**Hospital Management System  
 Documentation**

### **1. Project Overview**

The **Hospital Management System** is a Java-based application that allows hospitals to manage patient appointments with doctors. The system follows an object-oriented approach and interacts with a MySQL database. The project uses JDBC for database interaction, implementing CRUD (Create, Read, Update, Delete) functionalities for appointments.

**2. Documentation for Each Directory and Module**

**2.1. entity**

This directory contains the entity classes that represent the tables in the database. Each entity class corresponds to a table and contains attributes that match the table columns.

1. **Patient.java**
   * Represents a patient in the hospital.
   * Attributes:
     + patientId: Unique ID for each patient (Primary Key).
     + firstName: First name of the patient.
     + lastName: Last name of the patient.
     + dateOfBirth: Date of birth of the patient.
     + gender: Gender of the patient.
     + contactNumber: Contact number of the patient.
     + address: Address of the patient.
   * Methods:
     + Getters and setters for each attribute.
     + toString() method to print patient details.
2. **Doctor.java**
   * Represents a doctor in the hospital.
   * Attributes:
     + doctorId: Unique ID for each doctor (Primary Key).
     + firstName: First name of the doctor.
     + lastName: Last name of the doctor.
     + specialization: Doctor's specialization field.
     + contactNumber: Doctor's contact number.
   * Methods:
     + Getters and setters for each attribute.
     + toString() method to print doctor details.
3. **Appointment.java**
   * Represents an appointment between a patient and a doctor.
   * Attributes:
     + appointmentId: Unique ID for the appointment (Primary Key).
     + patientId: ID of the patient associated with the appointment (Foreign Key).
     + doctorId: ID of the doctor associated with the appointment (Foreign Key).
     + appointmentDate: Date of the appointment.
     + description: Description of the appointment.
   * Methods:
     + Getters and setters for each attribute.
     + toString() method to print appointment details.

**2.2. dao**

This directory contains the Data Access Object (DAO) pattern classes. The IHospitalService interface defines the core business logic, and HospitalServiceImpl implements the interface for interacting with the database.

1. **IHospitalService.java**
   * Defines the service interface for managing hospital appointments.
   * Methods:
     + getAppointmentById(int appointmentId): Fetches an appointment by its ID.
     + getAppointmentsForPatient(int patientId): Fetches all appointments for a given patient.
     + getAppointmentsForDoctor(int doctorId): Fetches all appointments for a given doctor.
     + scheduleAppointment(Appointment appointment): Schedules a new appointment.
     + updateAppointment(Appointment appointment): Updates an existing appointment.
     + cancelAppointment(int appointmentId): Cancels an appointment by its ID.
2. **HospitalServiceImpl.java**
   * Implements the IHospitalService interface.
   * Performs actual database interaction using JDBC to manage appointments (CRUD operations).
   * Methods:
     + Implements all methods defined in IHospitalService to interact with the database, like scheduling, updating, and deleting appointments.

**2.3. exception**

Contains custom exceptions to handle specific errors related to the application.

1. **PatientNumberNotFoundException.java**
   * A custom exception that is thrown when an invalid patient ID is used.
   * Inherits from Exception class and provides an error message for debugging.

**2.4. util**

Contains utility classes for database connection management.

1. **DBConnectionUtil.java**
   * Manages the database connection using JDBC.
   * Methods:
     + getConnection(): Establishes and returns a connection to the MySQL database using the details provided in the db.properties file.
     + Uses the DriverManager class to manage connections.
2. **DBPropertyUtil.java**
   * Loads database configuration details from the db.properties file.
   * Methods:
     + getPropertyString(String fileName): Reads the db.properties file and returns the connection string required for database connection.

**2.5. main**

Contains the main class that runs the entire application and provides a user interface to interact with the system.

1. **MainModule.java**
   * Contains the main method that drives the program using a menu-driven interface.
   * Features:
     + The user can:
       - Schedule a new appointment.
       - View an appointment by ID.
       - View all appointments for a patient or a doctor.
       - Update or cancel an appointment.
     + Calls the methods from HospitalServiceImpl to perform the requested operations.
     + Includes input validation and error handling.

**2.6. resources**

Contains configuration files for the application.

1. **db.properties**
   * Stores the database configuration such as hostname, database name, username, and password.
   * The application uses this file to establish a database connection.
   * Example:

hostname=localhost

dbname=hospitaldb

username=root

password=yourpassword

portnumber=3306

connectionString=jdbc:mysql://${hostname}:${portnumber}/${dbname}

**Database Documentation**

In this section, we will document the database schema used in the **Hospital Management System**. The database manages entities like **patients**, **doctors**, and their **appointments**. Below, we'll explain the structure of each table, their attributes, and the relationships between them.

