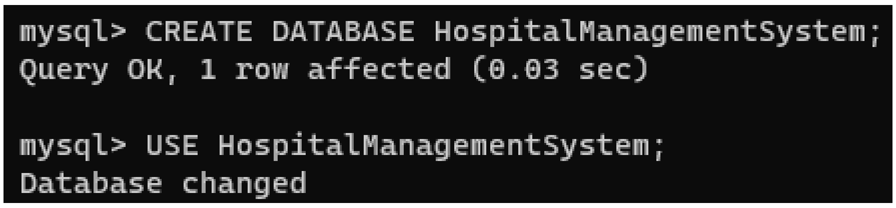
**Rushalee Das**

**Coding Challenge: Hospital Management System**

**Problem Statement:**

1**Create SQL Schema from the following classes class, use the class attributes for table column names.**

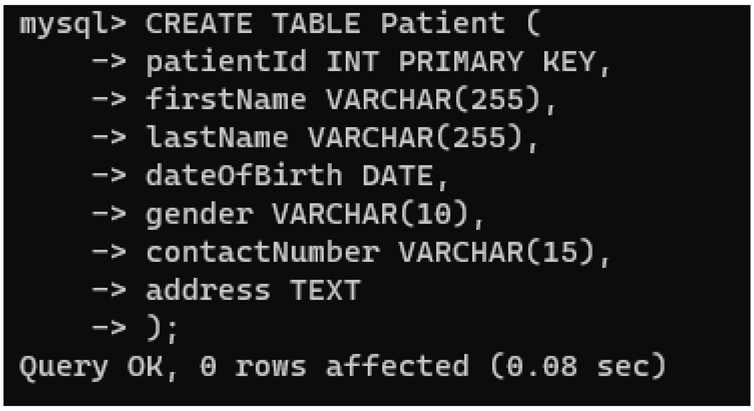


```
mysql> CREATE DATABASE HospitalManagementSystem;
Query OK, 1 row affected (0.03 sec)

mysql> USE HospitalManagementSystem;
Database changed
```

1. Create the following **model/entity classes** within package **entity** with variables declared private, constructors(default and parametrized,getters,setters and toString())

1. Define **`Patient`** class with the following confidential attributes:

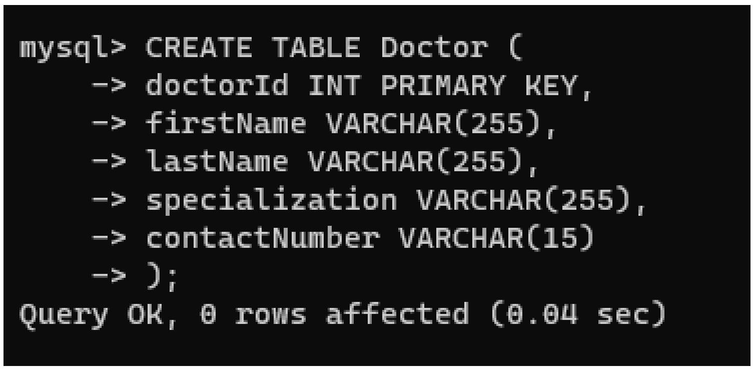
a. patientId b. firstName  c. lastName; d. dateOfBirth e. gender f. contactNumber g. address;



```
mysql> CREATE TABLE Patient (
    -> patientId INT PRIMARY KEY,
    -> firstName VARCHAR(255),
    -> lastName VARCHAR(255),
    -> dateOfBirth DATE,
    -> gender VARCHAR(10),
    -> contactNumber VARCHAR(15),
    -> address TEXT
    -> );
Query OK, 0 rows affected (0.08 sec)
```

