

**Міністерство освіти і науки України**  
**Національний технічний університет України «Київський політехнічний**  
**інститут імені Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**

**Кафедра ІІІ**

**Звіт**

до лабораторної роботи № 8 з дисципліни  
«Основи програмування. Частина 2. Методології програмування»

**“ВІДНОШЕННЯ МІЖ КЛАСАМИ ТА ОБ’ЄКТАМИ”**

**Виконав(ла)**

ІП-42 Книр Я. В.

**Перевірив(ла)**

Куценко М. О.

Київ 2025

**Мета лабораторної роботи:** дослідити типи відношень між класами та об'єктами в ООП, навчитися проектувати об'єктно-орієнтовану модель предметної галузі.

Варіант 13

### Варіант 13

---

#### *Реєстратура лікарні: облік хворих та запис до лікарів на прийом*

---

##### **Функціональні вимоги до програмного забезпечення**

1. Управління лікарями
  - 1.1. Можливість додавати лікаря
  - 1.2. Можливість видаляти лікаря
  - 1.3. Можливість змінити дані про лікаря
  - 1.4. Можливість перегляду списку всіх лікарів та їх спеціалізацій
2. Управління хворими
  - 2.1. Можливість додавати хворого
  - 2.2. Можливість видаляти хворого
  - 2.3. Можливість ведення електронної картки хворого
3. Управління розкладом прийому
  - 3.1. Можливість додавати розклад
  - 3.2. Можливість змінювати розклад
  - 3.3. Можливість записувати хворого до лікаря на прийом
4. Пошук
  - 4.1. Можливість пошуку хворого за його даними (прізвище, ім'я)
  - 4.2. Можливість пошуку лікаря
  - 4.3. Можливість отримання розкладу лікаря на певний час