**1. Database Overview**

* **Database Name**: hospitaldb
* **Tables**:
  1. patients
  2. doctors
  3. appointments

This database schema is designed to store information about patients, doctors, and the appointments between them. It ensures data consistency through the use of **foreign keys** to define relationships between the tables.

**2. Tables and Relationships**

**2.1. patients Table**

The patients table stores information about each patient in the hospital. It has the following columns:

A screenshot of a computer

Description automatically generated

 **Primary Key**: patientId

 **Purpose**: Stores personal details of patients in the hospital.

**2.2. doctors Table**

The doctors table stores information about the doctors available in the hospital. It contains the following columns:

A screenshot of a computer

Description automatically generated

 **Primary Key**: doctorId

 **Purpose**: Stores information about the doctors, including their specialization and contact information.

**2.3. appointments Table**

The appointments table stores information about appointments between patients and doctors. It also defines the relationships between the patients and doctors tables using **foreign keys**.

A screenshot of a computer

Description automatically generated

* **Primary Key**: appointmentId
* **Foreign Keys**:
  + patientId references the patients table.
  + doctorId references the doctors table.
* **Purpose**: Manages the appointments between patients and doctors, linking the two entities through foreign key relationships.

**3. Relationships Between Tables**

The appointments table acts as the **junction** between the patients and doctors tables, creating a **one-to-many relationship** between patients and doctors through appointments. Here’s how the relationships work:

1. **One-to-Many Relationship Between patients and appointments**:
   * A patient can have **multiple appointments**.
   * Each appointment belongs to **one patient**, identified by the patientId foreign key in the appointments table.
2. **One-to-Many Relationship Between doctors and appointments**:
   * A doctor can have **multiple appointments**.
   * Each appointment belongs to **one doctor**, identified by the doctorId foreign key in the appointments table.

**Relationship Diagram (ER Diagram) Overview:**

* **Entities**: patients, doctors, appointments
* **Foreign Keys**:
  + patientId in appointments references patients(patientId).
  + doctorId in appointments references doctors(doctorId).

**Methods and Functionality Documentation**

**1. entity/ Directory Documentation**

This directory contains entity classes, each representing a database table. These classes encapsulate the attributes and methods related to the entities and mirror the structure of the tables in the database.

**1.1. Patient.java**

* **Purpose**: Represents a patient in the hospital.
* **Attributes**:
  + private int patientId: Unique identifier for each patient (primary key).
  + private String firstName: Patient’s first name.
  + private String lastName: Patient’s last name.
  + private String dateOfBirth: Date of birth of the patient.
  + private String gender: Patient’s gender.
  + private String contactNumber: Contact number of the patient.
  + private String address: Patient’s residential address.
* **Methods**:
  + **Constructors**:
    - Default constructor.
    - Parametrized constructor to initialize all attributes.
  + **Getters and Setters**: For each attribute to encapsulate and expose the data safely.
  + **toString() Method**: Returns a string representation of the patient’s details, used when printing or logging patient information.

**1.2. Doctor.java**

* **Purpose**: Represents a doctor in the hospital.
* **Attributes**:
  + private int doctorId: Unique identifier for each doctor (primary key).
  + private String firstName: Doctor’s first name.
  + private String lastName: Doctor’s last name.
  + private String specialization: Doctor’s medical specialty (e.g., cardiology, pediatrics).
  + private String contactNumber: Doctor’s contact number.
* **Methods**:
  + **Constructors**:
    - Default constructor.
    - Parametrized constructor to initialize all attributes.
  + **Getters and Setters**: For each attribute.
  + **toString() Method**: Returns a string representation of the doctor’s details.

**1.3. Appointment.java**

* **Purpose**: Represents an appointment between a patient and a doctor.
* **Attributes**:
  + private int appointmentId: Unique identifier for each appointment (primary key).
  + private int patientId: Identifier of the patient involved in the appointment (foreign key).
  + private int doctorId: Identifier of the doctor involved in the appointment (foreign key).
  + private String appointmentDate: The date of the appointment.
  + private String description: Brief description of the appointment or medical reason.
* **Methods**:
  + **Constructors**:
    - Default constructor.
    - Parametrized constructor to initialize all attributes.
  + **Getters and Setters**: For each attribute.
  + **toString() Method**: Returns a string representation of the appointment details.

**2. dao/ Directory Documentation**

This directory contains the service interface and its implementation to manage appointments, following the **Data Access Object (DAO)** design pattern. These classes handle all interactions with the database.

**2.1. IHospitalService.java**

* **Purpose**: Defines the interface that outlines the methods for interacting with the appointments data in the database.
* **Methods**:
  + **Appointment getAppointmentById(int appointmentId)**: Fetches an appointment from the database by its ID.
  + **List<Appointment> getAppointmentsForPatient(int patientId)**: Retrieves all appointments for a given patient.
  + **List<Appointment> getAppointmentsForDoctor(int doctorId)**: Retrieves all appointments for a given doctor.
  + **boolean scheduleAppointment(Appointment appointment)**: Schedules a new appointment.
  + **boolean updateAppointment(Appointment appointment)**: Updates an existing appointment.
  + **boolean cancelAppointment(int appointmentId)**: Cancels an appointment by its ID.

