Python Coding Challenge

Hospital Management System

Problem Statement: Create SQL Schema from the following classes class, use the class attributes for table column names.

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

```
mysql> create database hospital;
Query OK, 1 row affected (0.02 sec)

mysql> use hospital;
Database changed
```

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 (
    ->     patient_Id INT AUTO_INCREMENT PRIMARY KEY,
    ->     firstName VARCHAR(50) NOT NULL,
    ->     lastName VARCHAR(50) NOT NULL,
    ->     dateOfBirth DATE NOT NULL,
    ->     gender VARCHAR(10) NOT NULL,
    ->     contactNumber VARCHAR(15) NOT NULL,
    ->     address VARCHAR(255) NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Define 'Doctor` class with the following confidential attributes:
a. doctorId
b. firstName
c. lastName
d. specialization
e. contactNumber

```
mysql> CREATE TABLE Doctor (
    ->     doctor_Id INT AUTO_INCREMENT PRIMARY KEY,
    ->     firstName VARCHAR(50) NOT NULL,
    ->     lastName VARCHAR(50) NOT NULL,
    ->     specialization VARCHAR(50) NOT NULL,
    ->     contactNumber VARCHAR(15) NOT NULL
    -> );
Query OK, 0 rows affected (0.06 sec)
```

Define Appointment Class:
a. appointmentId
b. patientId
c. doctorId
d. appointmentDate
e. description

```
mysql> CREATE TABLE Appointments (
    ->     appointmentId INT AUTO_INCREMENT PRIMARY KEY,
    ->     patient_Id INT NOT NULL,
    ->     doctor_Id INT NOT NULL,
    ->     appointment_date DATE NOT NULL,
    ->     description VARCHAR(255),
    ->     FOREIGN KEY (patient_Id) REFERENCES Patient(patient_Id),
    ->     FOREIGN KEY (doctor_Id) REFERENCES Doctor(doctor_Id)
    -> );
Query OK, 0 rows affected (0.05 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.

```python
from DBConnUtil import DBConnUtil
from datetime import date
class Patient:
    def __init__(self, patient_id, first_name, last_name, date_of_birth, gender, contact_number, address):
        self.patient_id = patient_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.gender = gender
        self.contact_number = contact_number
        self.address = address

    def __str__(self):
        return f"Patient ID: {self.patient_id}, Name: {self.first_name} {self.last_name}, " \
               f"DOB: {self.date_of_birth}, Gender: {self.gender}, Contact: {self.contact_number}, " \
               f"Address: {self.address}"
```

```python
from DBConnUtil import DBConnUtil
class Doctor:
    def __init__(self, doctor_id, first_name, last_name, specialization, contact_number):
        self.doctor_id = doctor_id
        self.first_name = first_name
        self.last_name = last_name
        self.specialization = specialization
        self.contact_number = contact_number

    def __str__(self):
        return f"Doctor ID: {self.doctor_id}, Name: {self.first_name} {self.last_name}, " \
               f"Specialization: {self.specialization}, Contact: {self.contact_number}"
```

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. cancelAppointment()
      i. Parameters: AppointmentId
      ii. ReturnType: Boolean

```python
from DBConnUtil import DBConnUtil
from datetime import date
class HospitalService:
    def get_appointment_by_id(self, appointment_id):
        pass

    def get_appointments_for_patient(self, patient_id):
        pass

    def get_appointments_for_doctor(self, doctor_id):
        pass

    def schedule_appointment(self, appointment):
        pass

    def update_appointment(self, date, new_description, appointment_id):
        pass

