Hospital Management System

Problem Statement:

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

```
-- Create Patient Table
CREATE TABLE Patient (
    patientId VARCHAR(20) PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    dateOfBirth DATE,
    gender VARCHAR(10),
    contactNumber VARCHAR(15),
    address TEXT
);
```

```
mysql> CREATE DATABASE HospitalManagementSystem;
Query OK, 1 row affected (0.060 sec)

mysql> USE HospitalManagementSystem;
Database changed
mysql> CREATE TABLE Patient ( patientId VARCHAR(20) PRIMARY KEY, firstName VARCHAR(50), lastName VARCHAR(50), dateOfBirt
h DATE, gender VARCHAR(10), contactNumber VARCHAR(15), address TEXT);
Query OK, 0 rows affected (0.171 sec)

mysql>
```

```
-- Create Doctor Table
CREATE TABLE Doctor (
    doctorId VARCHAR(20) PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    specialization VARCHAR(50),
    contactNumber VARCHAR(15)
);
```

```
mysql> CREATE TABLE Doctor ( doctorId VARCHAR(20) PRIMARY KEY, firstName VARCHAR(50), lastName VARCHAR(50), specializati
on VARCHAR(50), contactNumber VARCHAR(15));
Query OK, 0 rows affected (0.154 sec)

mysql>
```

```
-- Create Appointment Table
CREATE TABLE Appointment (
    appointmentId VARCHAR(20) PRIMARY KEY,
    patientId VARCHAR(20),
    doctorId VARCHAR(20),
    appointmentDate DATE,
    description TEXT,
    FOREIGN KEY (patientId) REFERENCES Patient(patientId),
```

FOREIGN KEY (doctorId) REFERENCES Doctor(doctorId)
);

```
mysql> CREATE TABLE Appointment ( appointmentId VARCHAR(20) PRIMARY KEY, patientId VARCHAR(20), doctorId VARCHAR(20), ap
pointmentDate DATE, description TEXT, FOREIGN KEY (patientId) REFERENCES Patient(patientId), FOREIGN KEY (doctorId) REFE
RENCES Doctor(doctorId));
Query OK, 0 rows affected (0.345 sec)

mysql>
```

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, patientId=None, firstName=None, lastName=None,
dateOfBirth=None, gender=None, contactNumber=None, address=None):
        self.__patientId = patientId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__dateOfBirth = dateOfBirth
        self.__gender = gender
        self.__contactNumber = contactNumber
        self.__address = address

    # Getters and setters
    def get_patientId(self): return self.__patientId
    def set_patientId(self, patientId): self.__patientId = patientId

    def get_firstName(self): return self.__firstName
    def set_firstName(self, firstName): self.__firstName = firstName

    def get_lastName(self): return self.__lastName
    def set_lastName(self, lastName): self.__lastName = lastName

    def get_dateOfBirth(self): return self.__dateOfBirth
    def set_dateOfBirth(self, dateOfBirth): self.__dateOfBirth = dateOfBirth
```

```
    def get_gender(self): return self.__gender
    def set_gender(self, gender): self.__gender = gender

    def get_contactNumber(self): return self.__contactNumber
    def set_contactNumber(self, contactNumber): self.__contactNumber =
contactNumber

    def get_address(self): return self.__address
    def set_address(self, address): self.__address = address

    def __str__(self):
        return f"Patient[{self.__patientId}, {self.__firstName}, {self.__lastName},
{self.__dateOfBirth}, {self.__gender}, {self.__contactNumber},
{self.__address}]"
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Patient.py"
PS C:\Users\HP>
```

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=None, firstName=None, lastName=None,
specialization=None, contactNumber=None):
        self.__doctorId = doctorId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__specialization = specialization
        self.__contactNumber = contactNumber

    # Getters and setters
    def get_doctorId(self): return self.__doctorId
    def set_doctorId(self, doctorId): self.__doctorId = doctorId

    def get_firstName(self): return self.__firstName
    def set_firstName(self, firstName): self.__firstName = firstName

    def get_lastName(self): return self.__lastName
    def set_lastName(self, lastName): self.__lastName = lastName
```

```
    def get_specialization(self): return self.__specialization
    def set_specialization(self, specialization): self.__specialization =
specialization

    def get_contactNumber(self): return self.__contactNumber
    def set_contactNumber(self, contactNumber): self.__contactNumber =
contactNumber

    def __str__(self):
        return f"Doctor[{self.__doctorId}, {self.__firstName}, {self.__lastName},
{self.__specialization}, {self.__contactNumber}]"
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Doctor.py"
PS C:\Users\HP>
```