**2.2. HospitalServiceImpl.java**

* **Purpose**: Implements the IHospitalService interface. It handles the actual interaction with the database to perform CRUD (Create, Read, Update, Delete) operations on the appointment data.
* **Methods**:
  + **Appointment getAppointmentById(int appointmentId)**: Retrieves an appointment by its ID using a SQL SELECT query.
  + **List<Appointment> getAppointmentsForPatient(int patientId)**: Retrieves all appointments for a given patient using a SQL SELECT query.
  + **List<Appointment> getAppointmentsForDoctor(int doctorId)**: Retrieves all appointments for a given doctor using a SQL SELECT query.
  + **boolean scheduleAppointment(Appointment appointment)**: Inserts a new appointment into the database using a SQL INSERT query.
  + **boolean updateAppointment(Appointment appointment)**: Updates an existing appointment in the database using a SQL UPDATE query.
  + **boolean cancelAppointment(int appointmentId)**: Deletes an appointment from the database using a SQL DELETE query.

**3. exception/ Directory Documentation**

This directory contains custom exceptions designed to handle specific error cases within the application.

**3.1. PatientNumberNotFoundException.java**

* **Purpose**: Custom exception that is thrown when a patient number provided by the user does not exist in the database.
* **Attributes**:
  + Inherits from Java’s Exception class.
* **Methods**:
  + **Constructor**: Takes a custom error message that is displayed when the exception is thrown, making it easier to debug patient-related errors.

**4. util/ Directory Documentation**

The util/ directory contains utility classes that help manage database connections and read configuration settings from external files.

**4.1. DBConnectionUtil.java**

* **Purpose**: Manages the database connection using the JDBC (Java Database Connectivity) API.
* **Methods**:
  + **Connection getConnection()**:
    - Loads the database properties from the db.properties file using DBPropertyUtil.
    - Establishes and returns a connection to the MySQL database using DriverManager.getConnection().
    - If the connection already exists, it returns the existing connection (singleton pattern).
  + Handles SQL and IO exceptions to ensure proper error reporting if the connection fails.

**4.2. DBPropertyUtil.java**

* **Purpose**: Reads the database connection details from the db.properties file.
* **Methods**:
  + **String getPropertyString(String fileName)**:
    - Reads the properties file and extracts the connection string, username, password, and other relevant information required to connect to the database.
    - Returns the connection string to be used by DBConnectionUtil for establishing a connection.

**5. main/ Directory Documentation**

This directory contains the main class that runs the application. It provides a user interface to interact with the system and allows users to perform various operations on appointments.

**5.1. MainModule.java**

* **Purpose**: Provides a menu-driven interface for users to interact with the system, manage appointments, and perform CRUD operations.
* **Features**:
  + **Menu System**: The main method presents a menu to the user, allowing them to:
    - Schedule a new appointment.
    - View an appointment by ID.
    - View all appointments for a specific patient or doctor.
    - Update an existing appointment.
    - Cancel an appointment.
  + **Interaction with HospitalServiceImpl**: Each menu option corresponds to a method call in HospitalServiceImpl, which performs the desired operation in the database.
* **Methods**:
  + **scheduleAppointment(Scanner sc)**: Allows the user to input appointment details and schedule a new appointment.
  + **viewAppointmentById(Scanner sc)**: Fetches and displays details of an appointment based on the ID.
  + **viewAppointmentsForPatient(Scanner sc)**: Fetches and displays all appointments for a specified patient.
  + **viewAppointmentsForDoctor(Scanner sc)**: Fetches and displays all appointments for a specified doctor.
  + **updateAppointment(Scanner sc)**: Allows the user to modify an existing appointment.
  + **cancelAppointment(Scanner sc)**: Cancels an appointment by its ID.

**6. resources/ Directory Documentation**

This directory contains configuration files that provide external settings for the application, such as the database connection details.

**6.1. db.properties**

* **Purpose**: Stores the configuration settings for connecting to the MySQL database.
* **Fields**:
  + **hostname**: The address of the database server (usually localhost for local databases).
  + **dbname**: The name of the database to connect to (hospitaldb).
  + **username**: The username used to authenticate with the database (e.g., root).
  + **password**: The password for the specified database user.
  + **portnumber**: The port number on which the MySQL server is running (usually 3306).
  + **connectionString**: A formatted string that includes the hostname, port, and database name, used to establish the connection.

**Conclusion**

This **Hospital Management System** project showcases an object-oriented approach to managing appointments in a hospital setting. It demonstrates how Java applications can interact with a MySQL database using JDBC, while following standard design patterns such as DAO (Data Access Object) and proper error handling.