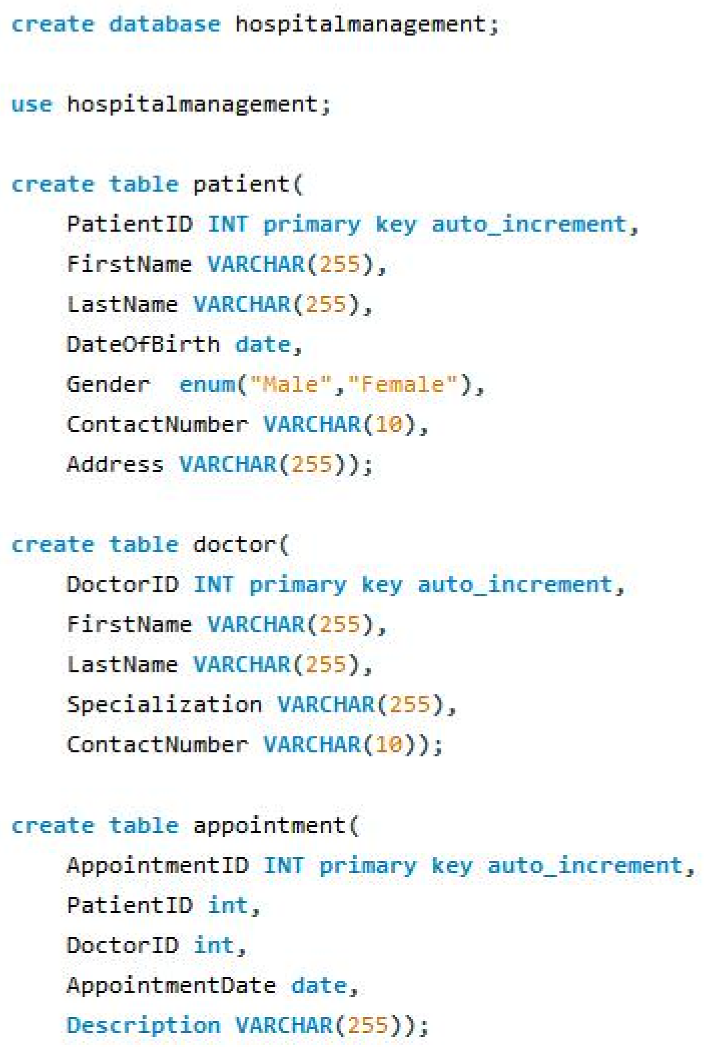
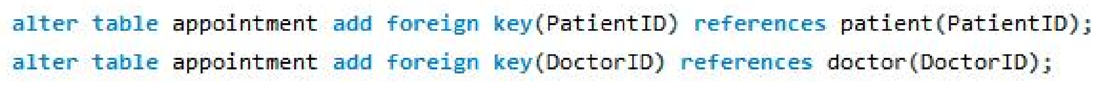
Python Coding Challenge - Hospital Management System

By Dhanush A

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



```
create database hospitalmanagement;

use hospitalmanagement;

create table patient(
    PatientID INT primary key auto_increment,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    DateOfBirth date,
    Gender  enum("Male","Female"),
    ContactNumber VARCHAR(10),
    Address VARCHAR(255));

create table doctor(
    DoctorID INT primary key auto_increment,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Specialization VARCHAR(255),
    ContactNumber VARCHAR(10));

create table appointment(
    AppointmentID INT primary key auto_increment,
    PatientID int,
    DoctorID int,
    AppointmentDate date,
    Description VARCHAR(255));
```



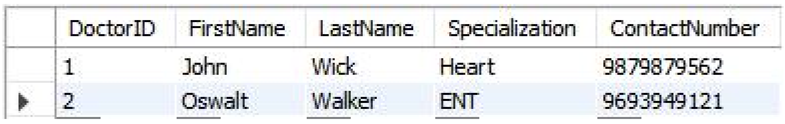
```
alter table appointment add foreign key(PatientID) references patient(PatientID);
alter table appointment add foreign key(DoctorID) references doctor(DoctorID);
```

```sql
INSERT INTO `doctor` (`DoctorID`, `FirstName`, `LastName`, `Specialization`, `ContactNumber`)
 VALUES ('1', 'John', 'Wick', 'Heart', '9879879562');
INSERT INTO `doctor` (`DoctorID`, `FirstName`, `LastName`, `Specialization`, `ContactNumber`)
 VALUES ('2', 'Oswalt', 'Walker', 'ENT', '9693949121');
```

```sql
INSERT INTO `patient` (`PatientID`, `FirstName`, `LastName`, `DateOfBirth`, `Gender`, `ContactNumber`, `Address`)
 VALUES ('1', 'John', 'Doe', '1990-05-15', 'Male', '1234567890', '456 Oak Street, Cityville');
INSERT INTO `patient` (`PatientID`, `FirstName`, `LastName`, `DateOfBirth`, `Gender`, `ContactNumber`, `Address`)
 VALUES ('2', 'Alice', 'Smith', '1985-08-22', 'Female', '9876543210', '789 Maple Avenue, Townsville');
INSERT INTO `patient` (`PatientID`, `FirstName`, `LastName`, `DateOfBirth`, `Gender`, `ContactNumber`, `Address`)
 VALUES ('3', 'Robert', 'Johnson', '1977-03-10', 'Male', '5553337777', '123 Pine Street, Villagetown');
```

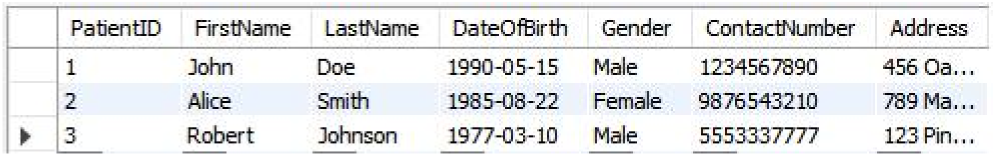
```sql
INSERT INTO `appointment` (`AppointmentID`, `PatientID`, `DoctorID`, `AppointmentDate`, `Description`)
VALUES ('1', '1', '1', '2024-02-15', 'General Checkup');
INSERT INTO `appointment` (`AppointmentID`, `PatientID`, `DoctorID`, `AppointmentDate`, `Description`)
VALUES ('2', '2', '2', '2024-02-20', 'Dental Cleaning');
INSERT INTO `appointment` (`AppointmentID`, `PatientID`, `DoctorID`, `AppointmentDate`, `Description`)
VALUES ('3', '3', '1', '2024-02-25', 'Cardiology Consultation');
INSERT INTO `appointment` (`AppointmentID`, `PatientID`, `DoctorID`, `AppointmentDate`, `Description`)
VALUES ('4', '1', '2', '2024-03-05', 'Eye Exam');
INSERT INTO `appointment` (`AppointmentID`, `PatientID`, `DoctorID`, `AppointmentDate`, `Description`)
VALUES ('5', '3', '1', '2024-03-10', 'Orthopedic Consultation');
```

Doctor Table:



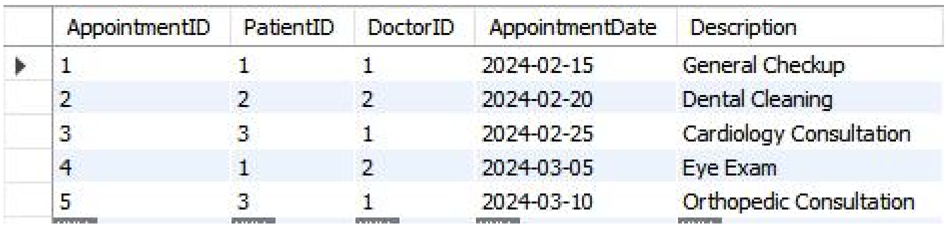
DoctorID	FirstName	LastName	Specialization	ContactNumber
1	John	Wick	Heart	9879879562
2	Oswalt	Walker	ENT	9693949121

Patient Table :



PatientID	FirstName	LastName	DateOfBirth	Gender	ContactNumber	Address
1	John	Doe	1990-05-15	Male	1234567890	456 Oa...
2	Alice	Smith	1985-08-22	Female	9876543210	789 Ma...
3	Robert	Johnson	1977-03-10	Male	5553337777	123 Pin...

