Coding challenge-3: - Hospital management system

Name: Dheeraj Vemula

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

Creating patient table in SQL.

```
mysql> CREATE TABLE patient (
    ->     patientId INT PRIMARY KEY,
    ->     firstName VARCHAR(50),
    ->     lastName VARCHAR(50),
    ->     dateOfBirth DATE,
    ->     gender VARCHAR(10),
    ->     contactNumber VARCHAR(15),
    ->     address VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.12 sec)
```

Creating doctor table in SQL.

```
mysql> CREATE TABLE doctor (
    ->     doctorId INT PRIMARY KEY,
    ->     firstName VARCHAR(50),
    ->     lastName VARCHAR(50),
    ->     specialization VARCHAR(50),
    ->     contactNumber VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

Creating appointment table in SQL.

```
mysql> CREATE TABLE appointment (
    ->     appointmentId INT PRIMARY KEY,
    ->     patientId INT,
    ->     doctorId INT,
    ->     appointmentDate DATE,
    ->     description VARCHAR(255),
    ->     FOREIGN KEY (patientId) REFERENCES patient(patientId),
    ->     FOREIGN KEY (doctorId) REFERENCES doctor(doctorId)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

2.Create the following model/entity classes within package entity with variables declared private, constructors (default and parametrized,getters,setters and toString()).Creating `Patient` class with the following confidential attributes:patient_Id, firstName, lastName, dateOfBirth, gender, contactNumber and address;

```python
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

    # Getter methods
    def get_patient_id(self):
        return self.__patient_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_date_of_birth(self):
        return self.__date_of_birth

    def get_gender(self):
        return self.__gender

    def get_contact_number(self):
        return self.__contact_number
```

Creating `Doctor` class with the following confidential attributes:

DoctorId, firstName, lastName, specialization, contactNumber;

```python
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

    # Getter methods
    def get_doctor_id(self):
        return self.__doctor_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_specialization(self):
        return self.__specialization

    def get_contact_number(self):
        return self.__contact_number

    # Setter methods (if needed)
    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
```

```python
    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_specialization(self, specialization):
        self.__specialization = specialization

    def set_contact_number(self, contact_number):
        self.__contact_number = contact_number
```

Creating `Appointment` class with the following confidential attributes:

```
class Appointment:
    def __init__(self, appointment_id, patient_id, doctor_id, appointment_date, description):
        self.__appointment_id = appointment_id
        self.__patient_id = patient_id
        self.__doctor_id = doctor_id
        self.__appointment_date = appointment_date
        self.__description = description

    # Getter methods
    def get_appointment_id(self):
        return self.__appointment_id

    def get_patient_id(self):
        return self.__patient_id

    def get_doctor_id(self):
        return self.__doctor_id

    def get_appointment_date(self):
        return self.__appointment_date

    def get_description(self):
        return self.__description

    # Setter methods (if needed)
    def set_patient_id(self, patient_id):
        self.__patient_id = patient_id
```

```
def set_doctor_id(self, doctor_id):
    self.__doctor_id = doctor_id

def set_appointment_date(self, appointment_date):
    self.__appointment_date = appointment_date

def set_description(self, description):
    self.__description = description
```

3.Define IHospitalService interface/abstract class with following methods to interact with database.

```python
from abc import ABC, abstractmethod
from typing import List
from assignments.hospman.entity.appointment import Appointment

3 usages
class IHospitalService(ABC):
    @abstractmethod
    def get_appointment_by_id(self, appointment_id) -> Appointment:
        pass

    @abstractmethod
    def get_appointments_for_patient(self, patient_id) -> List[Appointment]:
        pass

    @abstractmethod
    def get_appointments_for_doctor(self, doctor_id) -> List[Appointment]:
        pass

    @abstractmethod
    def schedule_appointment(self, appointment: Appointment) -> bool:
        pass

    @abstractmethod
    def update_appointment(self, appointment: Appointment) -> bool:
        pass

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

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

getAppointmentById():

