HOSPITAL MANAGEMENT SYSTEM

Name : VIKRAM M.

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

```
create database Host;
use Host;
```

1. Define `Patient` class with the following confidential attributes:
a. patientId
b. firstName
c. lastName;
d. dateOfBirth
e. gender
f. contactNumber
g.address

```
CREATE TABLE Patient (
    patientId INT PRIMARY KEY,
    firstName VARCHAR(250),
    lastName VARCHAR(250),
    dateOfBirth DATE,
    gender VARCHAR(10),
    contactNumber VARCHAR(20),
    address VARCHAR(255)
);
```

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

```
CREATE TABLE Doctor (
    doctorId INT PRIMARY KEY,
    firstName VARCHAR(255),
    lastName VARCHAR(255),
    specialization VARCHAR(255),
    contactNumber VARCHAR(20)
);
```

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

```
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)
);

insert into patient values(001, 'Emily', 'Johnson', '1999-12-15', 'Female',
'9876543210', '123 Main Street CA');
insert into patient values(002, 'Sophia', 'Williams', '2003-05-20', 'Female',
'9765432109', '456 Oak Avenue IL');

insert into Doctor values(333, 'Jessica', 'Smith', 'Dermatologist', '9258347098');
insert into Doctor values(444, 'Michael', 'Johnson', 'Orthopedic Surgeon',
'9456712308');

insert into appointment values(51234,001,333,'2024-03-11','Regular Checkup');
insert into appointment values(51235,002,444,'2024-03-12','Endoscopy');
```

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.

For Appointment class,

```
class Appointment:



   def __init__(self):
       self.__appointmentId = None
       self.__patientId = None
       self.__doctorId = None
       self.__appointmentDate = None
       self.__description = None



   def __init__(self, appointmentId, patientId, doctorId,
appointmentDate, description):
       self.__appointmentId = appointmentId
       self.__patientId = patientId
       self.__doctorId = doctorId
       self.__appointmentDate = appointmentDate
       self.__description = description



   # Getter methods
   def getAppointmentId(self):
```

```
        return self.__appointmentId


    def getPatientId(self):
        return self.__patientId


    def getDoctorId(self):
        return self.__doctorId


    def getAppointmentDate(self):
        return self.__appointmentDate


    def getDescription(self):
        return self.__description

    def setAppointmentId(self, appointmentId):
        self.__appointmentId = appointmentId


    def setPatientId(self, patientId):
        self.__patientId = patientId


    def setDoctorId(self, doctorId):
        self.__doctorId = doctorId


    def setAppointmentDate(self, appointmentDate):
        self.__appointmentDate = appointmentDate


    def setDescription(self, description):
        self.__description = description


    def __str__(self):
        return f"Appointment ID: {self.__appointmentId}, Patient ID:
{self.__patientId}, Doctor ID: {self.__doctorId}, Date:
{self.__appointmentDate}, Description: {self.__description}"
```

```
    def printDetails(self):
        print(f"Appointment ID: {self.__appointmentId}")
        print(f"Patient ID: {self.__patientId}")
        print(f"Doctor ID: {self.__doctorId}")
        print(f"Date: {self.__appointmentDate}")
        print(f"Description: {self.__description}")
```

For Patient class,

```
class Patient:

    def __init__(self):
        self.__patientId = None
        self.__firstName = None
        self.__lastName = None
        self.__dateOfBirth = None
        self.__gender = None
        self.__contactNumber = None
        self.__address = None

    def __init__(self, patientId, firstName, lastName, dateOfBirth,
gender, contactNumber, address):
        self.__patientId = patientId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__dateOfBirth = dateOfBirth
        self.__gender = gender
        self.__contactNumber = contactNumber
        self.__address = address

    def getPatientId(self):
        return self.__patientId


    def getFirstName(self):
        return self.__firstName


    def getLastName(self):
        return self.__lastName
```

```
def getDateOfBirth(self):
    return self.__dateOfBirth


def getGender(self):
    return self.__gender


def getContactNumber(self):
    return self.__contactNumber


def getAddress(self):
    return self.__address


def setPatientId(self, patientId):
    self.__patientId = patientId


def setFirstName(self, firstName):
    self.__firstName = firstName


def setLastName(self, lastName):
    self.__lastName = lastName


def setDateOfBirth(self, dateOfBirth):
    self.__dateOfBirth = dateOfBirth


def setGender(self, gender):
    self.__gender = gender


def setContactNumber(self, contactNumber):
    self.__contactNumber = contactNumber


def setAddress(self, address):
    self.__address = address
```

```
    def __str__(self):
        return f"Patient ID: {self.__patientId}, Name:
{self.__firstName} {self.__lastName}, DOB: {self.__dateOfBirth},
Gender: {self.__gender}, Contact: {self.__contactNumber}, Address:
{self.__address}"

    def printDetails(self):
        print(f"Patient ID: {self.__patientId}")
        print(f"Name: {self.__firstName} {self.__lastName}")
        print(f"DOB: {self.__dateOfBirth}")
        print(f"Gender: {self.__gender}")
        print(f"Contact: {self.__contactNumber}")
        print(f"Address: {self.__address}")
```