## Діаграма класів :

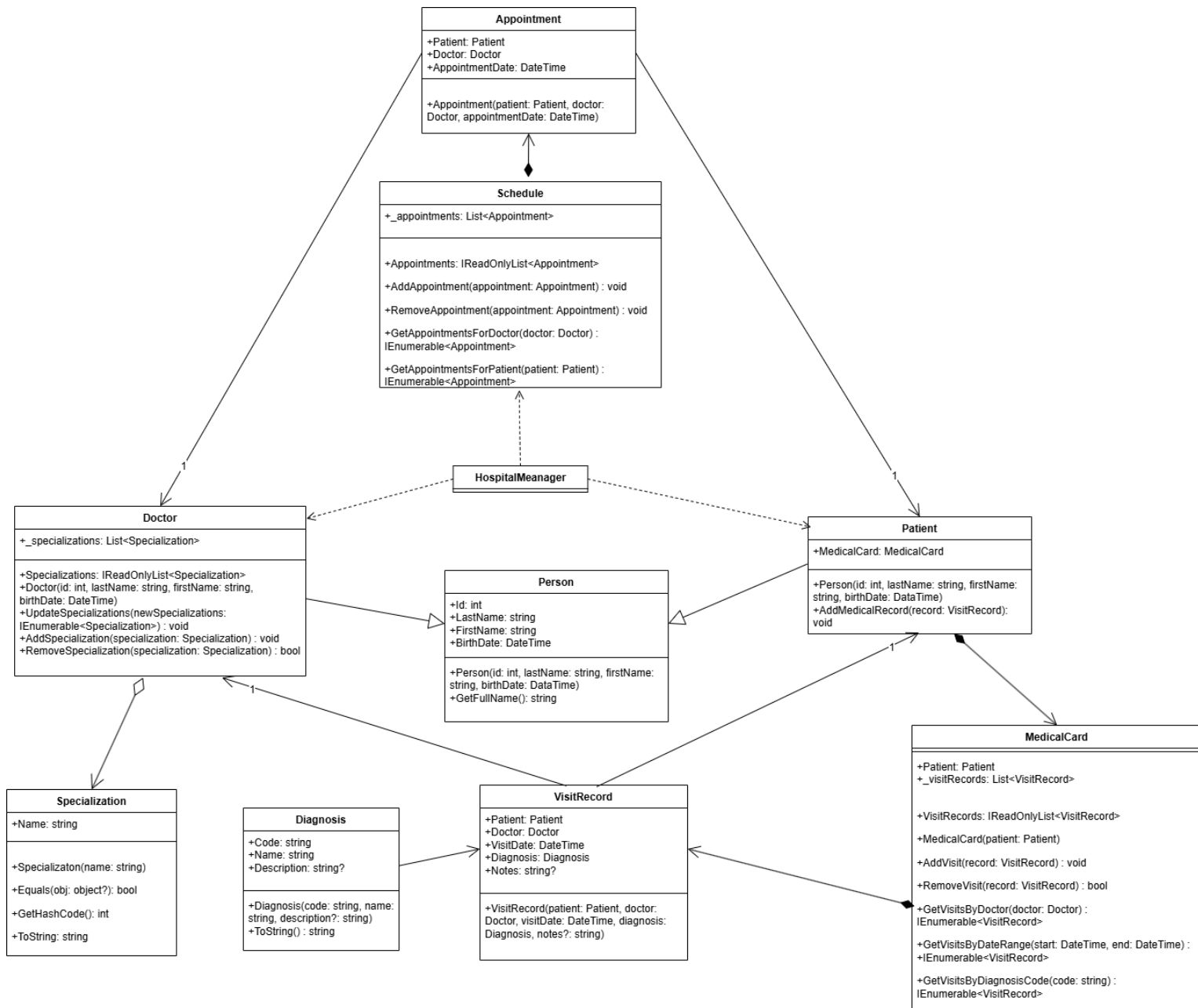


Рисунок 1. Діаграма класів

## Код програми :

- Бібліотека класів

```
using System;
using System.Collections.Generic;

namespace HospitalSystem
{
    public abstract class Person
    {
        public int Id { get; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public DateTime BirthDate { get; set; }

        public Person(int id, string lastName, string firstName, DateTime birthDate)
        {
            if (string.IsNullOrEmpty(lastName) ||
string.IsNullOrEmpty(firstName))
            {
                throw new ArgumentException("Name and surname cannot be empty.");
            }

            Id = id;
            LastName = lastName;
            FirstName = firstName;
            BirthDate = birthDate;
        }

        public string GetFullName() => $"{LastName} {FirstName}";
    }

    public class Patient : Person
    {
        public MedicalCard MedicalCard { get; }

        public Patient(int id, string lastName, string firstName, DateTime birthDate)
            : base(id, lastName, firstName, birthDate)
        {
            MedicalCard = new MedicalCard();
        }

        public void AddMedicalRecord(VisitRecord record)
        {
            if (record == null)
            {
                throw new ArgumentNullException(nameof(record));
            }
            MedicalCard.AddVisit(record);
        }
    }

    public class Doctor : Person
    {
        private readonly List<Specialization> _specializations = new();

        public IReadOnlyList<Specialization> Specializations =>
            _specializations.AsReadOnly();

        public Doctor(int id, string lastName, string firstName, DateTime birthDate):
            base(id, lastName, firstName, birthDate) { }

        public void UpdateSpecializations(IEnumerable<Specialization> newSpecializations)
        {
            if (newSpecializations == null)
            {

```

```

        throw new ArgumentNullException(nameof(newSpecializations));
    }

    var distinctSpecs = newSpecializations.Distinct().ToList();

    if (distinctSpecs.Count > 10)
    {
        throw new InvalidOperationException("A doctor can have maximum 10
specializations.");
    }

    _specializations.Clear();
    _specializations.AddRange(distinctSpecs);
}

public void AddSpecialization(Specialization specialization)
{
    if (specialization == null)
    {
        throw new ArgumentNullException(nameof(specialization));
    }

    if (_specializations.Contains(specialization))
    {
        return;
    }

    if (_specializations.Count >= 10)
    {
        throw new InvalidOperationException("A doctor can have at most 10
specializations.");
    }

    _specializations.Add(specialization);
}

public bool RemoveSpecialization(Specialization specialization)
{
    return _specializations.Remove(specialization);
}
}

public class VisitRecord
{
    public Patient Patient { get; }

    public Doctor Doctor { get; }

    public DateTime VisitDate { get; }

    public Diagnosis Diagnosis { get; }

    public string? Notes { get; }

    public VisitRecord(Patient patient, Doctor doctor, DateTime visitDate, Diagnosis
diagnosis, string? notes = null)
    {
        if (patient == null)
        {
            throw new ArgumentNullException(nameof(patient));
        }

        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor));
        }
    }
}