**2.** Define **'Doctor`** class with the following confidential attributes:

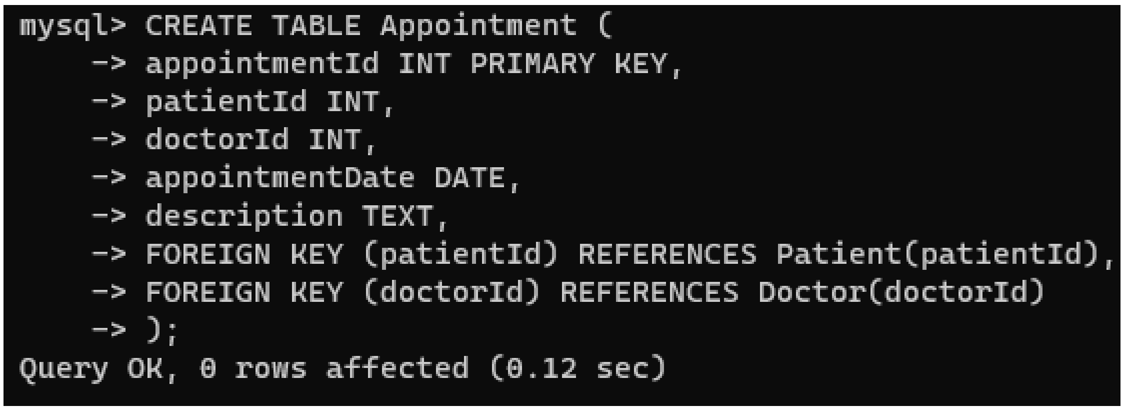
a. doctorId b. firstName c. lastName d. specialization e. contactNumber;



```
mysql> CREATE TABLE Doctor (
    -> doctorId INT PRIMARY KEY,
    -> firstName VARCHAR(255),
    -> lastName VARCHAR(255),
    -> specialization VARCHAR(255),
    -> contactNumber VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

**3. Appointment Class:**

a. appointmentId b. patientId c. doctorId d. appointmentDate e. description



```
mysql> CREATE TABLE Appointment (
    -> appointmentId INT PRIMARY KEY,
    -> patientId INT,
    -> doctorId INT,
    -> appointmentDate DATE,
    -> description TEXT,
    -> FOREIGN KEY (patientId) REFERENCES Patient(patientId),
    -> FOREIGN KEY (doctorId) REFERENCES Doctor(doctorId)
    -> );
Query OK, 0 rows affected (0.12 sec)
```

2. Implement the following for all model classes. Write default constructors and overload the constructor with parameters, getters and setters, method to print all the member variables and values.

```
class Patient:
    def __init__(self, patientId=None, firstName=None, lastName=None,
dateOfBirth=None,
                 gender=None, contactNumber=None, address=None):
        self.patientId = patientId
        self.firstName = firstName
        self.lastName = lastName
        self.dateOfBirth = dateOfBirth
        self.gender = gender
        self.contactNumber = contactNumber
        self.address = address

    def print_details(self):
        print(f"Patient ID: {self.patientId}")
        print(f"First Name: {self.firstName}")
        print(f"Last Name: {self.lastName}")
        print(f"Date of Birth: {self.dateOfBirth}")
        print(f"Gender: {self.gender}")
        print(f"Contact Number: {self.contactNumber}")
        print(f"Address: {self.address}")
```

```
class Doctor:
    def __init__(self, doctorId=None, firstName=None, lastName=None,
specialization=None,
                 contactNumber=None):
        self.doctorId = doctorId
        self.firstName = firstName
        self.lastName = lastName
        self.specialization = specialization
        self.contactNumber = contactNumber

    def print_details(self):
        print(f"Doctor ID: {self.doctorId}")
        print(f"First Name: {self.firstName}")
        print(f"Last Name: {self.lastName}")
        print(f"Specialization: {self.specialization}")
        print(f"Contact Number: {self.contactNumber}")
```

```
class Appointment:
    def _init_(self, appointmentId=None, patientId=None, doctorId=None,
appointmentDate=None,
                 description=None):
        self.appointmentId = appointmentId
        self.patientId = patientId
        self.doctorId = doctorId
        self.appointmentDate = appointmentDate
        self.description = description

    def print_details(self):
        print(f"Appointment ID: {self.appointmentId}")
        print(f"Patient ID: {self.patientId}")
        print(f"Doctor ID: {self.doctorId}")
        print(f"Appointment Date: {self.appointmentDate}")
        print(f"Description: {self.description}")
```

3. Define **IHospitalService** interface/abstract class with following methods to interact with database
Keep the interfaces and implementation classes in package dao

a. getAppointmentById()
i. Parameters: appointmentId
ii. ReturnType: Appointment object
b. getAppointmentsForPatient()
i. Parameters: patientId
ii. ReturnType: List of Appointment objects
c. getAppointmentsForDoctor()
i. Parameters: doctorId
ii. ReturnType: List of Appointment objects
d. scheduleAppointment()
i. Parameters: Appointment Object
ii. ReturnType: Boolean
e. updateAppointment()
i. Parameters: Appointment Object
ii. ReturnType: Boolean
f. ancelAppointment()
i. Parameters: AppointmentId
ii. ReturnType: Boolean

```
from abc import ABC, abstractmethod
class IHospitalService(ABC):