For Doctor class,

```
class Doctor:


    def __init__(self):
        self.__doctorId = None
        self.__firstName = None
        self.__lastName = None
        self.__specialization = None
        self.__contactNumber = None


    def __init__(self, doctorId, firstName, lastName, specialization,
contactNumber):
        self.__doctorId = doctorId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__specialization = specialization
        self.__contactNumber = contactNumber

    def getDoctorId(self):
        return self.__doctorId


    def getFirstName(self):
```

```
        return self.__firstName


    def getLastName(self):
        return self.__lastName


    def getSpecialization(self):
        return self.__specialization


    def getContactNumber(self):
        return self.__contactNumber


    # Setter methods
    def setDoctorId(self, doctorId):
        self.__doctorId = doctorId


    def setFirstName(self, firstName):
        self.__firstName = firstName


    def setLastName(self, lastName):
        self.__lastName = lastName


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


    def setContactNumber(self, contactNumber):
        self.__contactNumber = contactNumber

    def __str__(self):
        return f"Doctor ID: {self.__doctorId}, Name: {self.__firstName}
{self.__lastName}, Specialization: {self.__specialization}, Contact:
{self.__contactNumber}"


    def printDetails(self):
        print(f"Doctor ID: {self.__doctorId}")
```

```
        print(f"Name: {self.__firstName} {self.__lastName}")
        print(f"Specialization: {self.__specialization}")
        print(f"Contact: {self.__contactNumber}")
```

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

```
from abc import ABC, abstractmethod
from entity.appointment import Appointment
from typing import List
class IHospitalService(ABC):
    @abstractmethod
    def getAppointmentById(self, appointmentId) -> Appointment:
        pass

    @abstractmethod
    def getAppointmentsForPatient(self, patientId) -> List[Appointment]:
        pass

    @abstractmethod
    def getAppointmentsForDoctor(self, doctorId) -> List[Appointment]:
        pass

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

    @abstractmethod
```

```
    def updateAppointment(self, appointment: Appointment) -> bool:
        pass

    @abstractmethod
    def cancelAppointment(self, appointmentId) -> bool:
        pass
```

