**Hospital Management System**



Apollo hospital is the leading multi-specialty hospital in India. Patient and doctor details are maintained manually as the transactions are very huge and the hospital wants to automate the system.

**Requirements for the Hospital Management System:**

**1) Inpatient Registration Statement:**

The system allows the registration of inpatients, capturing their personal details, medical history, and emergency contacts.

Upon registration, a unique inpatient ID is generated for easy identification and tracking.

**2) Outpatient Registration Statement:**

The system facilitates the registration of outpatients, collecting their personal details, contact information, and reason for visit.

A unique outpatient ID is generated for each registration to ensure efficient record-keeping and future reference.

**3) Doctor Registration Statement:**

The system enables the registration of doctors, capturing their professional details, specialization, contact information, and availability.

Each doctor is assigned a unique doctor ID for identification and scheduling purposes.

**4)Appointment Scheduling for Outpatient Statement:**

Outpatients can request appointments with doctors based on their availability and specialization. The system allows staff members to schedule appointments by selecting the desired doctor, preferred date, and time slot.

**5) Allocation for Inpatient Statement:**

When an inpatient is registered, the system manages the allocation of rooms or beds based on availability, preferences, and medical requirements.

Room allocation may consider factors such as room type and patient preferences.

**6) Payment Statement:**

The system handles payment processes for both inpatients and outpatients.It generates bills for services, treatments, medications, and other charges incurred during the patient's stay or visit.The system provides options for payment modes, such as cash, credit/debit cards, or insurance claims.

**Required Software and Modules:**

***Eclipse IDE***: An integrated development environment for software development.

***Xampp/MySQL***: A software stack for local web development, including the MySQL database.

***Required Jar files***: MySQL connector based on the MySQL database version.

**Modules Required:**

The application should consist of 6 modules

1. OutPatient Module
2. InPatient Module
3. Doctor Module
4. Appointment Module
5. Allocation Module
6. Payment Module

**Tables Required:**

1. **For OutPatient: Create a table “outpatient”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| PATIENT\_ID | VARCHAR (25) primary **key** |
| PATIENT\_NAME | VARCHAR (25) |
| PHONE\_NUMBER | BIGINT |
| AGE | INT |
| GENDER | VARCHAR (50) |
| MEDICAL\_HISTORY | VARCHAR (50) |
| PREFFERED\_SPECIALIST | VARCHAR (50) |
| MEDICINE\_FEE | DOUBLE |
| PATIENT\_TYPE | VARCHAR (50) |
| REGISTRATION\_FEES | DOUBLE |

1. **For InPatient: Create a table “inpatient”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| PATIENT\_ID | VARCHAR (25) primary **key** |
| PATIENT\_NAME | VARCHAR (25) |
| PHONE\_NUMBER | BIGINT |
| AGE | INT |
| GENDER | VARCHAR (50) |
| MEDICAL\_HISTORY | VARCHAR (50) |
| PREFFERED\_SPECIALIST | VARCHAR (50) |
| MEDICINE\_FEE | DOUBLE |
| PATIENT\_TYPE | VARCHAR (50) |
| ADMISSION\_FEES | DOUBLE |
| TREATMENT | VARCHAR (50) |
| ROOM\_TYPE | VARCHAR (50) |
| WANT\_FOOD | VARCHAR (50) |

1. **Doctor Table: Create a table “doctor”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| DOCTOR\_ID | VARCHAR (25) primarykey |
| DOCTOR\_NAME | VARCHAR (25) |
| DOCTOR\_FEE | DOUBLE |
| SPECIALIZATION | VARCHAR (25) |
| AVAILABLE\_DATE | DATE |
| AVAILABLE\_TIME | TIME |

1. **Appointment Table: Create a table “appointment”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| APPOINTMENT\_ID | VARCHAR (25) primary key |
| PATIENT\_ID | VARCHAR (25) foreign key |
| DOCTOR\_ID | VARCHAR (25) foreign key |
| DOCTOR\_NAME | VARCHAR (25) |
| SPECIALIZATION | VARCHAR (25) |
| APPOINTMENT\_DATE | DATE |
| APPOINTMENT\_TIME | TIME |
| MODE\_OF\_APPOINTMENT | VARCHAR (25) |

