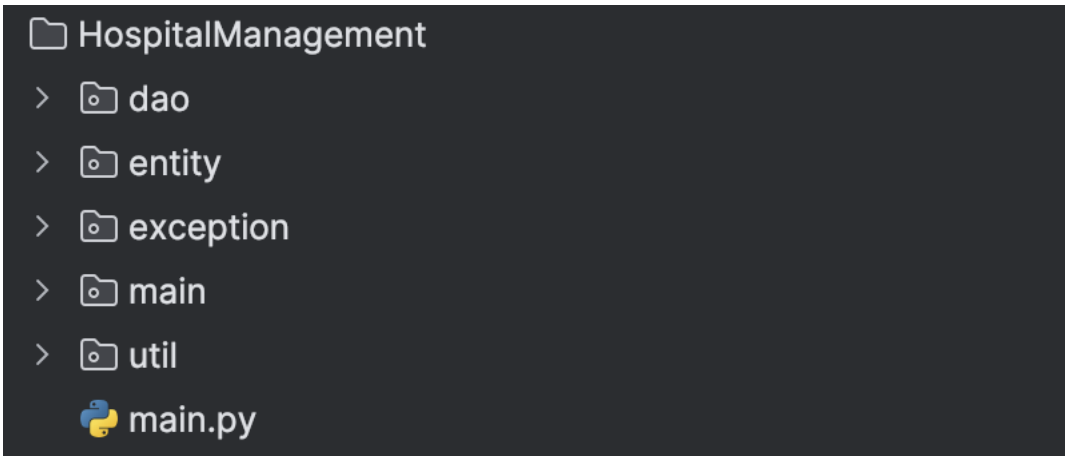
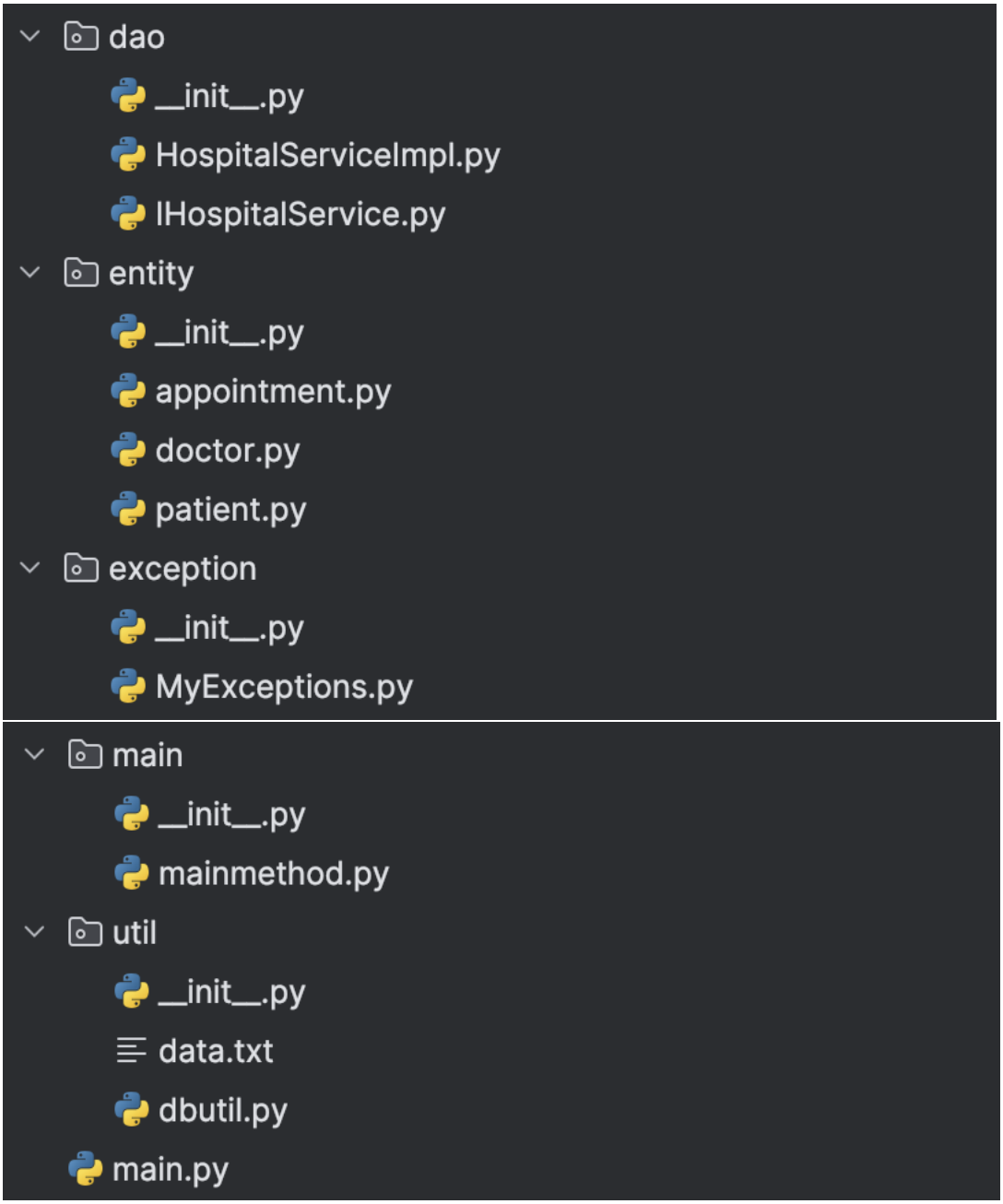
Coding challenge
Hospital Management System
SaiPrabath Chowdary S