```

```

        if (diagnosis == null)
        {
            throw new ArgumentNullException(nameof(diagnosis));
        }

        Patient = patient;
        Doctor = doctor;
        VisitDate = visitDate;
        Diagnosis = diagnosis;
        Notes = notes;
    }
}

public class Diagnosis
{
    public string Code { get; set; }

    public string Name { get; set; }

    public string? Description { get; set; }

    public Diagnosis(string code, string name, string? description = null)
    {
        if (string.IsNullOrEmpty(code) || string.IsNullOrEmpty(name))
        {
            throw new ArgumentException("Code and name of diagnosis cannot be
empty.");
        }

        Code = code;
        Name = name;
        Description = description;
    }

    public override string ToString() => $"{Code}: {Name}";
}

public class MedicalCard
{
    private const int MaxRecords = 100;
    private readonly List<VisitRecord> _records = new();

    public IReadOnlyList<VisitRecord> VisitRecords => _records.AsReadOnly();

    public void AddVisit(VisitRecord record)
    {
        if (_records.Count >= MaxRecords)
        {
            throw new InvalidOperationException("The medical card can onle have 100
records.");
        }
        _records.Add(record);
    }

    public bool RemoveVisit(VisitRecord record)
    {
        return _records.Remove(record);
    }
}

public class Appointment
{
    public Patient Patient { get; }

    public Doctor Doctor { get; }

    public DateTime AppointmentDate { get; }
}

```

```

    public Appointment(Patient patient, Doctor doctor, DateTime appointmentDate)
    {
        if (patient == null)
        {
            throw new ArgumentNullException(nameof(patient), "Patient cannot be
null.");
        }

        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor), "Doctor cannot be
null.");
        }

        Patient = patient;
        Doctor = doctor;
        AppointmentDate = appointmentDate;
    }
}

public class Schedule
{
    private readonly List<Appointment> _appointments = new();

    public IReadOnlyList<Appointment> Appointments => _appointments.AsReadOnly();

    public void AddAppointment(Appointment appointment)
    {
        if (appointment == null)
        {
            throw new ArgumentNullException(nameof(appointment), "Appointment cannot
be null.");
        }
        _appointments.Add(appointment);
    }

    public void RemoveAppointment(Appointment appointment)
    {
        if (appointment == null)
        {
            throw new ArgumentNullException(nameof(appointment), "Appointment cannot
be null.");
        }

        bool removed = _appointments.Remove(appointment);
        if (!removed)
        {
            throw new InvalidOperationException("Appointment not found in the
schedule.");
        }
    }

    public IEnumerable<Appointment> GetAppointmentsForDoctor(Doctor doctor)
    {
        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor), "Doctor cannot be
null.");
        }

        var result = new List<Appointment>();

        foreach (var appointment in _appointments)
        {
            if (appointment.Doctor == doctor)
            {
                result.Add(appointment);
            }
        }
    }
}

```

```

        }

        return result;
    }

    public IEnumerable<Appointment> GetAppointmentsForPatient(Patient patient)
    {
        if (patient == null)
        {
            throw new ArgumentNullException(nameof(patient), "Patient cannot be
null.");
        }

        var result = new List<Appointment>();

        foreach (var appointment in _appointments)
        {
            if (appointment.Patient == patient)
            {
                result.Add(appointment);
            }
        }

        return result;
    }
}

public class Specialization
{
    public string Name { get; }

    public Specialization(string name)
    {
        if (name == null)
        {
            throw new ArgumentNullException(nameof(name), "Name cannot be null.");
        }
        Name = name;
    }

    public override bool Equals(object? obj)
    {
        return obj is Specialization other && Name.Equals(other.Name,
StringComparison.OrdinalIgnoreCase);
    }

    public override int GetHashCode()
    {
        return Name.ToLowerInvariant().GetHashCode();
    }

    public override string ToString() => Name;
}

public class HospitalManager
{
    private readonly List<Doctor> _doctors = new();
    private readonly List<Patient> _patients = new();
    private readonly Schedule _schedule = new();

    public void AddDoctor(Doctor doctor)
    {
        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor));
        }
        _doctors.Add(doctor);
    }
}