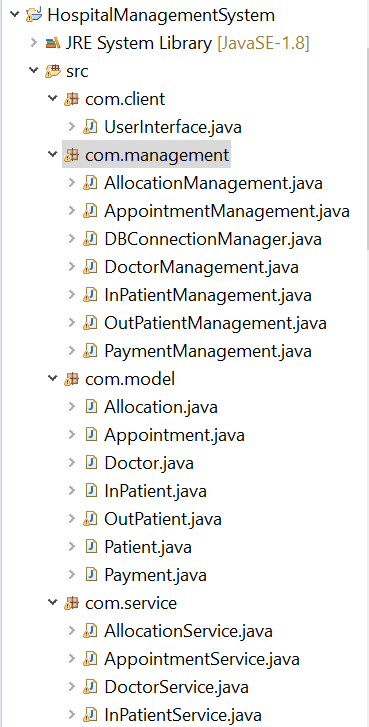
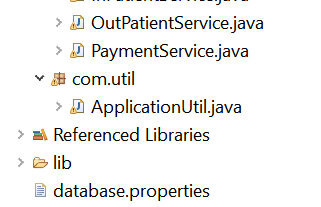
1. **Allocation Table: Create a table “allocation”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| ALLOCATION\_ID | VARCHAR (25) primarykey |
| PATIENT\_ID | VARCHAR (25) foreign key |
| ROOM\_NUMBER | INT |
| NO\_OF\_DAYS\_ADMITTED | INT |
| ADMISSION\_DATE | DATE |
| DISCHARGE\_DATE | DATE |
| TREATMENT | VARCHAR (25) |
| ROOM\_TYPE | VARCHAR (25) |
| WANT\_FOOD | VARCHAR (25) |

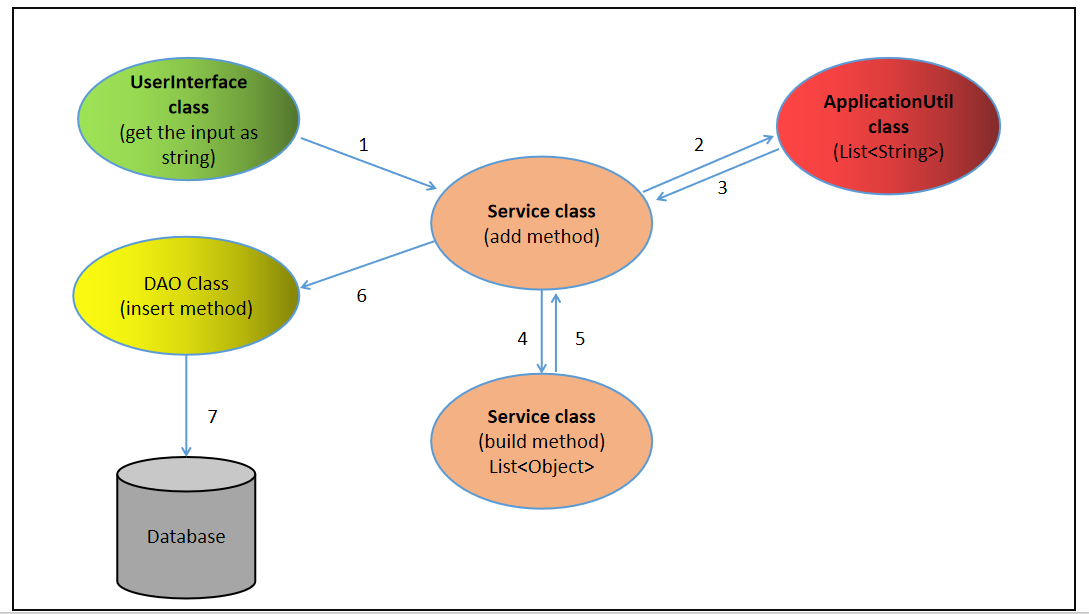
1. **Payment Table: Create a table “payment”**

|  |  |
| --- | --- |
| **Column Name** | **Data Type** |
| PAYMENT\_ID | VARCHAR (25) primarykey |
| PATIENT\_ID | VARCHAR (25) foreign key |
| PATIENT\_NAME | VARCHAR (25) |
| PATIENT\_TYPE | VARCHAR (25) |
| PAYMENT\_DATE | DATE |
| MODE\_OF\_PAYMENT | VARCHAR (25) |
| BILL\_AMOUNT | DOUBLE |

**Project Structure:**



**Project Flow:**



1. Receive string input in the user interface and pass it to the service class's "add" method.
2. In the "add" method of the service class, forward the input string to the applicationUtil class, extracting it to a List<String>, then return it to the "add" method.
3. In the "add" method of the service class, pass the List<String> to the "build" method to convert records into List<Object>, then return the result to the "add" method.
4. Within the "add" method of the service class, pass the list of objects to the "insert" method in the DAO class to insert the records into the database.