Coding Challenge: Hospital Management System

- Project submissions should be done through the partcipants' Github repository and the link should be shared with trainers and Hexavarsity.
- Follow object-oriented principles throughout the project. Use classes and objects to model real-world entities, encapsulate data and behavior, and ensure code reusability.
- Throw user defined exceptions from corresponding methods and handled.
- The following Directory structure is to be followed in the application.
    - **entity**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
        - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object(Use method defined in DBPropertyUtil class to get the connection String).
    - **main**
        - Create a class MainModule and demonstrate the functionalities in a menu driven application.




dao
__init__.py
HospitalServiceImpl.py
IHospitalService.py
entity
__init__.py
appointment.py
doctor.py
patient.py
exception
__init__.py
MyExceptions.py
main
__init__.py
mainmethod.py
util
__init__.py
data.txt
dbutil.py
main.py

Problem Statement:

1Create 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())

```
CREATE TABLE Patient (
patientId INT AUTO_INCREMENT PRIMARY KEY,
firstName VARCHAR(255),
lastName VARCHAR(255),
dateOfBirth DATE,
gender VARCHAR(20) ,
contactNumber VARCHAR(15),
address VARCHAR(255)
);
```

```
CREATE TABLE Doctor (
doctorId INT AUTO_INCREMENT PRIMARY KEY,
firstName VARCHAR(255),
lastName VARCHAR(255),
specialization VARCHAR(255),
contactNumber VARCHAR(15)
);
```

```
CREATE TABLE Appointment (
appointmentId INT AUTO_INCREMENT PRIMARY KEY,
patientId INT,
doctorId INT,
appointmentDate DATE,
description TEXT,
FOREIGN KEY (patientId) REFERENCES Patient(patientId),
FOREIGN KEY (doctorId) REFERENCES Doctor(doctorId)
);
```

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| patientId     | int          | NO   | PRI | NULL    | auto_increment |
| firstName     | varchar(255) | YES  |     | NULL    |                |
| lastName      | varchar(255) | YES  |     | NULL    |                |
| dateOfBirth   | date         | YES  |     | NULL    |                |
| gender        | varchar(20)  | YES  |     | NULL    |                |
| contactNumber | varchar(15)  | YES  |     | NULL    |                |
| address       | varchar(255) | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
7 rows in set (0.01 sec)

mysql> desc doctor;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| doctorId       | int          | NO   | PRI | NULL    | auto_increment |
| firstName      | varchar(255) | YES  |     | NULL    |                |
| lastName       | varchar(255) | YES  |     | NULL    |                |
| specialization | varchar(255) | YES  |     | NULL    |                |
| contactNumber  | varchar(15)  | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> desc appointment;
+-----------------+------+------+-----+---------+----------------+
| Field           | Type | Null | Key | Default | Extra          |
+-----------------+------+------+-----+---------+----------------+
| appointmentId   | int  | NO   | PRI | NULL    | auto_increment |
| patientId       | int  | YES  | MUL | NULL    |                |
| doctorId        | int  | YES  | MUL | NULL    |                |
| appointmentDate | date | YES  |     | NULL    |                |
| description     | text | YES  |     | NULL    |                |
+-----------------+------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