```
32      def get_appointment_by_id(self, appointment_id: int):
33          query = "SELECT * FROM appointment WHERE appointmentId = %s"
34          self.cursor.execute(query, (appointment_id,))
35          result = self.cursor.fetchone()
36
37          if result:
38              print(result[0], result[1], result[2], result[3], result[4])
```

getAppointmentsForPatient():

```
def get_appointments_for_patient(self, patient_id: int):
    query = "SELECT * FROM appointment WHERE patientId = %s"
    self.cursor.execute(query, (patient_id,))
    results = self.cursor.fetchall()

    if not results:
        raise PatientNumberNotFoundException(f"No appointments found for Patient with ID {patient_id}")
```

getAppointmentsForDoctor():

```
def get_appointments_for_doctor(self, doctor_id: int):
    query = "SELECT * FROM appointment WHERE doctorId = %s"
    self.cursor.execute(query, (doctor_id,))
    results = self.cursor.fetchall()

    appointments = []
    for result in results:
        print(result[0], result[1], result[2], result[3], result[4])
        appointments.append(result)

    return appointments
```

scheduleAppointment():

```
    def schedule_appointment(self, appointment: Appointment):
        query = "INSERT INTO appointment (appointmentId,patientId, doctorId, appointmentDate, description) VALUE
        print("Appointment is  scheduled" )
        values = (appointment[0], appointment[1], appointment[2], appointment[3],appointment[4])
        self.cursor.execute(query, values)
        self.connection.commit()
        return True
```

updateAppointment():

```
    def update_appointment(self, appointment: Appointment):
        query = "UPDATE appointment SET appointmentDate = %s, description = %s WHERE appointmentId = %s"
        values = (appointment.appointmentDate, appointment.description, appointment.appointmentId)
        self.cursor.execute(query, values)
        self.connection.commit()
        print("Appointment is Updated")
        return True
```

CancelAppointment():

```
def cancel_appointment(self, appointment_id: int):
    query = "DELETE FROM appointment WHERE appointmentId = %s"
    self.cursor.execute(query, (appointment_id,))
    self.connection.commit()
    print("Delection Successfull")
    return True
```

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.

```
from mysql.connector import connect
from assignments.hospman.util.propertyutil import PropertyUtil

4 usages
class DBConnection:
    connection = None

    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            connection_string = PropertyUtil.get_property_string()
            DBConnection.connection = connect(**connection_string)
        return DBConnection.connection
```

6.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.

```python
from typing import Dict

2 usages
class PropertyUtil:
    1 usage
    @staticmethod
    def get_property_string() -> Dict:
        # Replace with your actual logic to read properties from a file
        return {
            'host': 'localhost',
            'user': 'root',
            'password': 'Kevink25*',
            'database': 'hospital',
            'port': 3306  # Change the port number accordingly
        }
```

7.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

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

```python
def get_appointments_for_patient(self, patient_id: int):
    query = "SELECT * FROM appointment WHERE patientId = %s"
    self.cursor.execute(query, (patient_id,))
    results = self.cursor.fetchall()

    if not results:
        raise PatientNumberNotFoundException(f"No appointments found for Patient with ID {patient_id}")
```

8.Create class named Mainapp with main method in package main and trigger all the methods in the implementation class.

Coding_Challenge > main.py > ...

```python
from IHospitalService import HospitalService
from Appointment import Appointment
from datetime import date
from myexceptions import PatientNumberNotFoundException

class main:

    def get_appointment_by_id(self):
        print(hospital_service.get_appointment_by_id(1003), "\n")

    def get_appointments_for_patient(self):
        print(hospital_service.get_appointments_for_patient(102), "\n")

    def get_appointments_for_doctor(self):
        print(hospital_service.get_appointments_for_doctor(4), "\n")

    def schedule_appointment(self):
        new_appointment = Appointment(patientId=1, doctorId=1, appointmentDate=date(2023, 12, 31), description='
        print("Appointment Scheduled")

    def update_appointment(self):
        updated_appointment = Appointment(appointmentId=1, patientId=1, doctorId=1, appointmentDate=date(2023, 1
        print(hospital_service.update_appointment(updated_appointment))

    def cancel_appointment(self):
        print(hospital_service.cancel_appointment(1001))

```