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

```
class Appointment:
    def __init__(self, appointmentId=None, patientId=None, doctorId=None,
appointmentDate=None, description=None):
        self.__appointmentId = appointmentId
        self.__patientId = patientId
        self.__doctorId = doctorId
        self.__appointmentDate = appointmentDate
        self.__description = description

    # Getters and setters
    def get_appointmentId(self): return self.__appointmentId
    def set_appointmentId(self, appointmentId): self.__appointmentId =
appointmentId

    def get_patientId(self): return self.__patientId
    def set_patientId(self, patientId): self.__patientId = patientId

    def get_doctorId(self): return self.__doctorId
    def set_doctorId(self, doctorId): self.__doctorId = doctorId

    def get_appointmentDate(self): return self.__appointmentDate
```

```
    def set_appointmentDate(self, appointmentDate): self.__appointmentDate =
appointmentDate

    def get_description(self): return self.__description
    def set_description(self, description): self.__description = description

    def __str__(self):
        return f"Appointment[{self.__appointmentId}, {self.__patientId},
{self.__doctorId}, {self.__appointmentDate}, {self.__description}]"
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Appointment.py"
PS C:\Users\HP>
```

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 Patient:
    def __init__(self, patientId=None, firstName=None, lastName=None,
dateOfBirth=None, gender=None, contactNumber=None, address=None):
        self.__patientId = patientId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__dateOfBirth = dateOfBirth
        self.__gender = gender
        self.__contactNumber = contactNumber
        self.__address = address

    # Getters and setters
    def get_patientId(self): return self.__patientId
    def set_patientId(self, value): self.__patientId = value

    def get_firstName(self): return self.__firstName
    def set_firstName(self, value): self.__firstName = value

    def get_lastName(self): return self.__lastName
    def set_lastName(self, value): self.__lastName = value

    def get_dateOfBirth(self): return self.__dateOfBirth
    def set_dateOfBirth(self, value): self.__dateOfBirth = value

    def get_gender(self): return self.__gender
    def set_gender(self, value): self.__gender = value
```

```
    def get_contactNumber(self): return self.__contactNumber
    def set_contactNumber(self, value): self.__contactNumber = value

    def get_address(self): return self.__address
    def set_address(self, value): self.__address = value

    # Print method
    def print_details(self):
        print(f"Patient ID: {self.__patientId}")
        print(f"Name: {self.__firstName} {self.__lastName}")
        print(f"Date of Birth: {self.__dateOfBirth}")
        print(f"Gender: {self.__gender}")
        print(f"Contact Number: {self.__contactNumber}")
        print(f"Address: {self.__address}")
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Patient.py"
PS C:\Users\HP>
```

```
class Doctor:
    def __init__(self, doctorId=None, firstName=None, lastName=None,
specialization=None, contactNumber=None):
        self.__doctorId = doctorId
        self.__firstName = firstName
        self.__lastName = lastName
        self.__specialization = specialization
        self.__contactNumber = contactNumber

    # Getters and setters
    def get_doctorId(self): return self.__doctorId
    def set_doctorId(self, value): self.__doctorId = value

    def get_firstName(self): return self.__firstName
    def set_firstName(self, value): self.__firstName = value

    def get_lastName(self): return self.__lastName
    def set_lastName(self, value): self.__lastName = value

    def get_specialization(self): return self.__specialization
    def set_specialization(self, value): self.__specialization = value

    def get_contactNumber(self): return self.__contactNumber
    def set_contactNumber(self, value): self.__contactNumber = value