****Note:****

1) Column names are the attribute names for the respective POJO class (should be in camelCase). The access specifier for the attributes should be private.

Ex: For the Pojo class **“*InPatient*”**

The attributes should be written as:

private String patientId;

private String patientName;

2) The input should be a string separated by the delimiter.

Ex: InPatient details-

“Medona:990889789:17:Female:Kneereplacement:Orthopaedist:160800:inpatient:500:surgery:Semi-private:yes”.

3) Please follow the naming convention properly for class and method names.

**Scope of the service modules:**

|  |  |
| --- | --- |
| **Module** | **Responsibilities** |
| **Appointment Service** | * **Build Appointment List:** Generates a list of Appointment objects for a given patient, doctor's specialization, and mode of appointment. * **Add Appointment Details:** Inserts appointment records into the database based on patient ID, doctor specialization, and mode of appointment. * **Generate ID:** Creates a unique appointment ID using a specific prefix and an incremented count. * **Retrieve Appointment Details From DB:** Fetches appointment details from the database based on a patient ID. * **Delete Appointment Details From DB:** Deletes appointment records from the database for a specific patient ID. |
| **Allocation Service** | * **Build Allocation List**: Generates a list of Allocation objects based on in-patient details, including allocation ID, patient ID, room number, admission and discharge dates, treatment, room type, and food preference. * **Add Allocation Details**: Adds allocation records into the database using in-patient details. * **Generate ID**: Generates a unique allocation ID using a prefix and an incremented count. * **Retrieve Allocation Details From DB**: Retrieves allocation details from the database based on patient ID, showing allocation ID, patient ID, room number, days admitted, admission and discharge dates, treatment, room type, and food preference. * **Delete Allocation Details From DB**: Removes allocation details from the database based on patient ID. |
| **Docter Service** | * **Build Doctor List:** Creates a list of Doctor objects from a set of string records, including doctor ID, name, fee, specialization, available date, and time. * **Add Doctor List:** Inserts doctor records into the database based on a string array containing doctor details. * **Update Doctor Fee:** Modifies the doctor's fee in the database corresponding to the provided doctor ID. * **Update Doctor Available Date**: Updates the available date of a doctor in the database based on the given doctor ID and date. * **Retrieve Doctor Details:** Retrieves doctor details from the database based on a doctor ID, presenting doctor name, fee, specialization, available date, and time. |
| **Inpatient Service** | * Build InPatient List: Creates a list of InPatient objects from a list of string records, including patient ID, name, phone number, age, gender, medical history, specialist preference, medicine fee, patient type, admission fee, treatment, room type, and food preference. * Add InPatient List: Inserts in-patient records into the database based on a string array containing in-patient details. * Update Inpatient Phone Number: Modifies the phone number of an in-patient in the database based on the given patient ID. * Update Inpatient Room Type: Updates the room type of an in-patient in the database using the provided patient ID and room type. * Update Inpatient Food Preference: Modifies the food preference of an in-patient in the database for a specific patient ID. * Retrieve Inpatient Details: Fetches in-patient details from the database based on a patient ID, presenting patient's name, phone number, age, gender, medical history, specialist preference, medicine fee, patient type, admission fee, treatment, room type, and food preference. * Delete Inpatient Details From DB: Removes in-patient details from the database for a given |
| **Outpatient Service** | * **BuildOutPatientList:** Generates a list of OutPatient objects using a list of string records, including patient ID, name, phone number, age, gender, medical history, specialist preference, medicine fee, patient type, and registration fees. * **Generate ID**: Creates a unique outpatient ID using a specific prefix and an incremented count. * **Add OutPatient List:** Inserts outpatient records into the database based on a string array containing outpatient details. * **Update OutPatient Phone Number:** Modifies the phone number of an outpatient in the database using the given patient ID. * **Retrieve OutPatient Details:** Fetches outpatient details from the database based on a patient ID, presenting patient's name, phone number, age, gender, medical history, specialist preference, medicine fee, patient type, and registration fees. * **Delete Outpatient Details From DB:** Removes outpatient details from the database for a given patient ID. |
| **Payment Service** | * **Build Payment List:** Generates a list of Payment objects with details like patient ID, name, patient type (InPatient/OutPatient), payment date, and mode of payment. * **Add Payment Details:** Inserts payment records into the database based on patient ID, patient type, payment date, and mode of payment. * **Generate ID:** Creates a unique payment ID using a specific prefix and an incremented count. * **Calculate Bill Amount for OutPatients:** Calculates the bill amount for OutPatients based on details like doctor fees, medicine fees, and registration fees. * **Calculate Bill Amount for InPatient:** Calculates the bill amount for InPatients, considering factors like treatment, room type, food preference, and admission duration. * **Retrieve Payment Details From DB:** Fetches payment details from the database based on a patient ID, presenting payment ID, patient details, payment date, mode of payment, and the bill amount to be paid. |