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

```
import mysql.connector
from dao.ihospitalservice import IHospitalService
from entity.appointment import Appointment
from util.dbConnection import DBConnection
from exception.patient_exception import PatientNumberNotFoundException
from typing import List

class HospitalServiceImpl(IHospitalService):
    def __init__(self):
        self.conn = DBConnection.getConnection()
        self.cursor = self.conn.cursor()

    def __del__(self):
        self.conn.close()

    def getAppointmentById(self, appointmentId) -> Appointment:
        self.cursor.execute("SELECT * FROM appointment WHERE
appointmentId=?", (appointmentId,))
        result = self.cursor.fetchone()
        if result:
            appointment = Appointment(*result)
            return appointment
        return None

    def getAppointmentsForPatient(self, patientId) -> List[Appointment]:
        try:
            self.cursor.execute("SELECT * FROM appointment WHERE
patientId=?", (patientId,))
            results = self.cursor.fetchall()
            if not results:
                raise PatientNumberNotFoundException(patientId)

            appointment = [Appointment(*row) for row in results]
            return appointment
```

```
        except PatientNumberNotFoundException as e:
            print(f"Exception: {e}")
            return []
        except Exception as e:
            print(f"Unexpected exception: {e}")
            return []


    def getAppointmentsForDoctor(self, doctorId) -> List[Appointment]:
        self.cursor.execute("SELECT * FROM appointment WHERE
doctorId=?", (doctorId,))
        results = self.cursor.fetchall()
        appointment = [Appointment(*row) for row in results]
        return appointment


    def scheduleAppointment(self, appointment: Appointment) -> bool:
        try:
            self.cursor.execute("""
                INSERT INTO appointment (appointmentId, patientId,
doctorId, appointmentDate, description)
                VALUES (?, ?, ?, ?, ?)
            """, (appointment.getAppointmentId(),
appointment.getPatientId(),
                  appointment.getDoctorId(),
appointment.getAppointmentDate(),
                  appointment.getDescription()))
            self.conn.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error scheduling appointment: {e}")
            return False


    def updateAppointment(self, appointment: Appointment) -> bool:
        try:
            self.cursor.execute("""
                UPDATE appointment
                SET patientId=?, doctorId=?, appointmentDate=?,
description=?
                WHERE appointmentId=?
            """, (appointment.getPatientId(), appointment.getDoctorId(),
```

```
                appointment.getAppointmentDate(),
appointment.getDescription(),
                appointment.getAppointmentId()))
            self.conn.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error updating appointment: {e}")
            return False


    def cancelAppointment(self, appointmentId) -> bool:
        try:
            self.cursor.execute("DELETE FROM appointment WHERE
appointmentId=?", (appointmentId,))
            self.conn.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error canceling appointment: {e}")
            return False
```

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.

```
import pyodbc

class DBConnection:
    con = None
    @staticmethod
    def getConnection():
        if DBConnection.con is None:
            try:
                DBConnection.con = pyodbc.connect(
                    'Driver={SQL Server};'
                    'Server=LAPTOP-8ULRAAM2\\SQLEXPRESS;'
                    'Database=Host;'
                )
                print("DB Connected !!!")
            except pyodbc.Error as err:
                print(f"Error connecting DB: {err}")
        return DBConnection.con
```

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, patientId):

        super().__init__(f"Patient with ID {patientId} not found.")
```

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

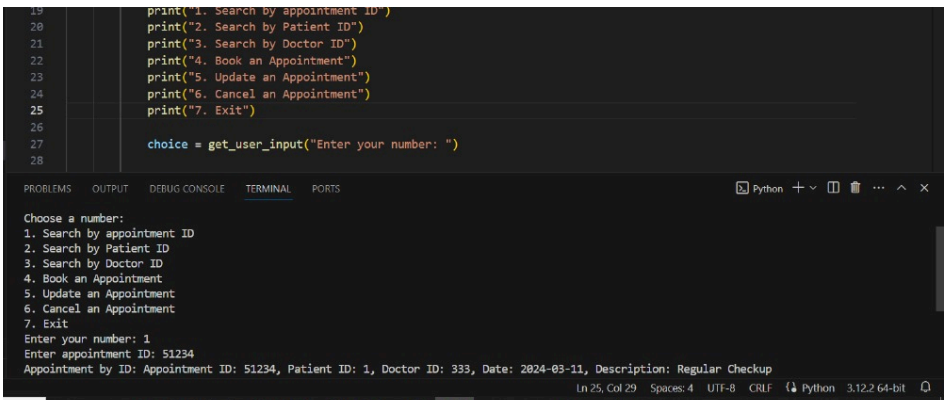
```
from dao.hospitalserviceimpl import HospitalServiceImpl

from entity.appointment import Appointment

from exception.patient_exception import PatientNumberNotFoundException



def get_user_input(prompt):

    while True:

        user_input = input(prompt).strip()

        if user_input:

            return user_input

        print("Input cannot be empty. Please try again.")