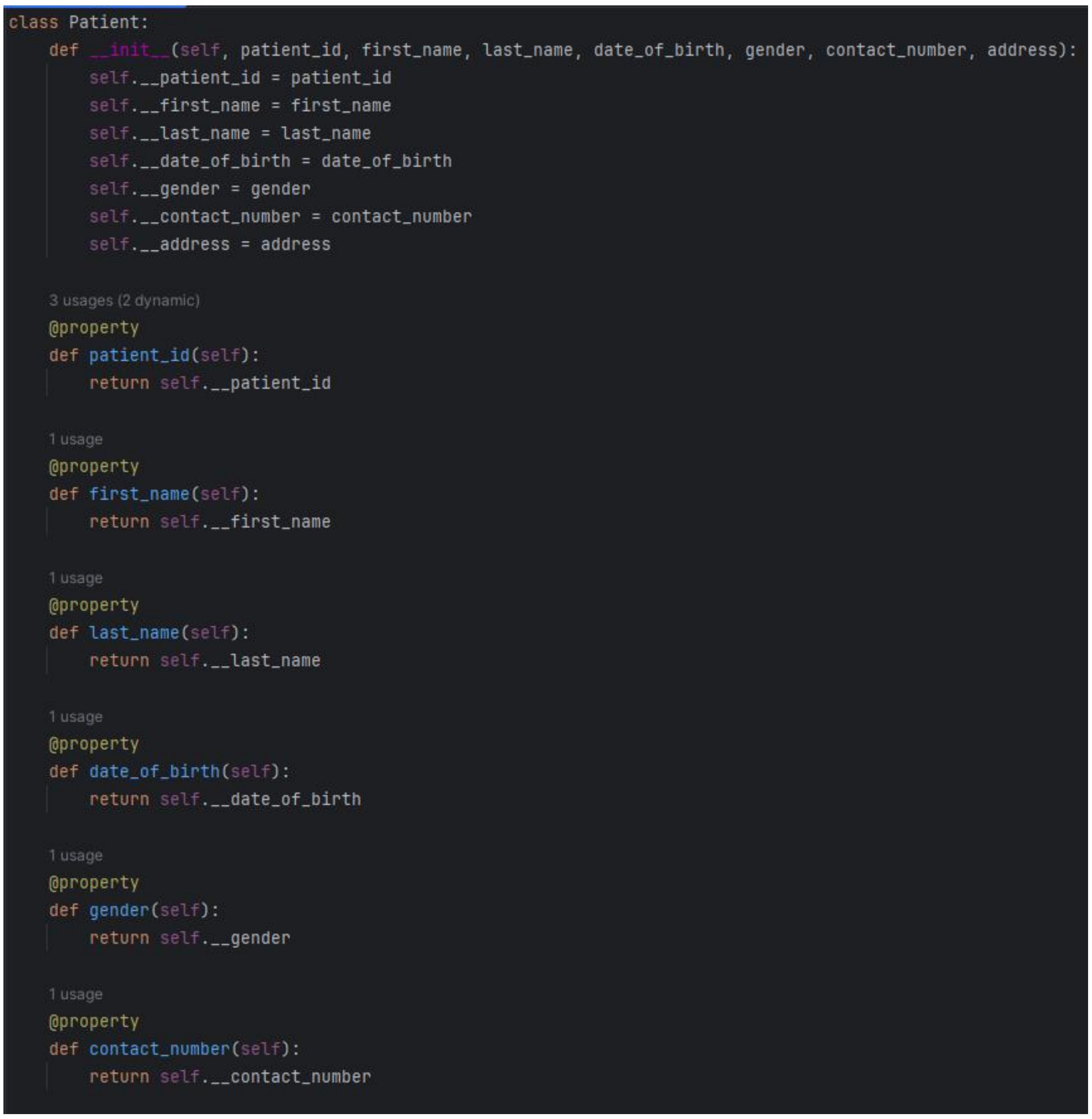
Appointment Table :



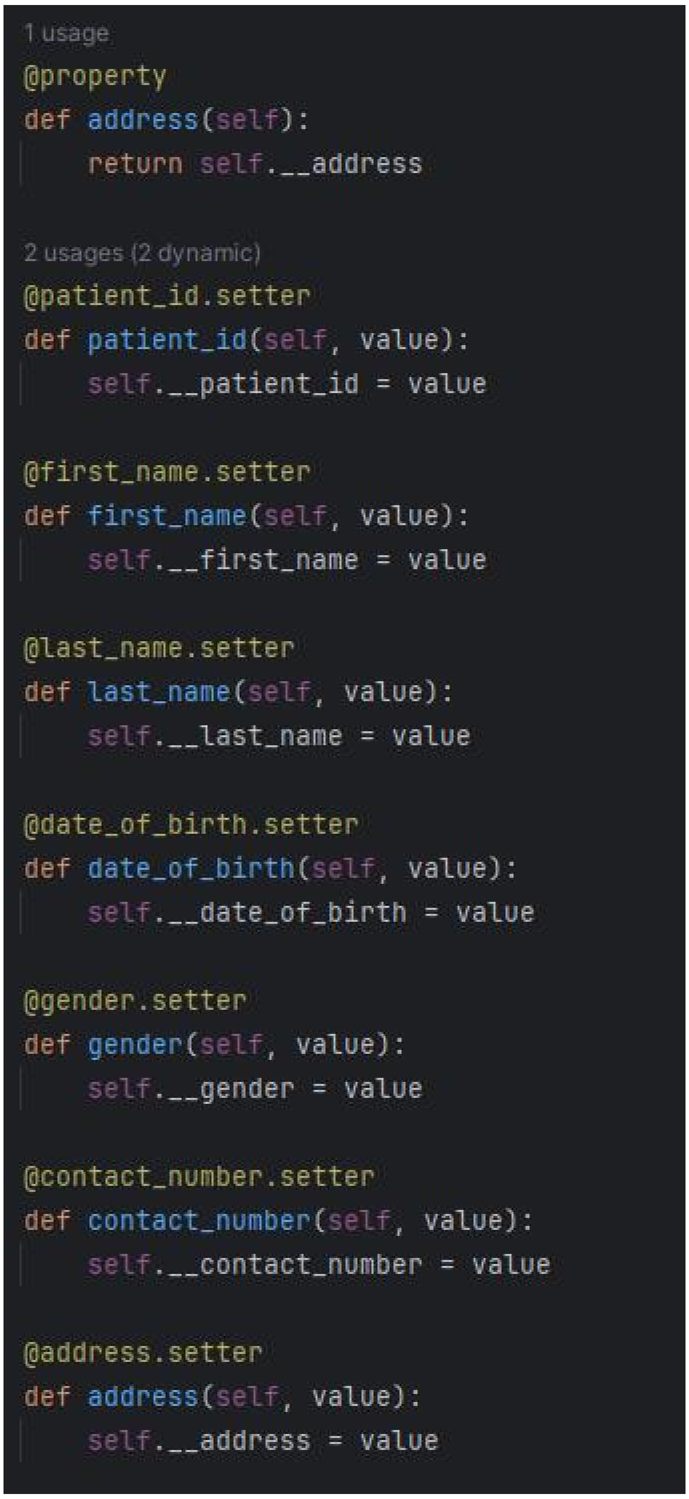
AppointmentID	PatientID	DoctorID	AppointmentDate	Description
1	1	1	2024-02-15	General Checkup
2	2	2	2024-02-20	Dental Cleaning
3	3	1	2024-02-25	Cardiology Consultation
4	1	2	2024-03-05	Eye Exam
5	3	1	2024-03-10	Orthopedic Consultation

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;



```
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

    3 usages (2 dynamic)
    @property
    def patient_id(self):
        return self.__patient_id

    1 usage
    @property
    def first_name(self):
        return self.__first_name

    1 usage
    @property
    def last_name(self):
        return self.__last_name

    1 usage
    @property
    def date_of_birth(self):
        return self.__date_of_birth

    1 usage
    @property
    def gender(self):
        return self.__gender

    1 usage
    @property
    def contact_number(self):
        return self.__contact_number
```



```
1 usage
@property
def address(self):
    return self.__address

2 usages (2 dynamic)
@patient_id.setter
def patient_id(self, value):
    self.__patient_id = value

@first_name.setter
def first_name(self, value):
    self.__first_name = value

@last_name.setter
def last_name(self, value):
    self.__last_name = value

@date_of_birth.setter
def date_of_birth(self, value):
    self.__date_of_birth = value

@gender.setter
def gender(self, value):
    self.__gender = value

@contact_number.setter
def contact_number(self, value):
    self.__contact_number = value

@address.setter
def address(self, value):
    self.__address = value
```