    # Print method
```

```
    def print_details(self):
        print(f"Doctor ID: {self.__doctorId}")
        print(f"Name: {self.__firstName} {self.__lastName}")
        print(f"Specialization: {self.__specialization}")
        print(f"Contact Number: {self.__contactNumber}")
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Doctor.py"
PS C:\Users\HP>
```

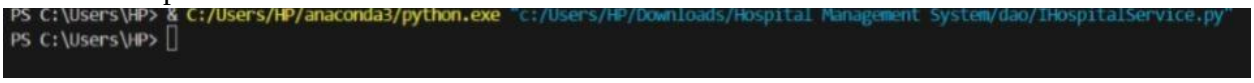
```
class Appointment:
    def __init__(self, appointmentId=None, patientId=None, doctorId=None,
appointmentDate=None, description=None):
        self.__appointmentId = appointmentId
        self.__patientId = patientId
        self.__doctorId = doctorId
        self.__appointmentDate = appointmentDate
        self.__description = description

    # Getters and setters
    def get_appointmentId(self): return self.__appointmentId
    def set_appointmentId(self, value): self.__appointmentId = value

    def get_patientId(self): return self.__patientId
    def set_patientId(self, value): self.__patientId = value

    def get_doctorId(self): return self.__doctorId
    def set_doctorId(self, value): self.__doctorId = value

    def get_appointmentDate(self): return self.__appointmentDate
    def set_appointmentDate(self, value): self.__appointmentDate = value

    def get_description(self): return self.__description
    def set_description(self, value): self.__description = value

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

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/entity/Appointment.py"
PS C:\Users\HP>
```

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
from entity.Appointment import Appointment

class IHospitalService(ABC):

    @abstractmethod
    def getAppointmentById(self, appointmentId: str) -> Appointment:
        pass

    @abstractmethod
    def getAppointmentsForPatient(self, patientId: str) -> list:
        pass

    @abstractmethod
    def getAppointmentsForDoctor(self, doctorId: str) -> list:
        pass

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

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

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




```
import mysql.connector
from entity.Appointment import Appointment
from dao.IHospitalService import IHospitalService
from util.DBConnUtil import get_connection

class HospitalServiceImpl(IHospitalService):

    def __init__(self):
        self.conn = get_connection("db_config.properties")
        self.cursor = self.conn.cursor(dictionary=True)

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

    def getAppointmentsForPatient(self, patientId: str) -> list:
        query = "SELECT * FROM Appointment WHERE patientId = %s"
        self.cursor.execute(query, (patientId,))
        results = self.cursor.fetchall()
        return [Appointment(**row) for row in results]

    def getAppointmentsForDoctor(self, doctorId: str) -> list:
        query = "SELECT * FROM Appointment WHERE doctorId = %s"
        self.cursor.execute(query, (doctorId,))
        results = self.cursor.fetchall()
        return [Appointment(**row) for row in results]

    def scheduleAppointment(self, appointment: Appointment) -> bool:
        try:
```

```
            query = """
                INSERT INTO Appointment (appointmentId, patientId, doctorId,
appointmentDate, description)
                VALUES (%s, %s, %s, %s, %s)
            """
            self.cursor.execute(query, (
                appointment.get_appointmentId(),
                appointment.get_patientId(),
                appointment.get_doctorId(),
                appointment.get_appointmentDate(),
                appointment.get_description()
            ))
            self.conn.commit()
            return True
        except Exception as e:
            print(f"Error scheduling appointment: {e}")
            return False

    def updateAppointment(self, appointment: Appointment) -> bool:
        try:
            query = """
                UPDATE Appointment
                SET patientId = %s, doctorId = %s, appointmentDate = %s, description
= %s
                WHERE appointmentId = %s
            """
            self.cursor.execute(query, (
                appointment.get_patientId(),
                appointment.get_doctorId(),
                appointment.get_appointmentDate(),
                appointment.get_description(),
                appointment.get_appointmentId()
            ))
            self.conn.commit()
            return self.cursor.rowcount > 0
        except Exception as e:
            print(f"Error updating appointment: {e}")
            return False

    def cancelAppointment(self, appointmentId: str) -> bool:
        try:
            query = "DELETE FROM Appointment WHERE appointmentId = %s"
            self.cursor.execute(query, (appointmentId,))
```