1. Define `**Patient**` class with the following confidential attributes:

a. patientId
b. firstName
c. lastName
d. dateOfBirth
e. gender
f. contactNumber
g. address

```python
class Patient:
    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 setPatientId(self, patientId):
        self.patientId = patientId
    def getFirstName(self):
        return self.firstName
```

```
    def setFirstName(self, firstName):
        self.firstName = firstName
    def getLastName(self):
        return self.lastName
    def setLastName(self, lastName):
        self.lastName = lastName
    def getDateOfBirth(self):
        return self.dateOfBirth
    def setDateOfBirth(self, dateOfBirth):
        self.dateOfBirth = dateOfBirth
    def getGender(self):
        return self.gender
    def setGender(self, gender):
        self.gender = gender
    def getContactNumber(self):
        return self.contactNumber
    def setContactNumber(self, contactNumber):
        self.contactNumber = contactNumber
    def getAddress(self):
        return self.address
    def setAddress(self, address            ):
        self.address = address
    def __str__(self):
        return f"Patient ID: {self.patientId}, Name: {self.firstName} {self.lastName}, Date of Birth:
{self.dateOfBirth}, Gender: {self.gender}, Contact Number: {self.contactNumber}, Address:
{self.address}"
```

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

```
class Doctor:
    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 setDoctorId(self, doctorId):
        self.doctorId = doctorId
    def getFirstName(self):
        return self.firstName
    def setFirstName(self, firstName):
        self.firstName = firstName
    def getLastName(self):
        return self.lastName
    def setLastName(self, lastName):
        self.lastName = lastName
```

```
    def getSpecialization(self):
        return self.specialization
    def setSpecialization(self, specialization):
        self.specialization = specialization
    def getContactNumber(self):
        return self.contactNumber
    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 Number: {self.contactNumber}"
```

3. Appointment Class:

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

```
class Appointment:
    def __init__(self, appointmentId, patientId, doctorId, appointmentDate, description):
        self.appointmentId = appointmentId
        self.patientId = patientId
        self.doctorId = doctorId
        self.appointmentDate = appointmentDate
        self.description = description
    def getAppointmentId(self):
        return self.appointmentId
    def setAppointmentId(self, appointmentId):
        self.appointmentId = appointmentId
    def getPatientId(self):
        return self.patientId
    def setPatientId(self, patientId):
        self.patientId = patientId
    def getDoctorId(self):
        return self.doctorId
    def setDoctorId(self, doctorId):
        self.doctorId = doctorId
    def getAppointmentDate(self):
        return self.appointmentDate
    def setAppointmentDate(self, appointmentDate):
        self.appointmentDate = appointmentDate
    def getDescription(self):
        return self.description
    def setDescription(self, description):
        self.description = description
    def __str__(self):
        return f"Appointment ID: {self.appointmentId}, Patient ID: {self.patientId}, Doctor ID: {self.doctorId}
\nAppointment Date: {self.appointmentDate}, 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

```python
from abc import ABC, abstractmethod
class IHospitalService(ABC):
    @abstractmethod
    def getAppointmentById(self, appointmentId):
        pass
    @abstractmethod
    def getAppointmentsForPatient(self, patientId):
        pass
    @abstractmethod
    def getAppointmentsForDoctor(self, doctorId):
        pass
    @abstractmethod
    def scheduleAppointment(self, appointment_obj):
        pass
    @abstractmethod
    def updateAppointment(self, appointment_obj):
        pass
    @abstractmethod
    def cancelAppointment(self, appointmentId) :
        pass
```

```
    @abstractmethod
    def addPatient(self, patient):
        pass
    @abstractmethod
    def addDoctor(self, doctor):
        pass
```

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

```
from mysql.connector import Error
from HospitalManagement.entity.appointment import Appointment
from HospitalManagement.dao.IHospitalService import IHospitalService
from HospitalManagement.util.dbutil import DBConnection
from HospitalManagement.exception.MyExceptions import PatientNumberNotFoundException,
AppointmentNotFoundException
```

```
class HospitalServiceImpl(IHospitalService):
    def __init__(self):
        self.connection = DBConnection.getConnection()
```

```
    def getAppointmentById(self, appointmentId):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Appointment WHERE appointmentId = %s"
            cursor.execute(query, (appointmentId,))
            appointment_data = cursor.fetchone()
            cursor.close()
            if appointment_data:
                appointment = Appointment(*appointment_data)
                return appointment
            else:
                raise AppointmentNotFoundException(f"Appointment with ID {appointmentId} not found.")
        except Error as e:
            print("Error Fetching Appointment: ", e)
            return None
```

```
    def getAppointmentsForPatient(self, patientId):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Appointment WHERE patientId = %s"
            appointment_data = cursor.fetchall()
            cursor.close()
            appointments = [Appointment(*data) for data in appointment_data]
            return appointments
        except PatientNumberNotFoundException as e:
```