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

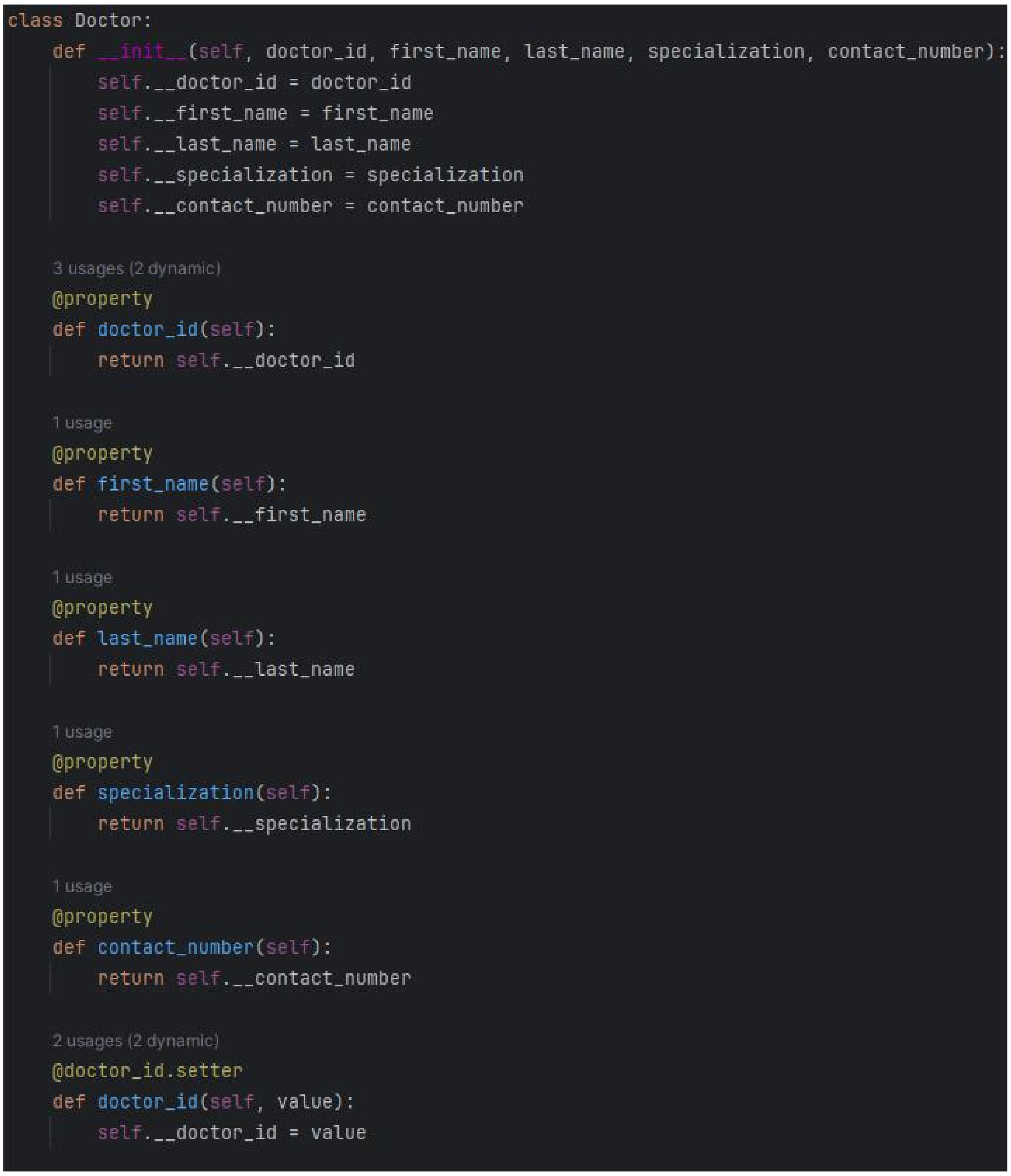
a. doctorId

b. firstName

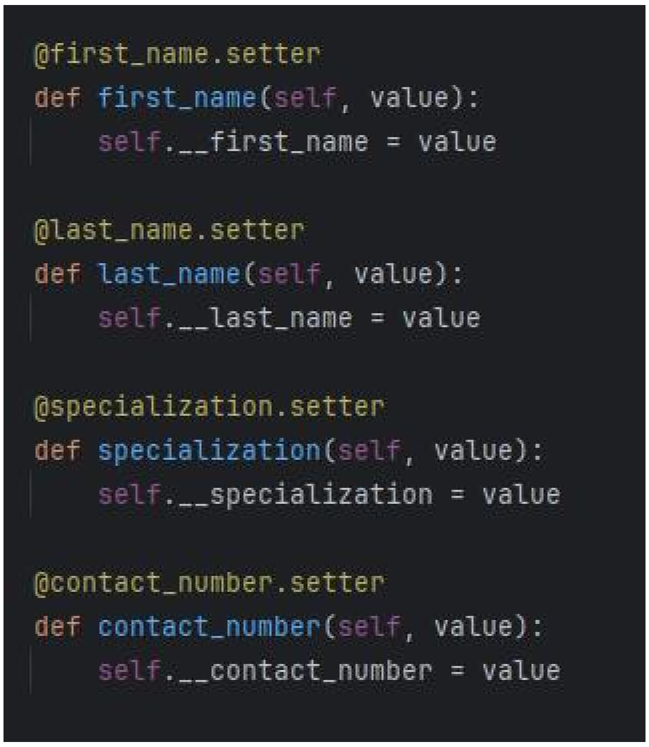
c. lastName

d. specialization

e. contactNumber;



```
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

    3 usages (2 dynamic)
    @property
    def doctor_id(self):
        return self.__doctor_id

    1 usage
    @property
    def first_name(self):
        return self.__first_name

    1 usage
    @property
    def last_name(self):
        return self.__last_name

    1 usage
    @property
    def specialization(self):
        return self.__specialization

    1 usage
    @property
    def contact_number(self):
        return self.__contact_number

    2 usages (2 dynamic)
    @doctor_id.setter
    def doctor_id(self, value):
        self.__doctor_id = value
```



```python
@first_name.setter
def first_name(self, value):
    self.__first_name = value

@last_name.setter
def last_name(self, value):
    self.__last_name = value

@specialization.setter
def specialization(self, value):
    self.__specialization = value

@contact_number.setter
def contact_number(self, value):
    self.__contact_number = value
```