def main():

    try:

        hospital_service = HospitalServiceImpl()

        while True:

            print("Choose a number:")

            print("1. Search by appointment ID")

            print("2. Search by Patient ID")

            print("3. Search by Doctor ID")

            print("4. Book an Appointment")
```

```
            print("5. Update an Appointment")
            print("6. Cancel an Appointment")
            print("7. Exit")
            choice = get_user_input("Enter your number: ")

            if choice == '1':
                appointment_id = int(get_user_input("Enter appointment
ID: "))
                appointment =
hospital_service.getAppointmentById(appointment_id)
                print("Appointment by ID:", appointment)

            elif choice == '2':
                patient_id = get_user_input("Enter patient ID: ")
                appointments_for_patient =
hospital_service.getAppointmentsForPatient(patient_id)
                print(f"Appointments for Patient {patient_id}:")
                for appt in appointments_for_patient:
                    print(appt)

            elif choice == '3':
                doctor_id = get_user_input("Enter doctor ID: ")
                appointments_for_doctor =
hospital_service.getAppointmentsForDoctor(doctor_id)
                print(f"Appointments for Doctor {doctor_id}:")
                for appt in appointments_for_doctor:
                    print(appt)

            elif choice == '4':
```

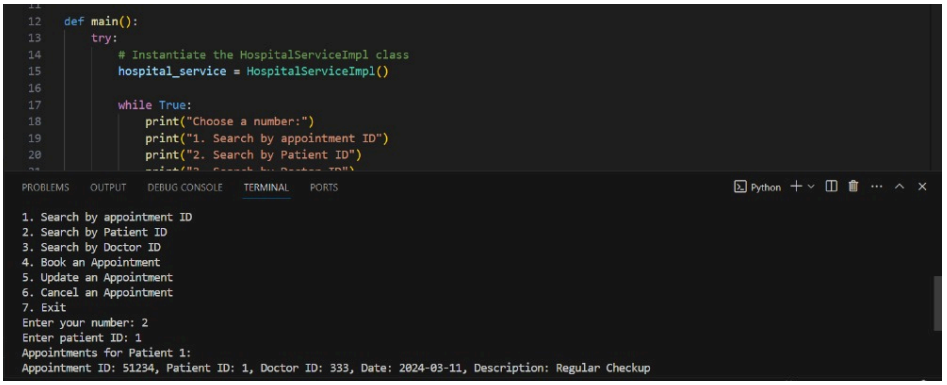
```
                new_appointment =
Appointment(appointmentId=int(get_user_input("Enter appointment ID:
")),

patientId=get_user_input("Enter patient ID: "),

doctorId=get_user_input("Enter doctor ID: "),

appointmentDate=get_user_input("Enter appointment date (YYYY-MM-DD):
"),

description=get_user_input("Enter description: "))
                success =
hospital_service.scheduleAppointment(new_appointment)
                print("Appointment Scheduled:", success)

            elif choice == '5':
                existing_appointment =
Appointment(appointmentId=int(get_user_input("Enter appointment ID:
")),

patientId=get_user_input("Enter patient ID: "),

doctorId=get_user_input("Enter doctor ID: "),

appointmentDate=get_user_input("Enter appointment date (YYYY-MM-DD):
"),

description=get_user_input("Enter description: "))
                success =
hospital_service.updateAppointment(existing_appointment)
                print("Appointment Updated:", success)

            elif choice == '6':
```

```
                appointment_to_cancel = int(get_user_input("Enter
appointment ID to cancel: "))
                success =
hospital_service.cancelAppointment(appointment_to_cancel)
                print("Appointment Canceled:", success)


            elif choice == '7':
                print("Exiting...")
                break


            else:
                print("Invalid choice. Please select a valid option.")


    except PatientNumberNotFoundException as e:
        print(f"Caught PatientNumberNotFoundException: {e}")
    except Exception as e:
        print(f"Unexpected exception: {e}")