```
            print("Error Fetching Appointments: ", e)
            return []
        except Error as e:
            print("Error Fetching Appointments: ", e)
            return []
```

```
    def getAppointmentsForDoctor(self, doctorId):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Appointment WHERE doctorId = %s"
            cursor.execute(query, (doctorId,))
            appointment_data = cursor.fetchall()
            cursor.close()
            appointments = [Appointment(*data) for data in appointment_data]
            return appointments
        except Error as e:
            print("Error Fetching Appointments: ", e)
            return []
```

```
    def scheduleAppointment(self, appointment):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Appointment (patientId, doctorId, appointmentDate, description) VALUES
(%s, %s, %s, %s)"
            values = (appointment.patientId, appointment.doctorId, appointment.appointmentDate,
appointment.description)
            cursor.execute(query, values)
            self.connection.commit()
            cursor.close()
            return True
        except Error as e:
            print("Error scheduling Appointment:", e)
            return False
```

```
    def updateAppointment(self, appointment):
        try:
            cursor = self.connection.cursor()
            query = "UPDATE Appointment SET patientId = %s, doctorId = %s, appointmentDate = %s,
description = %s WHERE appointmentId = %s"
            values = (appointment.patientId, appointment.doctorId, appointment.appointmentDate,
appointment.description, appointment.appointmentId)
            cursor.execute(query, values)
            self.connection.commit()
            cursor.close()
            return True
```

```
        except Error as e:
            print("Error updating Appointment:", e)
            return False
```

```
    def cancelAppointment(self, appointmentId):
        try:
            cursor = self.connection.cursor()
            query = "DELETE FROM Appointment WHERE appointmentId = %s"
            cursor.execute(query, (appointmentId,))
            self.connection.commit()
            cursor.close()
            return True
        except Error as e:
            print("Error:", e)
            return False
```

```
    def addPatient(self, patient):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Patient (firstName, lastName, dateOfBirth, gender, contactNumber, address)
VALUES (%s, %s, %s, %s, %s, %s)"
            values = (patient.getFirstName(), patient.getLastName(), patient.getDateOfBirth(),
patient.getGender(),patient.getContactNumber(), patient.getAddress())
            cursor.execute(query, values)
            self.connection.commit()
            cursor.close()
            return True
        except Error as e:
            print("Error Adding Patient:", e)
            return False
```

```
    def addDoctor(self, doctor):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Doctor (firstName, lastName, specialization, contactNumber) VALUES
(%s, %s, %s, %s)"
            values = (doctor.getFirstName(), doctor.getLastName(), doctor.getSpecialization(),
doctor.getContactNumber())
            cursor.execute(query, values)
            self.connection.commit()
            cursor.close()
            return True
        except Error as e:
            print("Error Adding Doctor:", 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.

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.

```
import mysql.connector
from mysql.connector import Error
```

```
class DBConnection:
    connection = None
    @staticmethod
    def getConnection():
        try:
            if DBConnection.connection is None or not DBConnection.connection.is_connected():
                connection_string = PropertyUtil.getPropertyString("data.txt")
                DBConnection.connection = mysql.connector.connect(**connection_string)
            return DBConnection.connection
        except Error as e:
            print("Error:", e)
            return None
```

```
class PropertyUtil:
    @staticmethod
    def getPropertyString(property_file):
        properties = {}
        with open(property_file) as file:
            for line in file:
                key, value = line.strip().split('=')
                properties[key] = value
        return properties
```

data.txt

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

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, message="Patient number not found in the database"):
        self.message = message
        super().__init__(self.message)
```

```
class AppointmentNotFoundException(Exception):
    def __init__(self, message="Appointment not found in the database"):
        self.message = message
        super().__init__(self.message)
```

9. Create class named MainModule with main method in package mainmod.

Trigger all the methods in service implementation class.