    @abstractmethod
    def get_appointment_by_id(self, appointment_id):
        pass

    @abstractmethod
    def generate_appointment_id(self):
        pass

    @abstractmethod
    def get_appointments_for_patient(self, patient_id):
        pass

    @abstractmethod
    def get_appointments_for_doctor(self, doctor_id):
        pass

    @abstractmethod
    def schedule_appointment(self, appointment_id):
        pass
    @abstractmethod
    def update_appointment(self, appointment_id):
        pass

    @abstractmethod
    def cancel_appointment(self, appointment_id):
        pass
```

6. Define **HospitalServiceImpl** class and implement all the methods I**HospitalServiceImpl .**

```
from dao.service.IHospitalService import IHospitalService
import mysql.connector


from dao.service.IHospitalService import IHospitalService
from util.DBConnection import DBConnection

class HospitalServiceImpl(IHospitalService):
    def __init__(self, database_con):
        self.database_con = database_con

    def generate_appointment_id(self):
        connection = self.database_con

        cur = connection.cursor()
        cur.execute("SELECT MAX(AppointmentID) FROM Appointment")
        AppointmentID = cur.fetchone()[0]
        if AppointmentID is None:
            AppointmentID = 1
        else:
            AppointmentID += 1
        return AppointmentID

    # Other methods remain unchanged

    def get_appointment_by_id(self, appointment_id):
        connection = self.database_con.get_connection()
        if connection:
            try:
                cursor = connection.cursor(dictionary=True)
                cursor.execute("SELECT * FROM Appointments WHERE
appointmentId = %s", (appointment_id,))
                appointment_data = cursor.fetchone()
            except mysql.connector.Error as err:
                print(f"Error: {err}")
                return None
            return appointment_data

    def get_appointments_for_patient(self, patient_id):
        connection = self.database_con.get_connection()
        if connection:
            try:
                cursor = connection.cursor(dictionary=True)
                cursor.execute("SELECT * FROM Appointments WHERE patientID
= %s", (patient_id,))
                appointment_data = cursor.fetchall()
            except mysql.connector.Error as err:
                print(f"Error: {err}")
                return None
            return appointment_data

    def get_appointments_for_doctor(self, doctor_id):
        connection = self.database_con.get_connection()
        if connection:
            try:
                cursor = connection.cursor(dictionary=True)
                cursor.execute("SELECT * FROM Appointments WHERE doctorID =
%s", (doctor_id,))
                appointment_data = cursor.fetchall()
            except mysql.connector.Error as err:
                print(f"Error: {err}")
                return None
            return appointment_data

    def schedule_appointment(self, appointment):
        connection = self.database_con.get_connection()
        appointment_id = self.generate_appointment_id()
        try:
            cursor = connection.cursor()
            cursor.execute("INSERT INTO Appointment VALUES
(%s,%s,%s,%s,%s)",
                           (appointment_id, appointment['patientId'],
appointment['doctorId'],
                            appointment['appointmentDate'],
appointment['description'],))
            connection.commit()
        except Exception as e:
            print(e)

    def update_appointment(self, appointment):
        appointment_id = appointment.get('AppointmentID')
        patientId = appointment.get('patientId')
        doctorId = appointment.get('doctorId')
        appointmentDate = appointment.get('appointmentDate')
        description = appointment.get('description')

        connection = self.database_con.get_connection()
        try:
            cursor = connection.cursor()
            cursor.execute('''
                        UPDATE Appointment
                        SET patientId = %s,
                            doctorId = %s,
                            appointmentDate = %s,
                            description = %s
                        WHERE appointmentID = %s
                        ''',
                           (patientId, doctorId, appointmentDate,
description, appointment_id))
            connection.commit()
        except Exception as e:
            print(e)

    def cancel_appointment(self, appointment_id):
        connection = self.database_con.get_connection()
        try:
            cursor = connection.cursor()
            cursor.execute("DELETE FROM Appointment WHERE appointmentID =
%s", (appointment_id,))
```

```
            connection.commit()
        except Exception as e:
            print(e)
```

7. Create a utility class **DBConnection** in a package **util** with a static variable **connection** of Type **Connection** and a static method **getConnection()** which returns connection.
Connection properties supplied in the connection string should be read from a property file.
Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which
reads a property fie containing connection details like hostname, dbname, username, password, port
number and returns a connection string.