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

Show values by appointment ID,



Show details by patient ID,

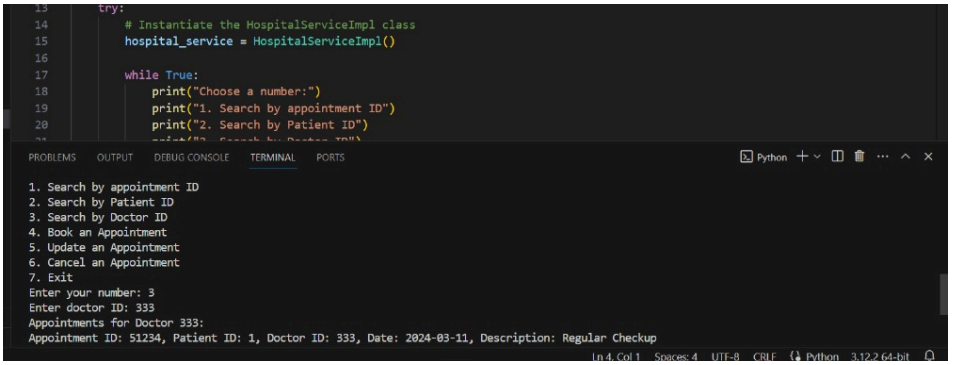


```
def main():
    try:
        # Instantiate the HospitalServiceImpl class
        hospital_service = HospitalServiceImpl()

        while True:
            print("Choose a number:")
            print("1. Search by appointment ID")
            print("2. Search by Patient ID")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1. Search by appointment ID
2. Search by Patient ID
3. Search by Doctor ID
4. Book an Appointment
5. Update an Appointment
6. Cancel an Appointment
7. Exit
Enter your number: 2
Enter patient ID: 1
Appointments for Patient 1:
Appointment ID: 51234, Patient ID: 1, Doctor ID: 333, Date: 2024-03-11, Description: Regular Checkup
```

Show details by DoctorID,

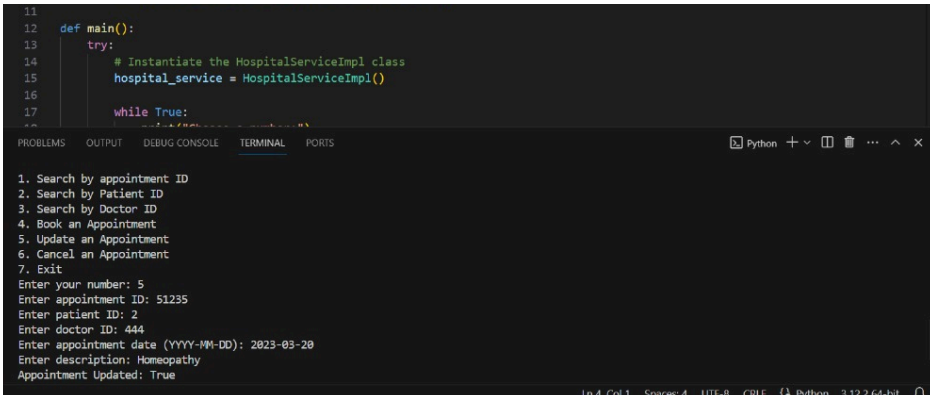


```
    try:
        # Instantiate the HospitalServiceImpl class
        hospital_service = HospitalServiceImpl()

        while True:
            print("Choose a number:")
            print("1. Search by appointment ID")
            print("2. Search by Patient ID")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1. Search by appointment ID
2. Search by Patient ID
3. Search by Doctor ID
4. Book an Appointment
5. Update an Appointment
6. Cancel an Appointment
7. Exit
Enter your number: 3
Enter doctor ID: 333
Appointments for Doctor 333:
Appointment ID: 51234, Patient ID: 1, Doctor ID: 333, Date: 2024-03-11, Description: Regular Checkup
```

Updating appointment,

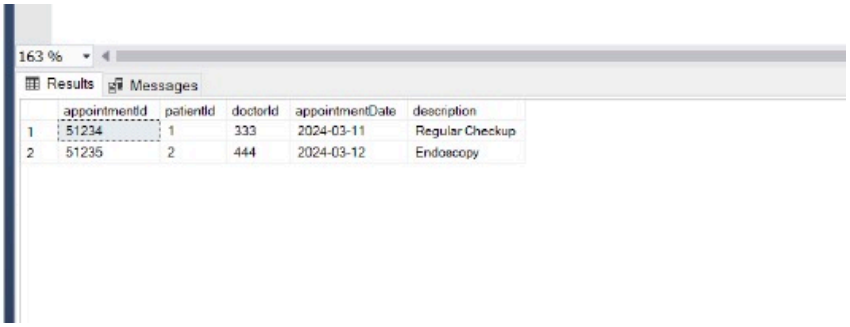


```
def main():
    try:
        # Instantiate the HospitalServiceImpl class
        hospital_service = HospitalServiceImpl()

        while True:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1. Search by appointment ID
2. Search by Patient ID
3. Search by Doctor ID
4. Book an Appointment
5. Update an Appointment
6. Cancel an Appointment
7. Exit
Enter your number: 5
Enter appointment ID: 51235
Enter patient ID: 2
Enter doctor ID: 444
Enter appointment date (YYYY-MM-DD): 2023-03-20
Enter description: Homeopathy
Appointment Updated: True
```