```
from HospitalManagement.dao.HospitalServiceImpl import HospitalServiceImpl
from HospitalManagement.entity.appointment import Appointment
from HospitalManagement.entity.doctor import Doctor
from HospitalManagement.entity.patient import Patient
```

```
class MainModule:
    def __init__(self):
        self.hospital_service = HospitalServiceImpl()
```

```
    def display_menu(self):
        print("\nWelcome to the Hospital Management System!")
        print("1. Get Appointment by ID")
        print("2. Get Appointments for Patient")
        print("3. Get Appointments for Doctor")
        print("4. Schedule Appointment")
        print("5. Update Appointment")
        print("6. Cancel Appointment")
        print("6. Add Doctor")
        print("6. Add Patient")
        print("9. Exit")
```

```
    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")
            if choice == "1":
                self.get_appointment_by_id()
            elif choice == "2":
                self.get_appointments_for_patient()
            elif choice == "3":
```

```
            self.get_appointments_for_doctor()
        elif choice == "4":
            self.schedule_appointment()
        elif choice == "5":
            self.update_appointment()
        elif choice == "6":
            self.cancel_appointment()
        elif choice == "7":
            self.add_doctor()
        elif choice == "8":
            self.add_patient()
        elif choice == "9":
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")


    def get_appointment_by_id(self):
        appointment_id = input("Enter the Appointment ID: ")
        appointment = self.hospital_service.getAppointmentById(appointment_id)
        if appointment:
            print(appointment)
        else:
            print("Appointment not found.")

    def get_appointments_for_patient(self):
        patient_id = input("Enter the Patient ID: ")
        appointments = self.hospital_service.getAppointmentsForPatient(patient_id)
        if appointments:
            for appointment in appointments:
                print(appointment)
        else:
            print("No appointments found for the patient.")

    def get_appointments_for_doctor(self):
        doctor_id = input("Enter the Doctor ID: ")
        appointments = self.hospital_service.getAppointmentsForDoctor(doctor_id)
        if appointments:
            for appointment in appointments:
                print(appointment)
        else:
            print("No appointments found for the doctor.")



    def schedule_appointment(self):
        try:
            patient_id = input("Enter Patient ID: ")
            doctor_id = input("Enter Doctor ID: ")
            appointment_date = input("Enter Appointment Date (YYYY-MM-DD): ")
```

```
            description = input("Enter Description: ")
            appointment = Appointment(None, patient_id, doctor_id, appointment_date, description)
            if self.hospital_service.scheduleAppointment(appointment):
                print("Appointment scheduled successfully!")
            else:
                print("Failed to schedule appointment.")
        except Exception as e:
            print(f"Error: {e}")

    def update_appointment(self):
        try:
            appointment_id = input("Enter Appointment ID to update: ")
            appointment = self.hospital_service.getAppointmentById(appointment_id)
            if appointment:
                print(appointment)
                patient_id = input("Enter Patient ID to update: ")
                doctor_id = input("Enter Doctor ID to update: ")
                appointment_date = input("Enter new Appointment Date (YYYY-MM-DD): ")
                description = input("Enter new Description: ")
                appointment.setPatientId(patient_id)
                appointment.setDoctorId(doctor_id)
                appointment.setAppointmentDate(appointment_date)
                appointment.setDescription(description)
                if self.hospital_service.updateAppointment(appointment):
                    print("Appointment updated successfully!")
                else:
                    print("Failed to update appointment.")
        except Exception as e:
            print(f"Error : {e}")


    def cancel_appointment(self):
        try:
            appointment_id = input("Enter Appointment ID to cancel: ")
            if self.hospital_service.cancelAppointment(appointment_id):
                print("Appointment canceled successfully!")
            else:
                print("Failed to cancel appointment.")
        except Exception as e:
            print(f"Error canceling appointment: {e}")

    def add_patient(self):
        try:
            first_name = input("Enter Patient First Name: ")
            last_name = input("Enter Patient Last Name: ")
            date_of_birth = input("Enter Patient Date of Birth (YYYY-MM-DD): ")
            gender = input("Enter Patient Gender: ")
            contact_number = input("Enter Patient Contact Number: ")
            address = input("Enter Patient Address: ")
            patient = Patient(None, first_name, last_name, date_of_birth, gender, contact_number, address)
            if self.hospital_service.addPatient(patient):
                print("Patient added successfully!")
            else:
                print("Failed to add patient.")
        except Exception as e:
```

```
            print(f"Error adding patient: {e}")
```

```
    def add_doctor(self):
        try:
            first_name = input("Enter Doctor First Name: ")
            last_name = input("Enter Doctor Last Name: ")
            specialization = input("Enter Doctor Specialization: ")
            contact_number = input("Enter Doctor Contact Number: ")
            doctor = Doctor(None, first_name, last_name, specialization, contact_number)
            if self.hospital_service.addDoctor(doctor):
                print("Doctor added successfully!")
            else:
                print("Failed to add doctor.")
        except Exception as e:
            print(f"Error adding doctor: {e}")
```

Main.py

```
from HospitalManagement.main.mainmethod import MainModule

if __name__ == "__main__":
    main_module = MainModule()
    main_module.run()
```