```



```

public bool RemoveDoctor(int doctorId)
{
    var doctor = _doctors.FirstOrDefault(d => d.Id == doctorId);
    if (doctor == null)
    {
        throw new InvalidOperationException($"Doctor with ID {doctorId} can not
be removed");
    }
    return _doctors.Remove(doctor);
}

public bool UpdateDoctor(int doctorId, string newLastName, string newFirstName,
DateTime newBirthDate, IEnumerable<Specialization>? newSpecializations = null)
{
    var doctor = _doctors.FirstOrDefault(d => d.Id == doctorId);
    if (doctor == null)
    {
        throw new InvalidOperationException($"Doctor with ID {doctorId} can not
be updated.");
    }

    doctor.LastName = newLastName;
    doctor.FirstName = newFirstName;
    doctor.BirthDate = newBirthDate;

    if (newSpecializations != null)
    {
        doctor.UpdateSpecializations(newSpecializations);
    }

    return true;
}

public List<Doctor> GetAllDoctors()
{
    return new(_doctors);
}

public void AddPatient(Patient patient)
{
    if (patient == null)
    {
        throw new ArgumentNullException(nameof(patient));
    }
    _patients.Add(patient);
}

public bool RemovePatient(int patientId)
{
    var patient = _patients.FirstOrDefault(p => p.Id == patientId);
    if (patient == null)
    {
        throw new InvalidOperationException($"Patient with Id {patientId} can not
be removed.");
    }
    return _patients.Remove(patient);
}

public Patient? GetPatientById(int id)
{
    return _patients.FirstOrDefault(p => p.Id == id);
}

public List<Patient> SearchPatientsByName(string lastName, string firstName)
{
    var results = new List<Patient>();

```

```

        foreach (var p in _patients)
        {
            if (p.LastName.Equals(lastName, StringComparison.OrdinalIgnoreCase) &&
                p.FirstName.Equals(firstName, StringComparison.OrdinalIgnoreCase))
            {
                results.Add(p);
            }
        }

        return results;
    }

    public List<Patient> GetAllPatients()
    {
        return new(_patients);
    }

    public void AddAppointment(Appointment appointment)
    {
        if (appointment == null)
        {
            throw new ArgumentNullException(nameof(appointment));
        }

        var time = appointment.AppointmentDate.TimeOfDay;

        if (time < TimeSpan.FromHours(8) || time > TimeSpan.FromHours(19))
        {
            throw new InvalidOperationException("Appointments can only be scheduled
between 8:00 and 19:00.");
        }

        bool conflict = _schedule.GetAppointmentsForDoctor(appointment.Doctor)
            .Any(a => a.AppointmentDate == appointment.AppointmentDate);

        if (conflict)
        {
            throw new InvalidOperationException("The doctor already has an
appointment at this time.");
        }

        _schedule.AddAppointment(appointment);
    }

    public void RemoveAppointment(Appointment appointment)
    {
        if (appointment == null)
        {
            throw new ArgumentNullException(nameof(appointment));
        }
        if (!_schedule.Appointments.Contains(appointment))
        {
            throw new InvalidOperationException("Appointment was not found in
base.");
        }
        _schedule.RemoveAppointment(appointment);
    }

    public bool UpdateAppointment(Appointment oldAppointment, Appointment
newAppointment)
    {
        if (oldAppointment == null || newAppointment == null)
        {
            throw new ArgumentNullException("Appointment can not be null.");
        }

        var time = newAppointment.AppointmentDate.TimeOfDay;