Database:

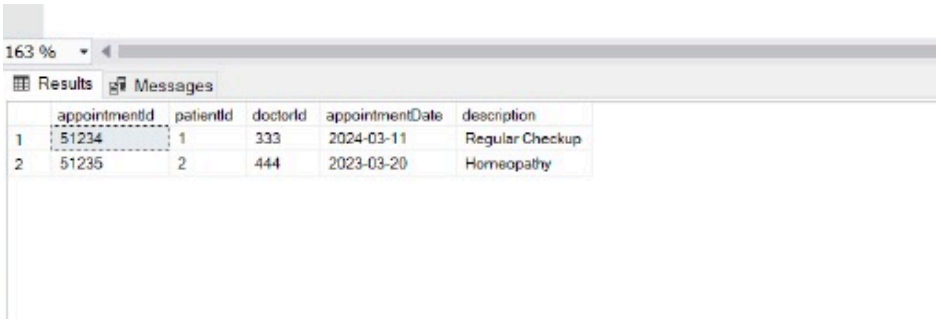


163 %

Results Messages

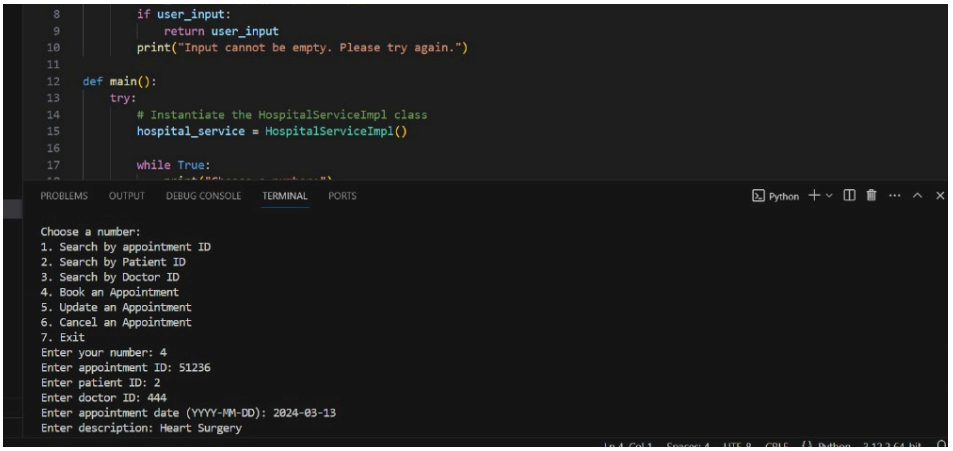
	appointmentId	patientId	doctorId	appointmentDate	description
1	51234	1	333	2024-03-11	Regular Checkup
2	51235	2	444	2024-03-12	Endoscopy

Updated database:



	appointmentId	patientId	doctorId	appointmentDate	description
1	51234	1	333	2024-03-11	Regular Checkup
2	51235	2	444	2023-03-20	Homeopathy

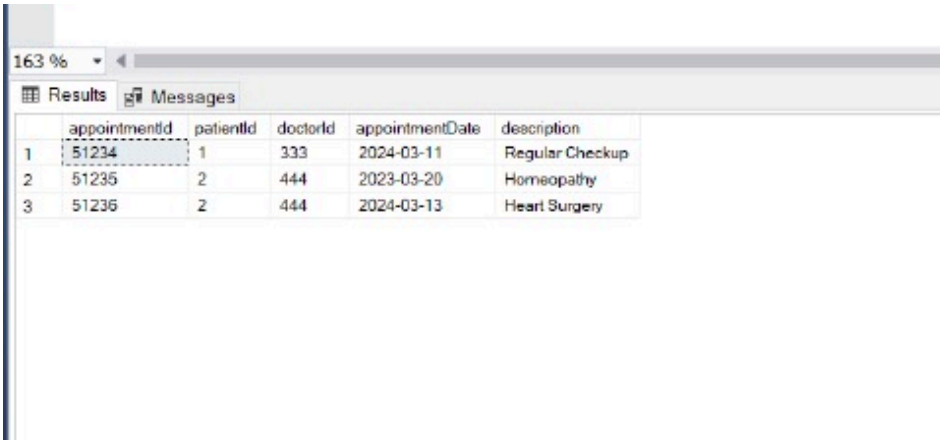
Scheduling appointment:



```
        if user_input:
            return user_input
        print("Input cannot be empty. Please try again.")

def main():
    try:
        # Instantiate the HospitalServiceImpl class
        hospital_service = HospitalServiceImpl()

        while True:
```