3. Appointment Class:

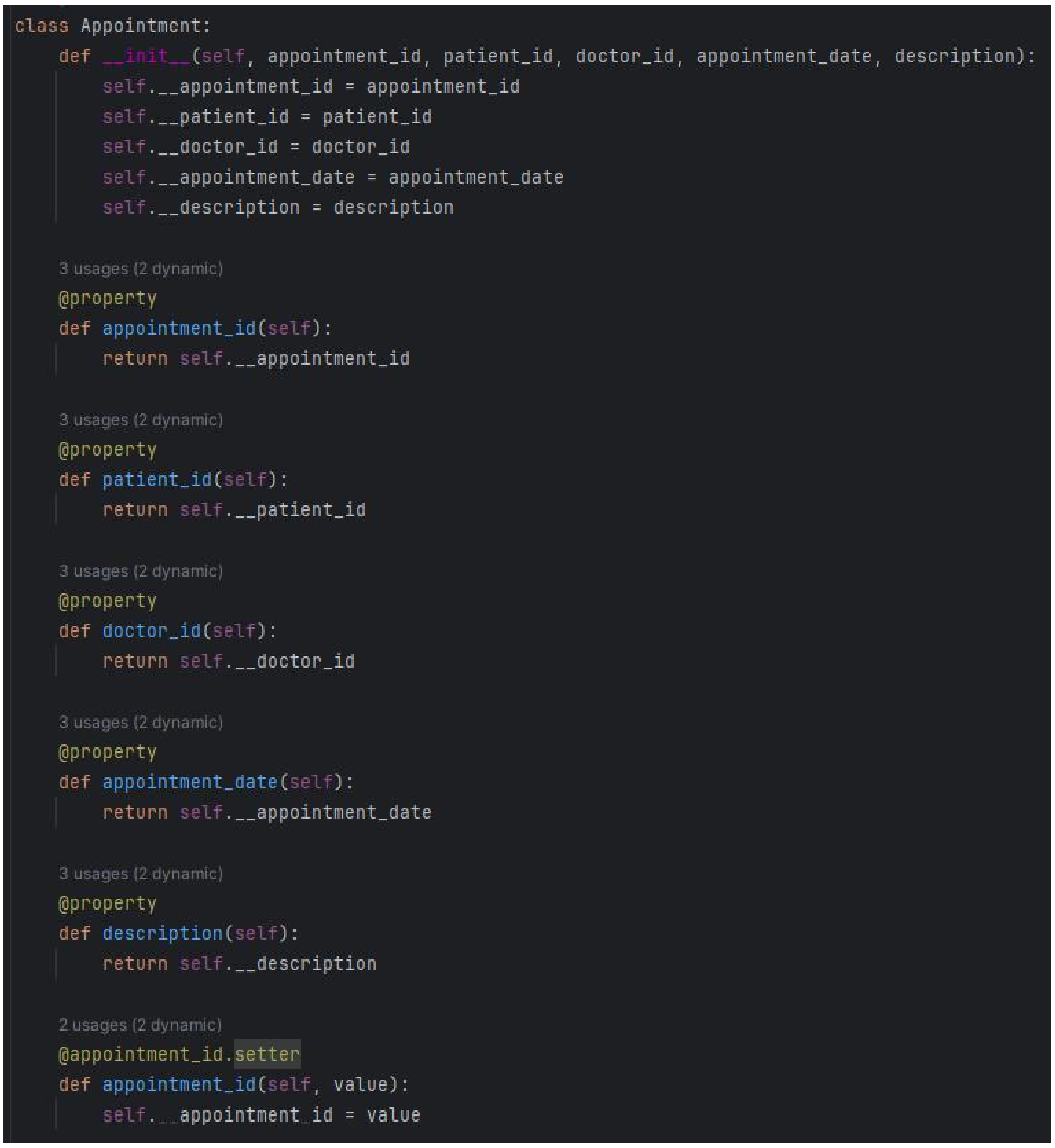
a. appointmentId

b. patientId

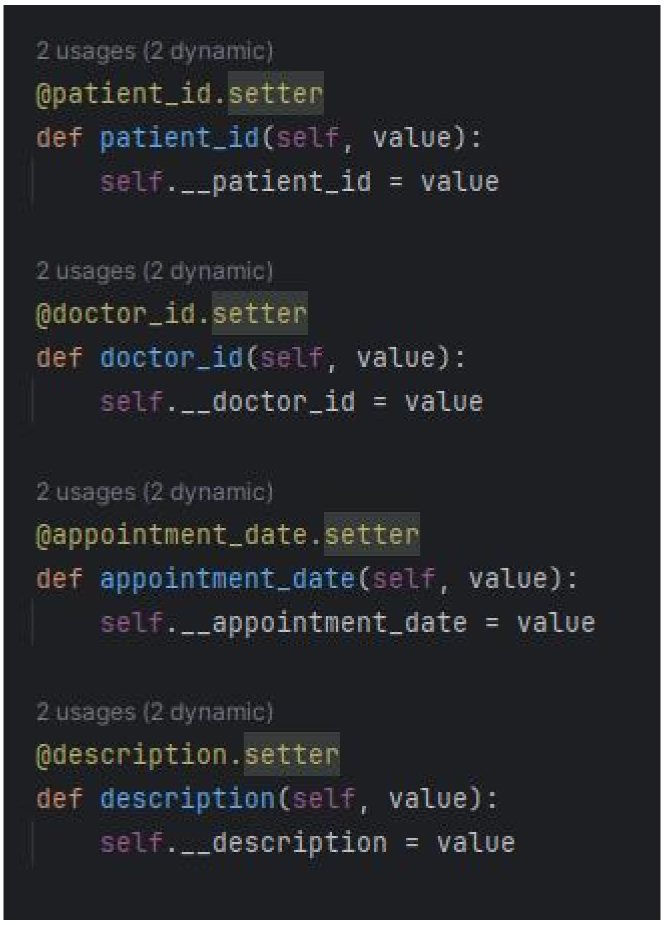
c. doctorId

d. appointmentDate

e. description

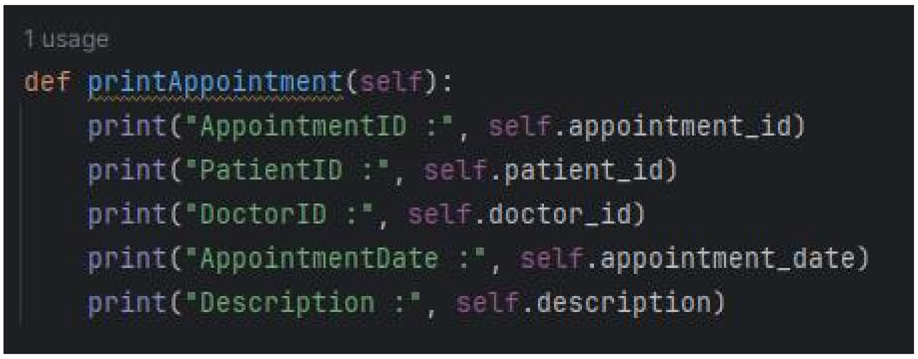


```
class Appointment:
    def __init__(self, appointment_id, patient_id, doctor_id, appointment_date, description):
        self.__appointment_id = appointment_id
        self.__patient_id = patient_id
        self.__doctor_id = doctor_id
        self.__appointment_date = appointment_date
        self.__description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

    3 usages (2 dynamic)
    @property
    def appointment_id(self):
        return self.__appointment_id

    3 usages (2 dynamic)
    @property
    def patient_id(self):
        return self.__patient_id

    3 usages (2 dynamic)
    @property
    def doctor_id(self):
        return self.__doctor_id

    3 usages (2 dynamic)
    @property
    def appointment_date(self):
        return self.__appointment_date

    3 usages (2 dynamic)
    @property
    def description(self):
        return self.__description

    2 usages (2 dynamic)
    @appointment_id.setter
    def appointment_id(self, value):
        self.__appointment_id = value
```



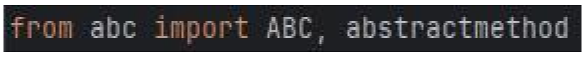
```
2 usages (2 dynamic)
@patient_id.setter
def patient_id(self, value):
    self.__patient_id = value

2 usages (2 dynamic)
@doctor_id.setter
def doctor_id(self, value):
    self.__doctor_id = value

2 usages (2 dynamic)
@appointment_date.setter
def appointment_date(self, value):
    self.__appointment_date = value

2 usages (2 dynamic)
@description.setter
def description(self, value):
    self.__description = value
```

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.

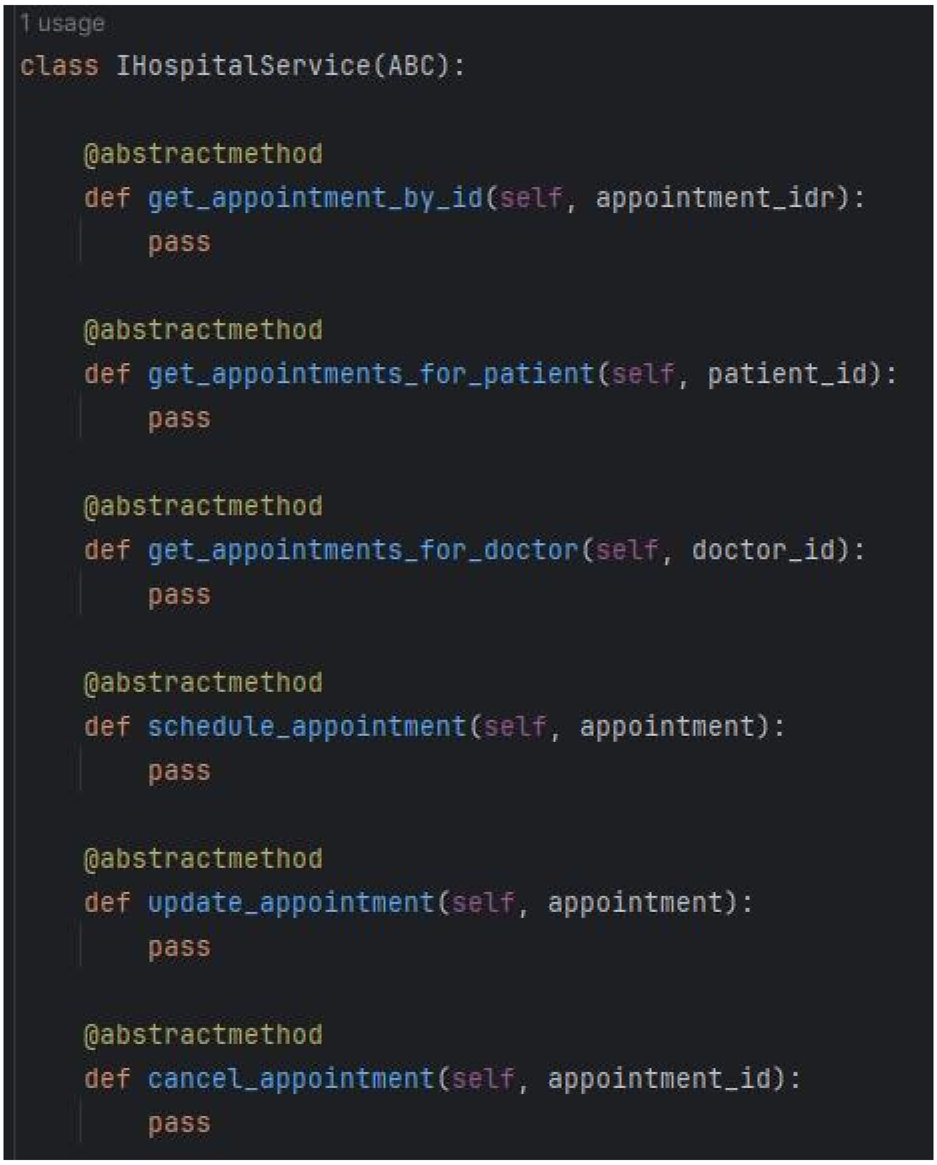


```
1 usage
def printAppointment(self):
    print("AppointmentID :", self.appointment_id)
    print("PatientID :", self.patient_id)
    print("DoctorID :", self.doctor_id)
    print("AppointmentDate :", self.appointment_date)
    print("Description :", self.description)
```