```
            self.conn.commit()
            return self.cursor.rowcount > 0
        except Exception as e:
            print(f"Error cancelling appointment: {e}")
            return False
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/dao/HospitalServiceImpl.py"
PS C:\Users\HP>
```

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

```
import mysql.connector
from dao.IHospitalService import IHospitalService
from entity.Appointment import Appointment
from util.DBConnection import DBConnection as get_connection

class HospitalServiceImpl(IHospitalService):

    def __init__(self):
        try:
            self.conn = get_connection("db_config.properties")
            self.cursor = self.conn.cursor(dictionary=True)
        except mysql.connector.Error as e:
            print("Database connection error:", e)

    def getAppointmentById(self, appointmentId: str) -> Appointment:
        try:
            query = "SELECT * FROM Appointment WHERE appointmentId = %s"
            self.cursor.execute(query, (appointmentId,))
            result = self.cursor.fetchone()
            if result:
                return Appointment(
                    appointmentId=result["appointmentId"],
                    patientId=result["patientId"],
                    doctorId=result["doctorId"],
                    appointmentDate=result["appointmentDate"],
                    description=result["description"]
                )
            else:
                return None
        except Exception as e:
            print("Error in getAppointmentById:", e)
            return None
```

```
    def getAppointmentsForPatient(self, patientId: str) -> list:
        appointments = []
        try:
            query = "SELECT * FROM Appointment WHERE patientId = %s"
            self.cursor.execute(query, (patientId,))
            rows = self.cursor.fetchall()
            for row in rows:
                appointment = Appointment(
                    appointmentId=row["appointmentId"],
                    patientId=row["patientId"],
                    doctorId=row["doctorId"],
                    appointmentDate=row["appointmentDate"],
                    description=row["description"]
                )
                appointments.append(appointment)
        except Exception as e:
            print("Error in getAppointmentsForPatient:", e)
        return appointments

    def getAppointmentsForDoctor(self, doctorId: str) -> list:
        appointments = []
        try:
            query = "SELECT * FROM Appointment WHERE doctorId = %s"
            self.cursor.execute(query, (doctorId,))
            rows = self.cursor.fetchall()
            for row in rows:
                appointment = Appointment(
                    appointmentId=row["appointmentId"],
                    patientId=row["patientId"],
                    doctorId=row["doctorId"],
                    appointmentDate=row["appointmentDate"],
                    description=row["description"]
                )
                appointments.append(appointment)
        except Exception as e:
            print("Error in getAppointmentsForDoctor:", e)
        return appointments

    def scheduleAppointment(self, appointment: Appointment) -> bool:
        try:
            query = """
                INSERT INTO Appointment (appointmentId, patientId, doctorId,
appointmentDate, description)
```

```
            VALUES (%s, %s, %s, %s, %s)
        """
        self.cursor.execute(query, (
            appointment.get_appointmentId(),
            appointment.get_patientId(),
            appointment.get_doctorId(),
            appointment.get_appointmentDate(),
            appointment.get_description()
        ))
        self.conn.commit()
        return True
    except Exception as e:
        print("Error in scheduleAppointment:", e)
        return False

def updateAppointment(self, appointment: Appointment) -> bool:
    try:
        query = """
            UPDATE Appointment
            SET patientId = %s, doctorId = %s, appointmentDate = %s, description
= %s
            WHERE appointmentId = %s
        """
        self.cursor.execute(query, (
            appointment.get_patientId(),
            appointment.get_doctorId(),
            appointment.get_appointmentDate(),
            appointment.get_description(),
            appointment.get_appointmentId()
        ))
        self.conn.commit()
        return self.cursor.rowcount > 0
    except Exception as e:
        print("Error in updateAppointment:", e)
        return False

def cancelAppointment(self, appointmentId: str) -> bool:
    try:
        query = "DELETE FROM Appointment WHERE appointmentId = %s"
        self.cursor.execute(query, (appointmentId,))
        self.conn.commit()
        return self.cursor.rowcount > 0
    except Exception as e:
```

```
        print("Error in cancelAppointment:", e)
        return False
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/dao/HospitalServiceImpl.py"
PS C:\Users\HP>
```

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.