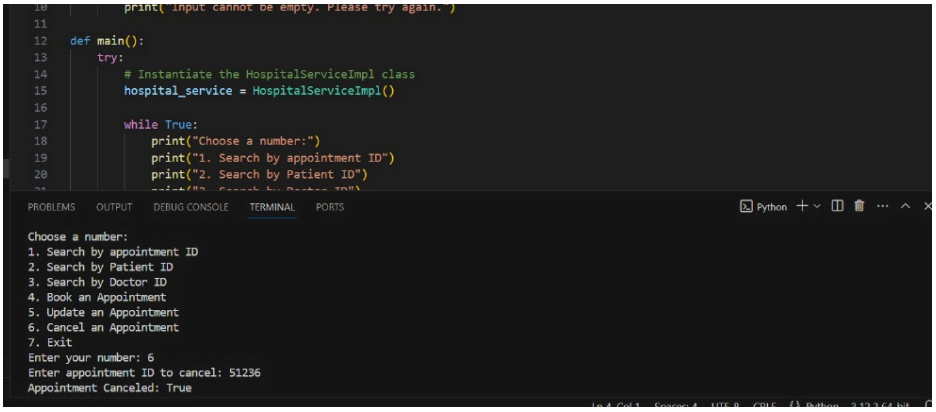
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Choose a number:
1. Search by appointment ID
2. Search by Patient ID
3. Search by Doctor ID
4. Book an Appointment
5. Update an Appointment
6. Cancel an Appointment
7. Exit
Enter your number: 4
Enter appointment ID: 51236
Enter patient ID: 2
Enter doctor ID: 444
Enter appointment date (YYYY-MM-DD): 2024-03-13
Enter description: Heart Surgery
```



	appointmentId	patientId	doctorId	appointmentDate	description
1	51234	1	333	2024-03-11	Regular Checkup
2	51235	2	444	2023-03-20	Homeopathy
3	51236	2	444	2024-03-13	Heart Surgery

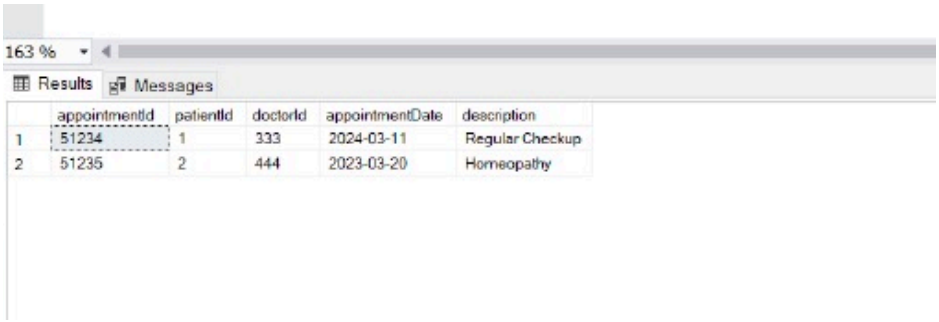
Cancelling appointment:



```
            print("Input cannot be empty. Please try again.")

def main():
    try:
        # Instantiate the HospitalServiceImpl class
        hospital_service = HospitalServiceImpl()

        while True:
            print("Choose a number:")
            print("1. Search by appointment ID")
            print("2. Search by Patient ID")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
Choose a number:
1. Search by appointment ID
2. Search by Patient ID
3. Search by Doctor ID
4. Book an Appointment
5. Update an Appointment
6. Cancel an Appointment
7. Exit
Enter your number: 6
Enter appointment ID to cancel: 51236
Appointment Canceled: True
```



163 %

Results Messages

	appointmentId	patientId	doctorId	appointmentDate	description
1	51234	1	333	2024-03-11	Regular Checkup
2	51235	2	444	2023-03-20	Homeopathy