**Scope of the management modules:**

|  |  |
| --- | --- |
| **Allocation Management** | * **insertAllocationList:** Inserts a list of **Allocation** records into the database using batch processing. * **checkIdExists:** Checks if a specific allocation ID exists in the database. * **retrieveAllocationDetailsFromDB:** Retrieves a list of **Allocation** details based on the patient ID. * **deleteAllocationDetailsFromDB:** Deletes allocation details from the database for a specific patient ID. |
| **DBConnectionManager** | * Establishes a connection to the database using the properties specified in the **database.properties** file. * Loads the database properties (driver name, URL, username, and password) from the file. * Reads a properties file named **database.properties** using a **FileInputStream**. * Retrieves the necessary properties and then registers the driver using **Class.forName**. * Uses the **DriverManager.getConnection** method to create a connection with the database specified in the properties file. * Returns the established database connection. |
| **Appointment Management** | * **insertAppointmentList:** Inserts a list of appointment records into the database, employing batch processing for efficiency. * **retrieveDoctorDetailsForAppointment:** Retrieves a list of doctors based on the specialization. * **checkIdExists:** Checks if a specific appointment ID exists in the database. * **retrieveAppointmentDetailsFromDB:** Retrieves a list of appointment details based on the patient ID. * **deleteAppointmentDetailsFromDB:** Deletes appointment details from the database for a specific patient ID. |
| **Doctor Management** | * **insertDoctorList:** Inserts a list of doctor records into the database, utilizing batch processing for efficiency. * **updateDoctorFee:** Updates the doctor's fee in the database based on the provided doctor ID. * **updateDoctorAvailableDate:** Updates the available date for a doctor in the database based on the provided doctor ID. * **retrieveDoctorDetails:** Retrieves doctor details from the database based on the provided doctor ID. |
| **RIResident Management** | * **insertInPatientList:**Inserts a list of in-patient records into the database, using batch processing to improve efficiency. * **checkIdExists:**Checks whether an in-patient ID exists in the database. * **updateInPatientPhoneNumber:**Updates the phone number of an in-patient based on the provided ID. * **updateInPatientRoomType:**Updates the room type of in-patient based on the provided ID. * **updateInpatientFoodPreference:**Updates the food preference of an in-patient based on the provided ID. * **retrieveInPatientList:**Retrieves in-patient details from the database based on the provided in-patient ID. |
| **Payment Management** | * **insertPaymentList:** Inserts a list of payment records into the database, using batch processing to enhance efficiency. * **checkIdExists:** Checks whether a payment ID exists in the database. * **retrievePaymentDetailsFromDB:** Retrieves payment details from the database based on the provided patient ID. |
| **Room Management** | * **insertOutPatientList:**Inserts a list of OutPatient records into the database, using batch processing for better performance. * **checkIdExists:**Checks if a given patient ID exists in the database for OutPatients. * **updateOutPatientPhoneNumber:**Updates the phone number for a specific OutPatient. * **retrieveOutPatientList:**Retrieves OutPatient details from the database based on the provided patient ID. * **deleteOutpatientDetailsFromDB:**Deletes OutPatient details from the database based on the provided patient ID. |