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

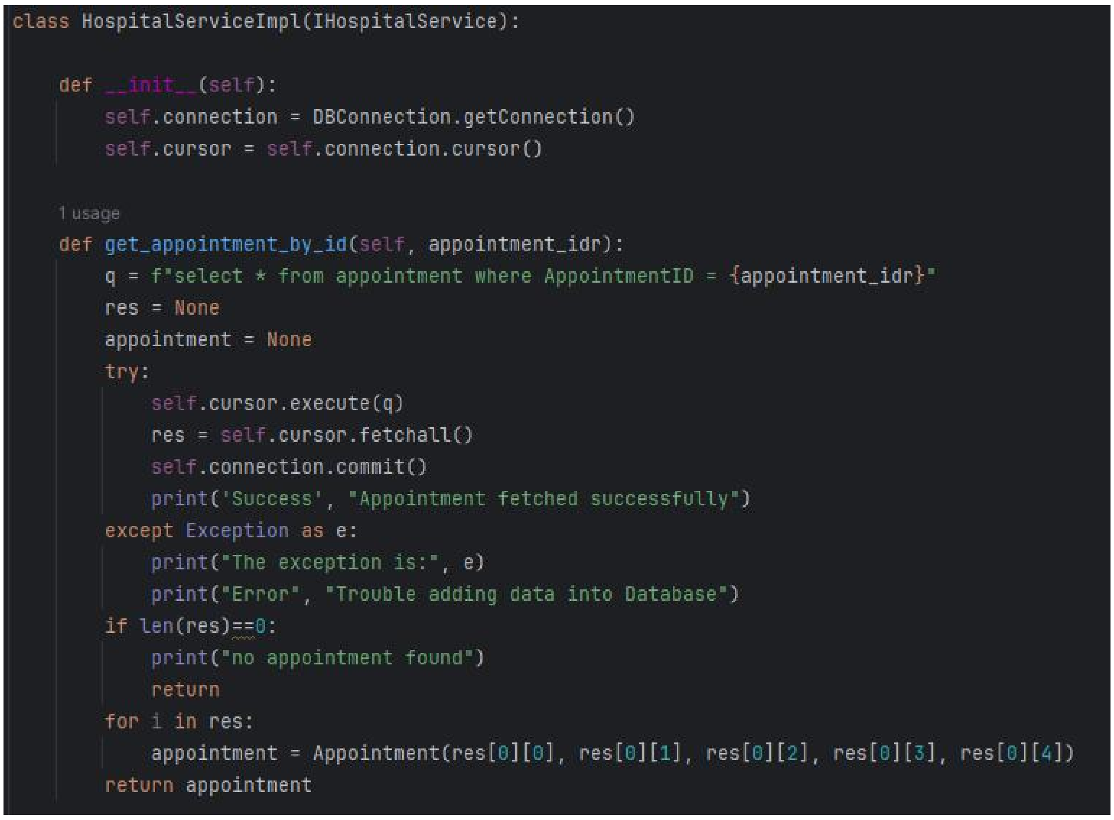


```
from abc import ABC, abstractmethod
```

