctor
4. View Doctor by ID
Enter your choice: 1
Enter First Name: Aravinth
Enter Last Name: Samy
Enter Specialty: Dermatologist
Enter Phone Number: 6365645323
Enter Email: aravinth@gmail.com
☑ Doctor added successfully!
```

```
--- DOCTOR MANAGEMENT ---
1. Add Doctor
2. Update Doctor
3. Delete Doctor
4. View Doctor by ID
Enter your choice: 4
Enter Doctor ID to View: 1
 Doctor Details:
Name: Sathiya Jothi
Specialty: Cardiologist
Phone: 9944749850
Email: sathiya@gmail.com
Hire Date: 2012-03-15
```

```
 HOSPITAL MANAGEMENT SYSTEM
1. Manage Patient
2. Manage Doctor
3. Manage Appointment
0. Exit
Enter your choice: 3

--- APPOINTMENT MANAGEMENT ---
1. Add Appointment
2. Update Appointment
3. Delete Appointment
4. View Appointment by ID
Enter your choice: 1
Enter Patient ID: 4
Enter Doctor ID: 3
Enter Appointment Date and Time (yyyy-mm-dd hh:mm:ss): 2025-05-12
Enter Reason: Weekly checkup
Enter Status (Scheduled/Completed/Cancelled): Scheduled
 Appointment added successfully!
```

```
 HOSPITAL MANAGEMENT SYSTEM
1. Manage Patient
2. Manage Doctor
3. Manage Appointment
0. Exit
Enter your choice: 3

--- APPOINTMENT MANAGEMENT ---
1. Add Appointment
2. Update Appointment
3. Delete Appointment
4. View Appointment by ID
Enter your choice: 4
Enter Appointment ID to View: 3
 Appointment Details:
Patient ID: 4
Doctor ID: 3
Date: 2025-05-12
Reason: Weekly checkup
Status: Scheduled
```

```
mysql> select * from appointment;
+----------------+------------+-----------+---------------------+-----------------------+-----------+
| appointment_id | patient_id | doctor_id | appointment_date    | reason                | status    |
+----------------+------------+-----------+---------------------+-----------------------+-----------+
|              1 |          1 |         1 | 2025-05-11 11:20:00 | health checkup        | Completed |
|              2 |          3 |         2 | 2025-05-11 12:00:00 | Migraine Consultation | Completed |
|              3 |          4 |         3 | 2025-05-12 00:00:00 | Weekly checkup        | Scheduled |
+----------------+------------+-----------+---------------------+-----------------------+-----------+
3 rows in set (0.01 sec)
```

```
mysql> select * from doctor;
+-----------+------------+-----------+----------------+--------------+--------------------+------------+
| doctor_id | first_name | last_name | specialty      | phone_number | email              | hire_date  |
+-----------+------------+-----------+----------------+--------------+--------------------+------------+
|         1 | Sathiya    | Jothi     | Cardiologist   | 9944700850   | sathiya@gmail.com  | 2012-03-15 |
|         2 | Arun       | Kumar     | Neurologist    | 9787836276   | arun@gmail.com     | 2025-05-11 |
|         3 | Aravinth   | Samy      | Dermatologist  | 6365645323   | aravinth@gmail.com | 2025-05-11 |
+-----------+------------+-----------+----------------+--------------+--------------------+------------+
3 rows in set (0.00 sec)
```

```
mysql> select * from patient;
+------------+------------+-----------+--------+---------------+--------------+-----------------+-----------------------------+-------------+---------------------+
| patient_id | first_name | last_name | gender | date_of_birth | phone_number | email           | address                     | blood_group | registration_date   |
+------------+------------+-----------+--------+---------------+--------------+-----------------+-----------------------------+-------------+---------------------+
|          1 | Jhon       | Doe       | Male   | 2003-03-01    | 7868831981   | jhon@gmail.com  | 47,North Street,Erode       | A+ve        | 2025-05-10 21:36:56 |
|          3 | Mary       | Smith     | Female | 1998-09-15    | 9143697298   | mary@gmail.com  | 37,West Street,Bangalore    | A+ve        | 2025-05-11 11:41:06 |
|          4 | Hari       | Prashath  | Male   | 1998-11-23    | 9876570698   | hari@gmail.com  | 76,west street,Madurai      | B+ve        | 2025-05-11 19:01:22 |
+------------+------------+-----------+--------+---------------+--------------+-----------------+-----------------------------+-------------+---------------------+
```

CONCLUSION

In conclusion, Java's JDBC (Java Database Connectivity) serves as the backbone of the Hospital Management System by providing a robust interface for seamless communication between the application and the database. It ensures that patient, doctor, and appointment data is efficiently stored, retrieved, and updated in real-time, thereby enabling smooth operation of the hospital's daily activities. The system guarantees data integrity, eliminates redundancy, and minimizes the risk of manual errors, making it a reliable tool for healthcare administration.

Furthermore, the application's use of structured exception handling enhances its stability by gracefully managing potential errors such as database connectivity issues, invalid user inputs, or system failures. This proactive approach to error management ensures that the system continues to function reliably, providing users with informative error messages and minimizing disruption. As a result, the Hospital Management System becomes a trustworthy and resilient solution, fostering improved operational efficiency and user satisfaction.