Name : Gayathri G

Coding Challenge: Hospital Management System

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;

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

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

3. **Appointment Class:**

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

TABLE doctors:

DoctorId	FirstName	Lastname	Specialization	ContactNumber
1	Kousalya	Velu	Gynocologist	9887654356
2	Radhakrishnan	palani	Ortho	1234840897
3	Raajendran	Kumar	ent	9121234543
NULL	NULL	NULL	NULL	NULL

TABLE patients:

PatientId	FirstName	LastName	DateofBirth	Gender	ContactNumber	Address
1	gayathri	Guna	2003-09-08	Female	9876543456	Coimbatore
2	Swathi	Guna	2001-05-03	Female	1234657897	Salem
3	Raaj	Kishor	2003-03-21	Male	9123456543	Punjab
NULL	NULL	NULL	NULL	NULL	NULL	NULL

TABLE appointments:

AppointmentId	PatientId	DoctorId	AppointmentDate	Description
2	1	1	2024-05-02	Take MRI Test
3	1	1	2024-05-03	Drink Water before test
4	2	1	2024-06-01	take medicines for next 2 weeks
5	3	1	2024-06-02	take injections daily
6	1	2	2024-07-01	bring test reports
7	1	3	2024-07-03	take blood test
8	2	3	2024-09-01	take sugar test

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 PATIENTS:

```
3.class Patients:
    def __init__(self, patient_id=0, 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}\nFirst Name:
{self.first_name}\nLast Name: {self.last_name}\nDate of Birth:
{self.date_of_birth}\nGender: {self.gender}\nContact Number:
{self.contact_number}\nAddress: {self.address}"

    @property
```

```
    def patient_id(self):
        return self._patient_id

    @patient_id.setter
    def patient_id(self, patient_id):
        self._patient_id = patient_id

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, first_name):
        self._first_name = first_name

    @property
    def last_name(self):
        return self._last_name

    @last_name.setter
    def last_name(self, last_name):
        self._last_name = last_name

    @property
    def date_of_birth(self):
        return self._date_of_birth

    @date_of_birth.setter
    def date_of_birth(self, date_of_birth):
        self._date_of_birth = date_of_birth

    @property
    def gender(self):
        return self._gender

    @gender.setter
    def gender(self, gender):
        self._gender = gender

    @property
    def contact_number(self):
        return self._contact_number

    @contact_number.setter
    def contact_number(self, contact_number):
        self._contact_number = contact_number

    @property
    def address(self):
        return self._address

    @address.setter
    def address(self, address):
        self._address = address

    def display(self):
        print("Patient id: ", self.patient_id)
        print("First Name: ", self.first_name)
        print("Last Name: ", self.last_name)
        print("Date of Birth: ", self.date_of_birth)
        print("Gender: ", self.gender)
```

```
        print("Contact Number: ", self.contact_number)
        print("Address: ", self.address)


patient = Patients(1, "Gayu", "Guna", "2003-09-08", "female",
9876564532, "Coimbatore")
patient.display()
```

```
Patient id:  1
First Name:  Gayu
Last Name:  Guna
Date of Birth:  2003-09-08
Gender:  female
Contact Number:  9876564532
Address:  Coimbatore

Process finished with exit code 0
```

CLASS DOCTORS:

```
class Doctors:
    def __init__(self, doctor_id=0, 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}\nFirst Name:
{self.first_name}\nLast Name: {self.last_name}\nSpecialization:
{self.specialization}\nContact Number: {self.contact_number}"

    @property
    def doctor_id(self):
        return self._doctor_id

    @doctor_id.setter
    def doctor_id(self, doctor_id):
        self._doctor_id = doctor_id

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, first_name):
        self._first_name = first_name

    @property
    def last_name(self):
        return self._last_name
```

```
    @last_name.setter
    def last_name(self, last_name):
        self._last_name = last_name

    @property
    def specialization(self):
        return self._specialization

    @specialization.setter
    def specialization(self, specialization):
        self._specialization = specialization

    @property
    def contact_number(self):
        return self._contact_number

    @contact_number.setter
    def contact_number(self, contact_number):
        self._contact_number = contact_number

    def display(self):
        print(self)

doct=Doctors(1,"Raam","Kumar","ENT",9987654321)
doct.display()
```

```
Doctor ID: 1
First Name: Raam
Last Name: Kumar
Specialization: ENT
Contact Number: 9987654321

Process finished with exit code 0
```