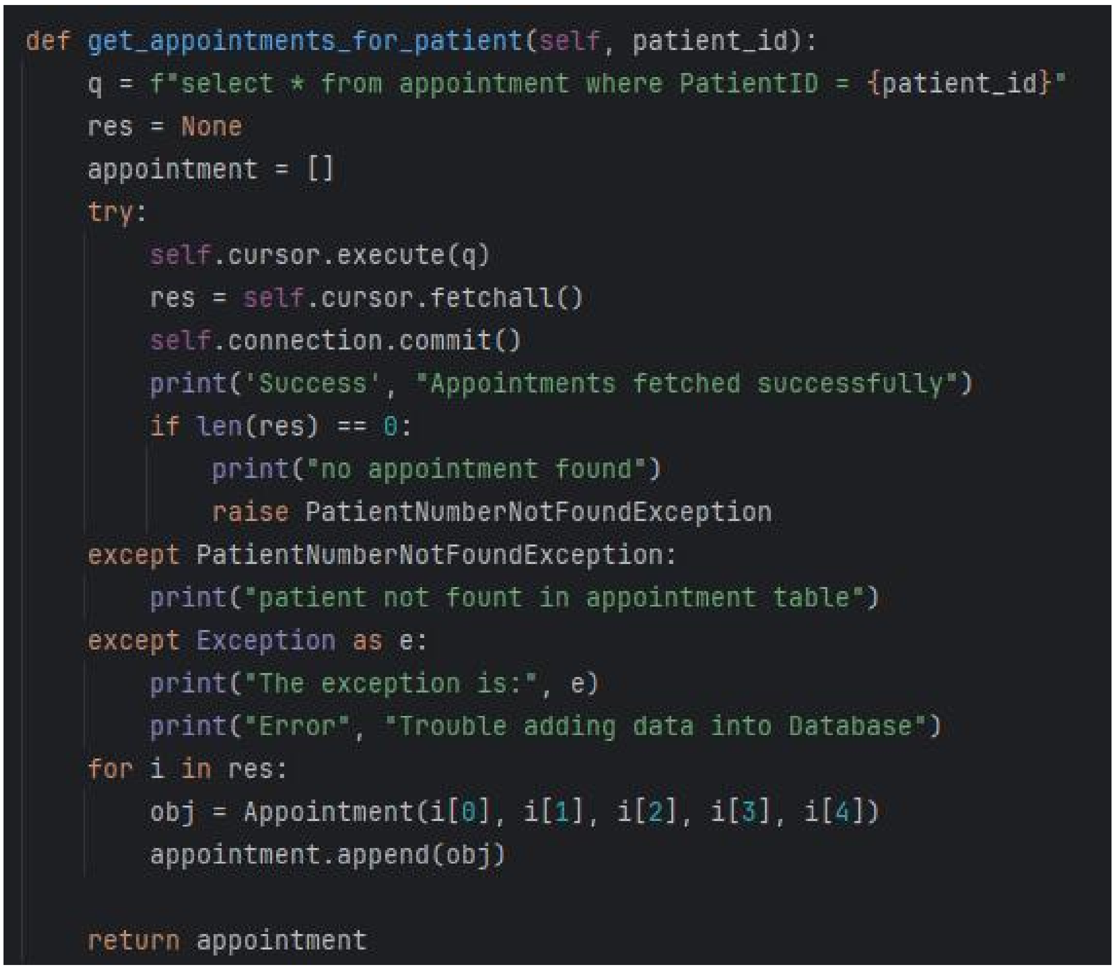


```
1 usage
class IHospitalService(ABC):

    @abstractmethod
    def get_appointment_by_id(self, appointment_idr):
        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):
        pass

    @abstractmethod
    def update_appointment(self, appointment):
        pass

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

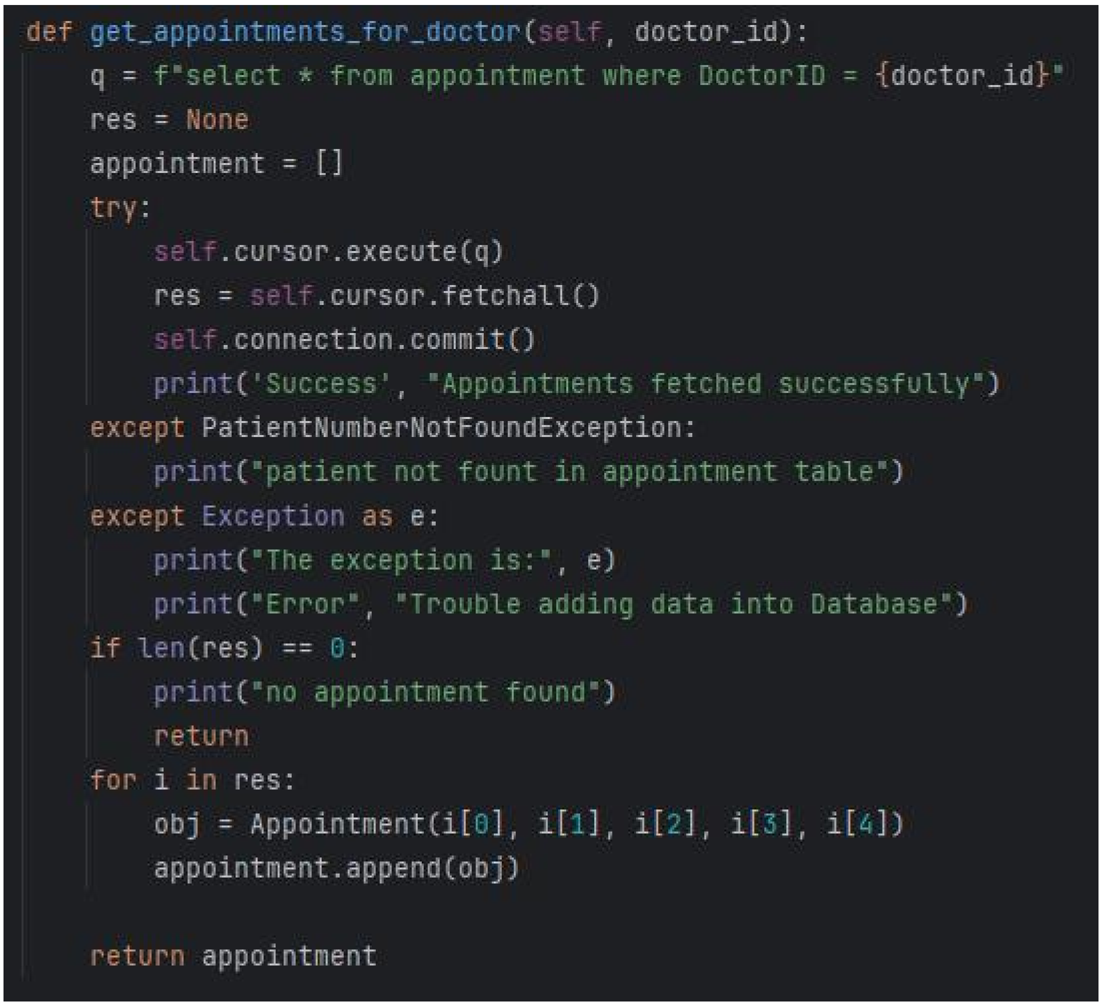
6. Define **HospitalServiceImpl** class and implement all the methods **IHospitalServiceImpl .**



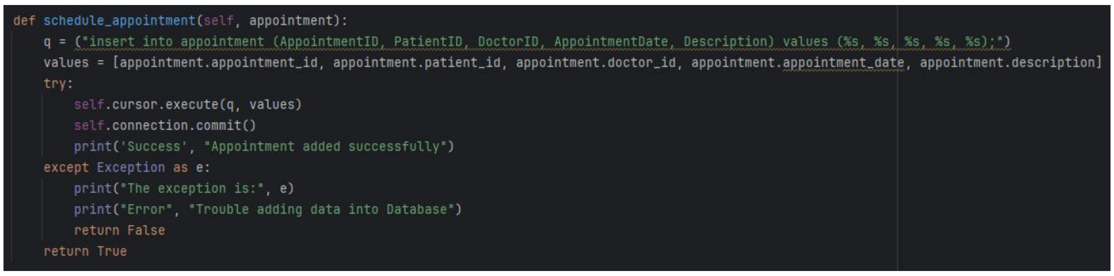
```
class HospitalServiceImpl(IHospitalService):

    def __init__(self):
        self.connection = DBConnection.getConnection()
        self.cursor = self.connection.cursor()

    1 usage
    def get_appointment_by_id(self, appointment_idr):
        q = f"select * from appointment where AppointmentID = {appointment_idr}"
        res = None
        appointment = None
        try:
            self.cursor.execute(q)
            res = self.cursor.fetchall()
            self.connection.commit()
            print('Success', "Appointment fetched successfully")
        except Exception as e:
            print("The exception is:", e)
            print("Error", "Trouble adding data into Database")
        if len(res)==0:
            print("no appointment found")
            return
        for i in res:
            appointment = Appointment(res[0][0], res[0][1], res[0][2], res[0][3], res[0][4])
        return appointment
```



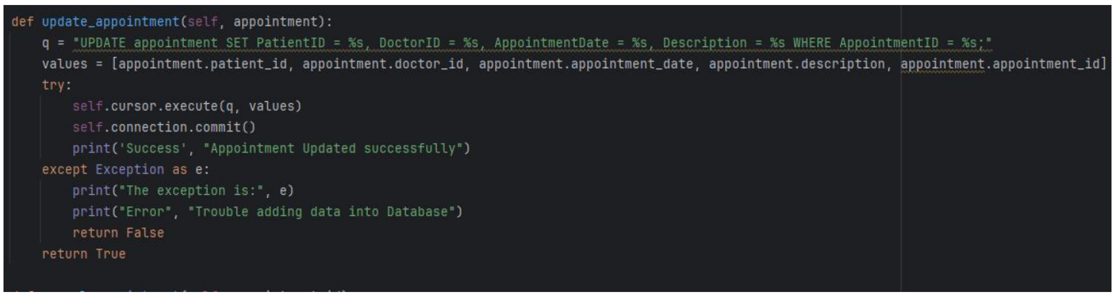
```python
def get_appointments_for_patient(self, patient_id):
    q = f"select * from appointment where PatientID = {patient_id}"
    res = None
    appointment = []
    try:
        self.cursor.execute(q)
        res = self.cursor.fetchall()
        self.connection.commit()
        print('Success', "Appointments fetched successfully")
        if len(res) == 0:
            print("no appointment found")
            raise PatientNumberNotFoundException
    except PatientNumberNotFoundException:
        print("patient not fount in appointment table")
    except Exception as e:
        print("The exception is:", e)
        print("Error", "Trouble adding data into Database")
    for i in res:
        obj = Appointment(i[0], i[1], i[2], i[3], i[4])
        appointment.append(obj)

    return appointment
```



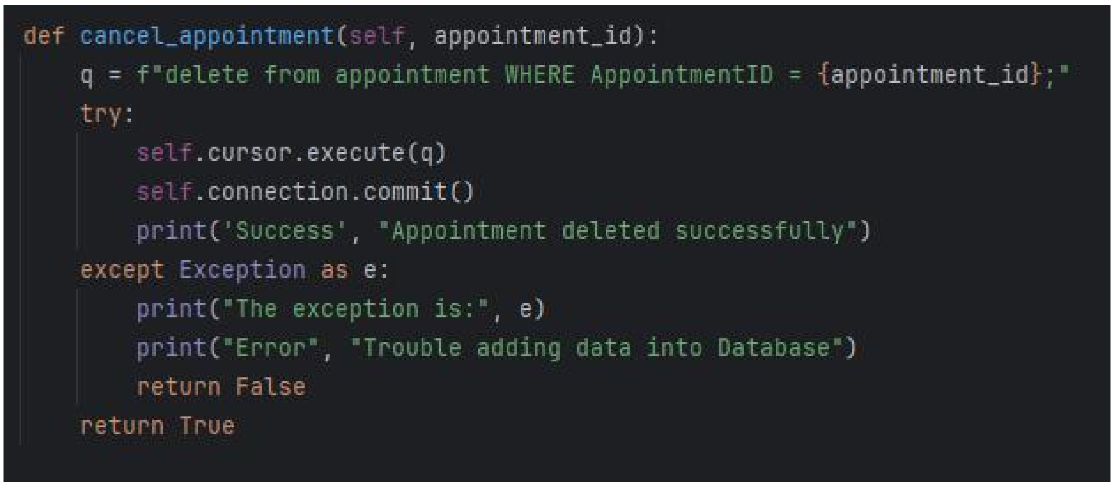
```python
def get_appointments_for_doctor(self, doctor_id):
    q = f"select * from appointment where DoctorID = {doctor_id}"
    res = None
    appointment = []
    try:
        self.cursor.execute(q)
        res = self.cursor.fetchall()
        self.connection.commit()
        print('Success', "Appointments fetched successfully")
    except PatientNumberNotFoundException:
        print("patient not fount in appointment table")
    except Exception as e:
        print("The exception is:", e)
        print("Error", "Trouble adding data into Database")
    if len(res) == 0:
        print("no appointment found")
        return
    for i in res:
        obj = Appointment(i[0], i[1], i[2], i[3], i[4])
        appointment.append(obj)