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

4. Define HospitalServiceImpl class and implement all the methods IHospitalServiceImpl .

```python
from HospitalService import HospitalService
import DBConnUtil

class HospitalServiceImpl(HospitalService):
    def execute_query(self, query, values=None):
        conn = None
        stmt = None
        try:
            conn = DBConnUtil.get_connection()
            stmt = conn.cursor(dictionary=True)
            stmt.execute(query, values)
            return stmt.fetchall()
        except Exception as e:
            print(f"Error executing query: {e}")
            return None
        finally:
            if conn:
                conn.close()

    def execute_update(self, query, values=None):
        conn = None
        stmt = None
        try:
            conn = DBConnUtil.get_connection()
            stmt = conn.cursor()
            stmt.execute(query, values)
            conn.commit()
            return True
```

```
        except Exception as e:
            print(f"Error executing update: {e}")
            return False
        finally:
            if conn:
                conn.close()

    def get_appointment_by_id(self, appointment_id):
        query = "SELECT * FROM appointments WHERE appointment_id = %s"
        result = self.execute_query(query, (appointment_id,))
        return result[0] if result else None

    def get_appointments_for_patient(self, patient_id):
        query = "SELECT * FROM appointments WHERE patient_id = %s"
        return self.execute_query(query, (patient_id,))

    def get_appointments_for_doctor(self, doctor_id):
        query = "SELECT * FROM appointments WHERE doctor_id = %s"
        return self.execute_query(query, (doctor_id,))

    def schedule_appointment(self, appointment):
        query = "INSERT INTO appointments (patient_id, doctor_id, appointment_date, description) VALUES (%s, %s, %s,
        values = (appointment.patient_id, appointment.doctor_id, appointment.appointment_date, appointment.descriptio
        return self.execute_update(query, values)

    def update_appointment(self, date, new_description, appointment_id):
        query = "UPDATE appointments SET appointment_date = %s, description = %s WHERE appointment_id = %s"
        values = (date, new_description, appointment_id)
```

```
def get_appointments_for_patient(self, patient_id):
    query = "SELECT * FROM appointments WHERE patient_id = %s"
    return self.execute_query(query, (patient_id,))

def get_appointments_for_doctor(self, doctor_id):
    query = "SELECT * FROM appointments WHERE doctor_id = %s"
    return self.execute_query(query, (doctor_id,))

def schedule_appointment(self, appointment):
    query = "INSERT INTO appointments (patient_id, doctor_id, appointment_date, description) VALUES (%s, %s, %s, %s)
    values = (appointment.patient_id, appointment.doctor_id, appointment.appointment_date, appointment.description)
    return self.execute_update(query, values)

def update_appointment(self, date, new_description, appointment_id):
    query = "UPDATE appointments SET appointment_date = %s, description = %s WHERE appointment_id = %s"
    values = (date, new_description, appointment_id)
    return self.execute_update(query, values)

def cancel_appointment(self, appointment_id):
    query = "DELETE FROM appointments WHERE appointment_id = %s"
    return self.execute_update(query, (appointment_id,))
```

5. 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.

property_file.txt

```
host=localhost
database=hospital
user=root
password=root
port=3306

```

```python
class DBPropertyUtil:
    @staticmethod
    def get_connection_properties(property_file_path='property_file.txt'):
        try:
            with open(property_file_path, 'r') as file:
                properties = {}
                for line in file:
                    key, value = line.strip().split('=')
                    properties[key.strip()] = value.strip()
                return properties
        except Exception as e:
            print(f"Error reading property file: {e}")
            return None
```

```python
import mysql.connector
from DBPropertyUtil import DBPropertyUtil

class DBConnUtil:
    @staticmethod
    def get_connection():
        try:
            properties = DBPropertyUtil.get_connection_properties()
            connection = mysql.connector.connect(**properties)
            print("Connected")
            return connection
        except mysql.connector.Error as e:
            print(f"Error connecting to MySQL: {e}")
            return None
```

6. 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,
   a. PatientNumberNotFoundException :throw this exception when user enters an invalid patient number which doesn't exist in db

```python
class PatientNumberNotFoundException(Exception):
    def __init__(self, patient_number):
        super().__init__(f"Patient with number {patient_number} not found.")
        self.patient_number = patient_number
```

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

```python
from DBConnUtil import DBConnUtil
class HospitalService:
    def get_appointment_by_id(self, appointment_id):
        pass

    def get_appointments_for_patient(self, patient_id):
        pass

    def get_appointments_for_doctor(self, doctor_id):
        pass

    def schedule_appointment(self, appointment):
        pass

    def update_appointment(self, date, new_description, appointment_id):
        pass

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

Output:

```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
0. Exit
Enter your choice:
```

```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
0. Exit
Enter your choice: 2
Enter Patient ID: 1
Connected
Appointments for Patient:
{'appointmentId': 1, 'patient_Id': 1, 'doctor_Id': 1, 'appointment_date': datetime.date(2023, 12, 23), 'description': 'Cardiologist'}
```

```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
0. Exit
Enter your choice: 4
Enter Patient ID for the appointment: 2
Enter Doctor ID for the appointment: 2
Enter Appointment Date (YYYY-MM-DD): 2023-12-23
Enter Appointment Description: Dermatologist
Connected
Appointment scheduled successfully.
```

```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
0. Exit
Enter your choice: 3
Enter Doctor ID: 2
Connected
Appointments for Doctor:
{'appointmentId': 3, 'patient_Id': 2, 'doctor_Id': 2, 'appointment_date': datetime.date(2023, 12, 23), 'description': 'Dermatologist'}
```

```
Menu:
1. Get Appointment by ID
2. Get Appointments for Patient
3. Get Appointments for Doctor
4. Schedule Appointment
5. Update Appointment
6. Cancel Appointment
0. Exit
Enter your choice: 0
```