CLASS APPOINTMENTS:

```
class Appointments:
    def __init__(self, appointment_id=0, patient_id=0, doctor_id=0,
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

    def __str__(self):
        return f"Appointment ID: {self.appointment_id}\nPatient ID:
{self.patient_id}\nDoctor ID: {self.doctor_id}\nAppointment Date:
{self.appointment_date}\nDescription: {self.description}"

    @property
    def appointment_id(self):
        return self._appointment_id
```

```python
    @appointment_id.setter
    def appointment_id(self, appointment_id):
        self._appointment_id = appointment_id

    @property
    def patient_id(self):
        return self._patient_id

    @patient_id.setter
    def patient_id(self, patient_id):
        self._patient_id = patient_id

    @property
    def doctor_id(self):
        return self._doctor_id

    @doctor_id.setter
    def doctor_id(self, doctor_id):
        self._doctor_id = doctor_id

    @property
    def appointment_date(self):
        return self._appointment_date

    @appointment_date.setter
    def appointment_date(self, appointment_date):
        self._appointment_date = appointment_date

    @property
    def description(self):
        return self._description

    @description.setter
    def description(self, description):
        self._description = description

    def display(self):
        print(self)

appoint=Appointments(1,1,1,"2024-05-03","Bring the MRI Report")
appoint.display()
```

```
Appointment ID: 1
Patient ID: 1
Doctor ID: 1
Appointment Date: 2024-05-03
Description: Bring the MRI Report

Process finished with exit code 0
```

4. 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

CLASS IHospitalService:

```
from abc import ABC, abstractmethod
from typing import List
from Entity import Appointments


class IHospitalService(ABC):
    @abstractmethod
    def get_appointment_by_id(self, appointment_id: int) -> Appointments:
        pass

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

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

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

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

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

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

```
from dao.IHospitalService import IHospitalService
from Entity.Appointments import Appointments
import mysql.connector
from typing import List
from myexceptions.PatientNumberNotFoundException import
PatientNumberNotFoundException
from util.DBConnection import DBConnection


class HospitalServiceImpl(IHospitalService):
    def __init__(self):
        self.connection=DBConnection.getConnection()
        # self.connection = mysql.connector.connect(
        #     host='localhost',
        #     user='root',
        #     password='root',
        #     port="3306",
        #     database='hospitalmanagement'
        # )
        self.mycursor = self.connection.cursor()

    def get_appointment_by_id(self, appointment_id: int) -> Appointments:
        appointment = None
        try:
            mycursor = self.connection.cursor()
            mycursor.execute("SELECT * FROM appointment WHERE
AppointmentId= %s", (appointment_id,))
            data = mycursor.fetchone()
            if data:
                appointment = Appointments(*data)
        except Exception as e:
            print("Error:", e)
        print(appointment)
        return appointment

    def get_appointments_for_patient(self, patient_id: int) ->
list[Appointments]:
        appointments = []
        self.mycursor.execute("select * from patients WHERE PatientId =
%s",(patient_id,))
        patient=self.mycursor.fetchone()
        if(patient is None):
            raise PatientNumberNotFoundException(patient_id)
        try:
            self.mycursor.execute("SELECT * FROM appointment WHERE
PatientId = %s", (patient_id,))
```

```
            data = self.mycursor.fetchall()
            for row in data:
                appointment = Appointments(*row)
                appointments.append(appointment)
        except Exception as e:
            print("Error:", e)
        for app in appointments:
            print(app)
        return appointments

    def get_appointments_for_doctor(self, doctor_id: int) ->
list[Appointments]:
        appointments = []
        try:
            mycursor = self.connection.cursor()
            mycursor.execute("SELECT * FROM appointment WHERE DoctorId =
%s", (doctor_id,))
            data = mycursor.fetchall()
            for row in data:
                appointment = Appointments(*row)
                appointments.append(appointment)
        except Exception as e:
            print("Error:", e)
        for app in appointments:
            print(app)
        return appointments