    return appointment
```



```python
def schedule_appointment(self, appointment):
    q = ("insert into appointment (AppointmentID, PatientID, DoctorID, AppointmentDate, Description) values (%s, %s, %s, %s, %s);")
    values = [appointment.appointment_id, appointment.patient_id, appointment.doctor_id, appointment.appointment_date, appointment.description]
    try:
        self.cursor.execute(q, values)
        self.connection.commit()
        print('Success', "Appointment added successfully")
    except Exception as e:
        print("The exception is:", e)
        print("Error", "Trouble adding data into Database")
        return False
    return True
```

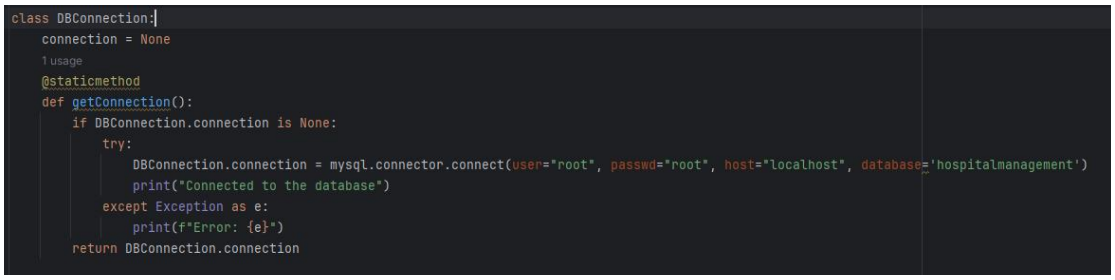


```python
def update_appointment(self, appointment):
    q = "UPDATE appointment SET PatientID = %s, DoctorID = %s, AppointmentDate = %s, Description = %s WHERE AppointmentID = %s;"
    values = [appointment.patient_id, appointment.doctor_id, appointment.appointment_date, appointment.description, appointment.appointment_id]
    try:
        self.cursor.execute(q, values)
        self.connection.commit()
        print('Success', "Appointment Updated successfully")
    except Exception as e:
        print("The exception is:", e)
        print("Error", "Trouble adding data into Database")
        return False
    return True
```



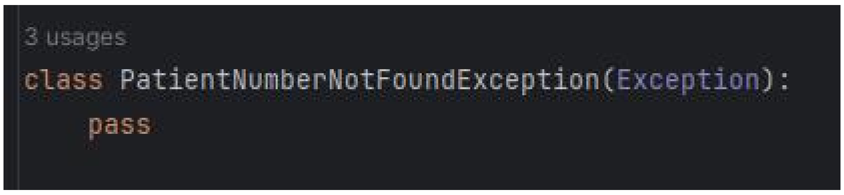
```python
def cancel_appointment(self, appointment_id):
    q = f"delete from appointment WHERE AppointmentID = {appointment_id};"
    try:
        self.cursor.execute(q)
        self.connection.commit()
        print('Success', "Appointment deleted successfully")
    except Exception as e:
        print("The exception is:", e)
        print("Error", "Trouble adding data into Database")
        return False
    return True
```

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.



```python
class DBConnection:
    connection = None
    1 usage
    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            try:
                DBConnection.connection = mysql.connector.connect(user="root", passwd="root", host="localhost", database='hospitalmanagement')
                print("Connected to the database")
            except Exception as e:
                print(f"Error: {e}")
        return DBConnection.connection
```

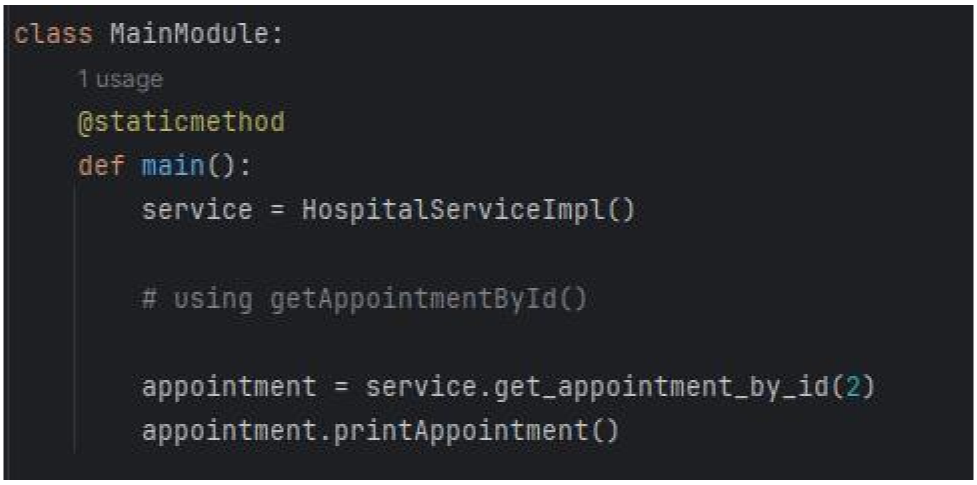
8. Create the exceptions in package myexceptions
Define the following custom exceptions and throw them in methods whenever needed. Handle all the
exceptions in main method,
1. **PatientNumberNotFoundException** :throw this exception when user enters an invalid patient
number which doesn't exist in db



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

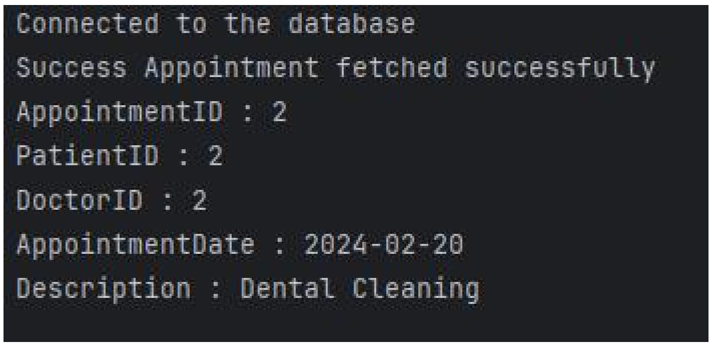
9. Create class named MainModule with main method in package mainmod.
Trigger all the methods in service implementation class.

using getAppointmentById()



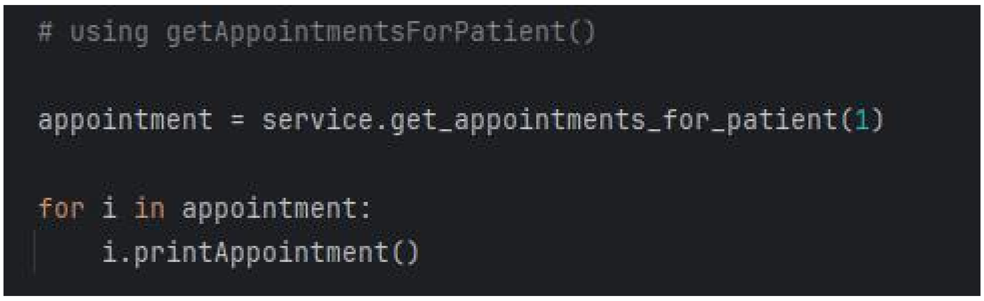
```
class MainModule:
    1 usage
    @staticmethod
    def main():
        service = HospitalServiceImpl()