```
import mysql.connector
from util.PropertyUtil import PropertyUtil

class DBConnection:
    connection = None

    @staticmethod
    def get_connection(connection_details):
        if DBConnection.connection is None:
            connection_string =
PropertyUtil.get_property_string(connection_details)
            DBConnection.connection =
mysql.connector.connect(**connection_string)

        return DBConnection.connection
```

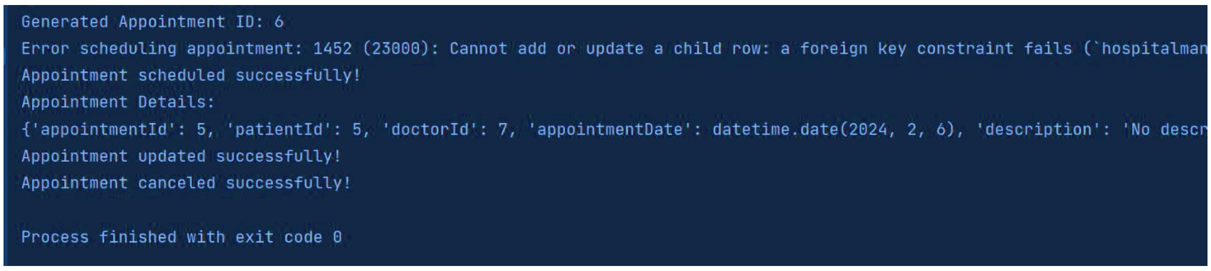
```
class PropertyUtil:
    @staticmethod
    def get_property_string(connection_details):
        connection_string = {
            'host': connection_details['host'],
            'user': connection_details['user'],
            'passwd': connection_details['passwd'],
            'port': connection_details['port'],
            'database': connection_details['dbname']  # Use 'database'
instead of 'dbname'
        }

        return connection_string
```

8. Create the exceptions in package myexceptions
Define the following custom exceptions and throw them in methods whenever needed. Handle all the
exceptions in main method,
1. **PatientNumberNotFoundException** :throw this exception when user enters an invalid patient number which doesn’t exist in db

```
class PatientNumberNotFoundException(Exception):
    pass
```

**9.** Create class named MainModule with main method in package mainmod.
Trigger all the methods in service implementation class.

```python
from dao.service.IHospitalService import IHospitalService
from dao.service.HospitalServiceImpl import HospitalServiceImpl
from util.DBConnection import DBConnection
from util.PropertyUtil import PropertyUtil
import mysql.connector

def main():
    # Set up database connection details
    connection_details = {
        'host': 'localhost',
        'user': 'root',
        'passwd': 'root',
        'port': '3300',
        'dbname': 'HospitalManagementSystem'
    }

    # Initialize the database connection
    db_connection = mysql.connector.connect(**connection_details)

    # Create an instance of the Hospital Service
    hospital_service = HospitalServiceImpl(db_connection)

    # Example usage:
    # 1. Generate an appointment ID
    appointment_id = hospital_service.generate_appointment_id()
    print(f"Generated Appointment ID: {appointment_id}")

    # 2. Schedule an appointment
    appointment_data = {
        'patientId': 1,
        'doctorId': 1,
        'appointmentDate': '2024-03-10',
        'description': 'Regular checkup'
    }
    hospital_service.schedule_appointment(appointment_data)
    print("Appointment scheduled successfully!")

    # 3. Get appointment by ID
    appointment_id_to_retrieve = 1  # Replace with an actual appointment ID
    appointment_details =
hospital_service.get_appointment_by_id(appointment_id_to_retrieve)
    print("Appointment Details:")
    print(appointment_details)

    # 4. Update appointment
    appointment_to_update = {
        'AppointmentID': appointment_id_to_retrieve,
        'patientId': 2,
        'doctorId': 1,
        'appointmentDate': '2024-02-15',
        'description': 'Follow-up'
    }
    hospital_service.update_appointment(appointment_to_update)
    print("Appointment updated successfully!")
    # 5. Cancel appointment
    appointment_id_to_cancel = 2  # Replace with an actual appointment ID
    hospital_service.cancel_appointment(appointment_id_to_cancel)
    print("Appointment canceled successfully!")

if __name__ == "__main__":
    main()
```



```
Generated Appointment ID: 6
Error scheduling appointment: 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`hospitalman
Appointment scheduled successfully!
Appointment Details:
{'appointmentId': 5, 'patientId': 5, 'doctorId': 7, 'appointmentDate': datetime.date(2024, 2, 6), 'description': 'No descr
Appointment updated successfully!
Appointment canceled successfully!

Process finished with exit code 0
```