    def schedule_appointment(self, appointment: Appointments) -> bool:
        try:
            mycursor = self.connection.cursor()
            mycursor.execute(
                "INSERT INTO appointment (AppointmentId,PatientId,
DoctorId, AppointmentDate, Description) VALUES ("
                "%s,%s, %s, %s, %s)",
                (appointment.appointment_id,appointment.patient_id,
appointment.doctor_id, appointment.appointment_date,
appointment.description))
            self.connection.commit()
            print("scheduled success")
            return True
        except Exception as e:
            print("Error:", e)
            return False


    def update_appointment(self, appointment: Appointments) -> bool:
        try:
            mycursor = self.connection.cursor()
            mycursor.execute(
                "UPDATE appointment SET PatientId = %s, DoctorId = %s,
AppointmentDate = %s, Description = %s WHERE AppointmentId = %s",
                (appointment.patient_id, appointment.doctor_id,
appointment.appointment_date, appointment.description,
                 appointment.appointment_id))
            self.connection.commit()
            print("Updated Successfully")
            return True
        except Exception as e:
            print("Error:", e)
            return False
```

```
    def cancel_appointment(self, appointment_id: int) -> bool:
        try:
            mycursor = self.connection.cursor()
            mycursor.execute("DELETE FROM appointment WHERE AppointmentId =
%s", (appointment_id,))
            self.connection.commit()
            print("Appointment Cancelled")
            return True
        except Exception as e:
            print("Error:", e)
            return False
```

```
Connection successful
Appointment ID: 3
Patient ID: 1
Doctor ID: 1
Appointment Date: 2024-05-03
Description: Drink Water before test
Patient with number 50 not found in the database
Appointment ID: 6
Patient ID: 1
Doctor ID: 2
Appointment Date: 2024-07-01
Description: bring test reports
scheduled success
Updated Successfully
Appointment Cancelled

Process finished with exit code 0
```

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.

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


class DBConnection:
    connection = None

    def getConnection():
        if DBConnection.connection is None:
            connection_string = PropertyUtil.getPropertyString()
            try:
                DBConnection.connection =
mysql.connector.connect(**connection_string)
```

```
                print("Connection successful")
            except mysql.connector.Error as error:
                print("Error while connecting to MySQL", error)
        return DBConnection.connection
```

```
Connection successful

Process finished with exit code 0
```

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.

```
class PropertyUtil:
    @staticmethod
    def getPropertyString():
        connection_string = {
            'host': "localhost",
            'database': "hospitalmanagement",
            'user': "root",
            'password': "root",
            'port': "3306"
        }
        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):
    def __init__(self, patient_number):
        self.patient_number = patient_number
        super().__init__(f"Patient with number {patient_number} not found
in the database")
```

```
Connection successful
Patient with number 50 not found in the database

Process finished with exit code 0
```

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

```
from dao.HospitalServiceImpl import HospitalServiceImpl
from myexceptions.PatientNumberNotFoundException import
PatientNumberNotFoundException
from Entity.Appointments import Appointments
from dao.IHospitalService import IHospitalService
class Mainmodule:
    def __init__(self):
        hosp = HospitalServiceImpl()
        hosp.get_appointment_by_id(3)
        try:
            hosp.get_appointments_for_patient(21)
        except PatientNumberNotFoundException as e:
            print(e)
        hosp.get_appointments_for_doctor(2)
        appointnew = Appointments(10, 1, 3, "2024-10-10", "come on time")
        hosp.schedule_appointment(appointnew)
        appointupdate = Appointments(1, 1, 1, "2024-05-01", "updated
appointment reports")
        hosp.update_appointment(appointupdate)
        hosp.cancel_appointment(1)

obj=Mainmodule()
```

```
Connection successful
Appointment ID: 3
Patient ID: 1
Doctor ID: 1
Appointment Date: 2024-05-03
Description: Drink Water before test
Patient with number 21 not found in the database
Appointment ID: 6
Patient ID: 1
Doctor ID: 2
Appointment Date: 2024-07-01
Description: bring test reports
scheduled success
Updated Successfully
Appointment Cancelled

Process finished with exit code 0
```