        # using getAppointmentById()

        appointment = service.get_appointment_by_id(2)
        appointment.printAppointment()
```

Output :



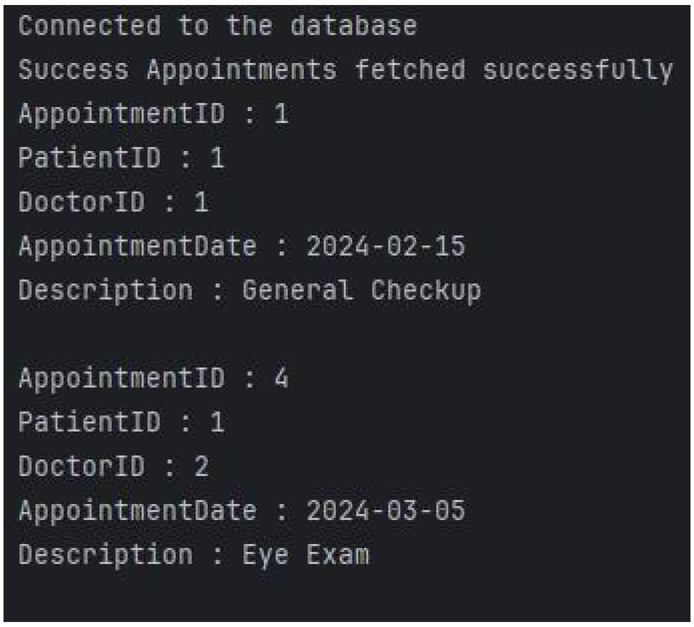
```
Connected to the database
Success Appointment fetched successfully
AppointmentID : 2
PatientID : 2
DoctorID : 2
AppointmentDate : 2024-02-20
Description : Dental Cleaning
```

using getAppointmentsForPatient()



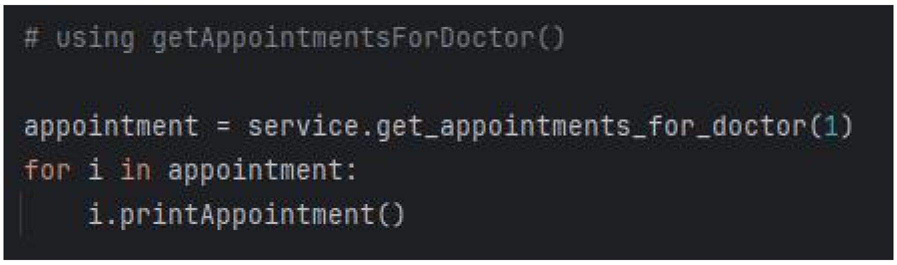
```python
# using getAppointmentsForPatient()

appointment = service.get_appointments_for_patient(1)

for i in appointment:
    i.printAppointment()
```

Output :



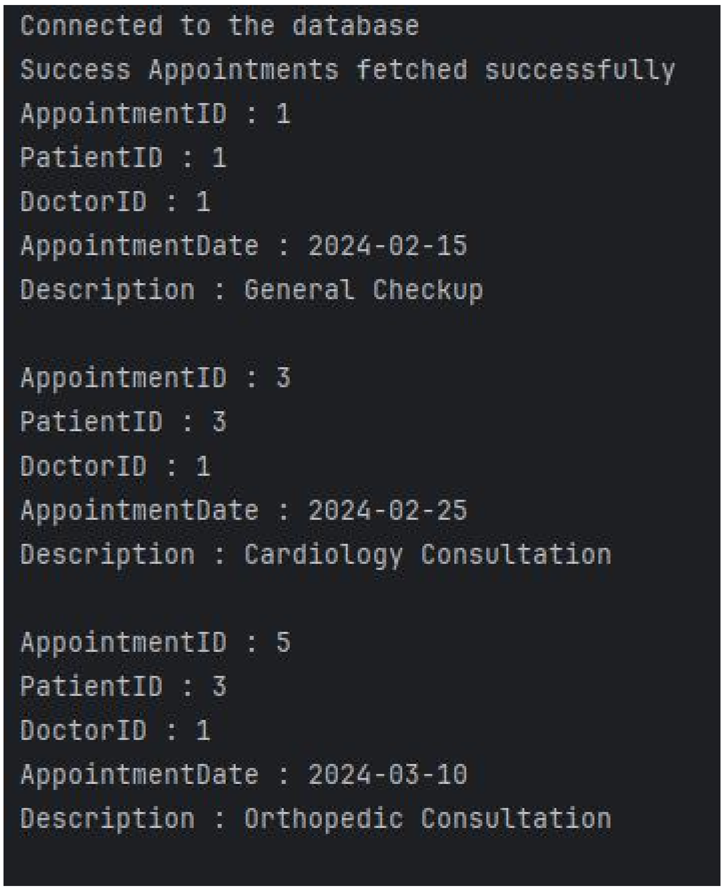
```
Connected to the database
Success Appointments fetched successfully
AppointmentID : 1
PatientID : 1
DoctorID : 1
AppointmentDate : 2024-02-15
Description : General Checkup

AppointmentID : 4
PatientID : 1
DoctorID : 2
AppointmentDate : 2024-03-05
Description : Eye Exam
```

using getAppointmentsForDoctor()



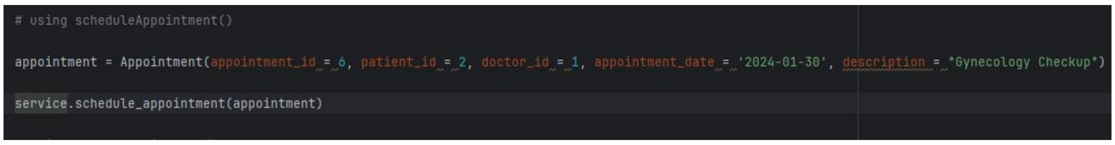
```python
# using getAppointmentsForDoctor()

appointment = service.get_appointments_for_doctor(1)
for i in appointment:
    i.printAppointment()
```

Output :



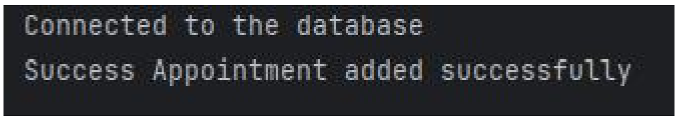
```
Connected to the database
Success Appointments fetched successfully
AppointmentID : 1
PatientID : 1
DoctorID : 1
AppointmentDate : 2024-02-15
Description : General Checkup

AppointmentID : 3
PatientID : 3
DoctorID : 1
AppointmentDate : 2024-02-25
Description : Cardiology Consultation

AppointmentID : 5
PatientID : 3
DoctorID : 1
AppointmentDate : 2024-03-10
Description : Orthopedic Consultation
```

using scheduleAppointment()



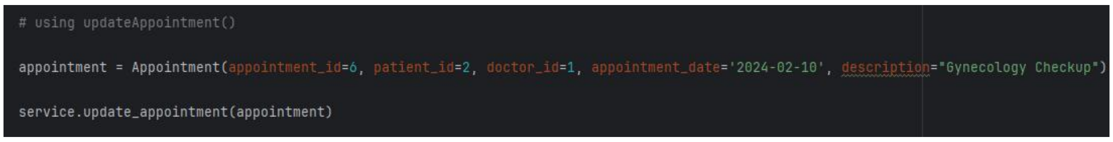
```
# using scheduleAppointment()

appointment = Appointment(appointment_id = 6, patient_id = 2, doctor_id = 1, appointment_date = '2024-01-30', description = "Gynecology Checkup")

service.schedule_appointment(appointment)
```

Output :



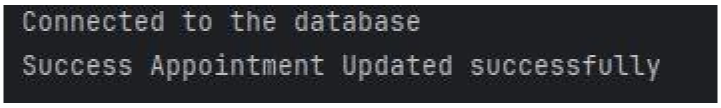
```
Connected to the database
Success Appointment added successfully
```

using updateAppointment()



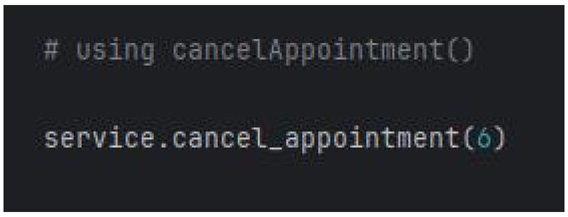
```
# using updateAppointment()

appointment = Appointment(appointment_id=6, patient_id=2, doctor_id=1, appointment_date='2024-02-10', description="Gynecology Checkup")

service.update_appointment(appointment)
```

Output :



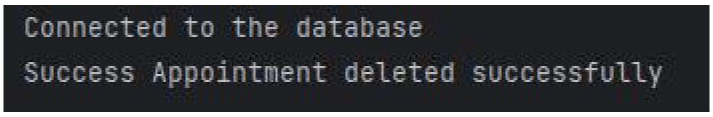
```
Connected to the database
Success Appointment Updated successfully
```

using cancelAppointment()



```
# using cancelAppointment()

service.cancel_appointment(6)
```

Output :



```
Connected to the database
Success Appointment deleted successfully
```