```
def getPropertyString(file_name="db_config.properties"):
    props = {}
    try:
        with open(file_name, 'r') as f:
            for line in f:
                line = line.strip()
                if not line or line.startswith('#'):
                    continue
                key, value = line.split('=')
                props[key.strip()] = value.strip()

        # Return formatted connection dictionary
        return {
            host=="localhost",
            user=="root",
            password== "password",
            port==3306',
            database=="HospitalManagementSystem"
        }

    except FileNotFoundError:
        print("Property file not found!")
        return None
    except Exception as e:
        print(f"Error reading properties: {e}")
        return None
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/util/PropertyUtil.py"
PS C:\Users\HP>
```

```
import mysql.connector
```

```
from util.PropertyUtil import getPropertyString

class DBConnection:
    __connection = None

    @staticmethod
    def getConnection():
        if DBConnection.__connection is None:
            try:
                props = getPropertyString()
                if props:
                    DBConnection.__connection = mysql.connector.connect(
                        host=="localhost",
                        user=="root",
                        password== "password",
                        port== 3306',
                        database=="HospitalManagementSystem"
                    )
                    print("Database connection established.")
                else:
                    print("Failed to load DB properties.")
            except Exception as e:
                print(f"Error connecting to database: {e}")
        return DBConnection.__connection
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/util/DBConnection.py"
PS C:\Users\HP>
```

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):
        super().__init__(f"Patient with number '{patient_number}' not found in the database.")
```

```
PS C:\Users\HP> & C:/Users/HP/anaconda3/python.exe "c:/Users/HP/Downloads/Hospital Management System/myexceptions/PatientNumberNotFoundException.py"
PS C:\Users\HP>
```

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

```
from service.HospitalServiceImpl import HospitalServiceImpl
```

```
from repository.PatientRepositoryImpl import PatientRepositoryImpl
from myexceptions.PatientNumberNotFoundException import
PatientNumberNotFoundException

def display_menu():
    print("\n====== HOSPITAL MANAGEMENT SYSTEM ======")
    print("1. Add Patient")
    print("2. View Patient by Number")
    print("3. View All Patients")
    print("4. Delete Patient")
    print("5. Exit")

def main():
    service = HospitalServiceImpl(PatientRepositoryImpl())

    while True:
        display_menu()
        choice = input("Enter your choice: ")

        try:
            if choice == "1":
                name = input("Enter Patient Name: ")
                age = int(input("Enter Patient Age: "))
                gender = input("Enter Gender (M/F): ")
                disease = input("Enter Disease Description: ")
                service.add_patient(name, age, gender, disease)
                print("Patient added successfully.")

            elif choice == "2":
                patient_no = int(input("Enter Patient Number: "))
                patient = service.get_patient_by_number(patient_no)
                print("\nPatient Found:")
                print(patient)
            elif choice == "3":
                print("\nAll Patients:")
                for p in service.get_all_patients():
                    print(p)
            elif choice == "4":
                patient_no = int(input("Enter Patient Number to Delete: "))
                result = service.delete_patient(patient_no)
                if result:
                    print("Patient deleted successfully.")
                else:
```

```
                print("No patient found with the given number.")
            elif choice == "5":
                print("Exiting system. Thank you!")
                break
            else:
                print(" Invalid option. Please try again.")
        except PatientNumberNotFoundException as e:
            print(e)
        except ValueError:
            print(" Invalid input type. Please enter numeric values where required.")
        except Exception as e:
            print(f" Unexpected error: {e}")

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