```

```

        if (time < TimeSpan.FromHours(8) || time > TimeSpan.FromHours(19))
        {
            throw new InvalidOperationException("Appointments must be between 8:00
and 19:00.");
        }

        bool conflict = _schedule.GetAppointmentsForDoctor(newAppointment.Doctor)
            .Any(a => a.AppointmentDate == newAppointment.AppointmentDate && a !=
oldAppointment);

        if (conflict)
        {
            throw new InvalidOperationException("The doctor already has an
appointment at this time.");
        }

        _schedule.RemoveAppointment(oldAppointment);
        _schedule.AddAppointment(newAppointment);
        return true;
    }

    public IReadOnlyList<Appointment> GetAllAppointments()
    {
        return _schedule.Appointments.OrderBy(a =>
a.AppointmentDate).ToList().AsReadOnly();
    }

    public void AddVisitRecordToPatient(int patientId, VisitRecord record)
    {
        var patient = GetPatientById(patientId);
        if (patient == null)
        {
            throw new ArgumentException("Patient was not found");
        }
        if (record == null)
        {
            throw new ArgumentNullException(nameof(record));
        }

        patient.MedicalCard.AddVisit(record);
    }

    public IReadOnlyList<VisitRecord> GetVisitRecordsForPatient(int patientId)
    {
        var patient = GetPatientById(patientId);
        if (patient == null)
        {
            throw new ArgumentException("Patient was not found");
        }
        return patient.MedicalCard.VisitRecords;
    }

    public bool RemoveVisitRecordFromPatient(int patientId, VisitRecord record)
    {
        var patient = GetPatientById(patientId);
        if (patient == null || record == null)
        {
            return false;
        }
        return patient.MedicalCard.RemoveVisit(record);
    }

    public List<Doctor> SearchDoctors(string? lastName = null, string? firstName =
null, string? specialization = null)
    {
        var results = new List<Doctor>();

        foreach (var d in _doctors)

```

```

        {
            bool matchesLastName = string.IsNullOrEmpty(lastName) ||
d.LastName.Equals(lastName, StringComparison.OrdinalIgnoreCase);
            bool matchesFirstName = string.IsNullOrEmpty(firstName) ||
d.FirstName.Equals(firstName, StringComparison.OrdinalIgnoreCase);

            bool matchesSpecialization = true;
            if (!string.IsNullOrEmpty(specialization))
            {
                matchesSpecialization = false;
                foreach (var s in d.Specializations)
                {
                    if (s.Name.Equals(specialization,
StringComparison.OrdinalIgnoreCase))
                    {
                        matchesSpecialization = true;
                        break;
                    }
                }
            }

            if (matchesLastName && matchesFirstName && matchesSpecialization)
            {
                results.Add(d);
            }
        }

        return results;
    }

    public IEnumerable<Appointment> GetAppointmentsForDoctor(Doctor doctor)
    {
        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor));
        }
        return _schedule.GetAppointmentsForDoctor(doctor);
    }

    public IEnumerable<Appointment> GetAppointmentsForPatient(Patient patient)
    {
        if (patient == null)
        {
            throw new ArgumentNullException(nameof(patient));
        }
        return _schedule.GetAppointmentsForPatient(patient);
    }

    public IEnumerable<Appointment> GetAppointmentsForDoctorInRange(Doctor doctor,
DateTime startDate, DateTime endDate)
    {
        if (doctor == null)
        {
            throw new ArgumentNullException(nameof(doctor));
        }

        var allAppointments = _schedule.GetAppointmentsForDoctor(doctor);
        var results = new List<Appointment>();

        foreach (var appointment in allAppointments)
        {
            if (appointment.AppointmentDate >= startDate &&
appointment.AppointmentDate <= endDate)
            {
                results.Add(appointment);
            }
        }
    }

```

```

        return results;
    }

    public IEnumerable<Appointment> GetAppointmentsAtTime(DateTime time)
    {
        var results = new List<Appointment>();

        foreach (var appointment in _schedule.Appointments)
        {
            if (appointment.AppointmentDate == time)
            {
                results.Add(appointment);
            }
        }

        return results;
    }
}

```

- Реалізація роботи програми

```

using System;
using System.Collections.Generic;
using HospitalSystem;

class Program
{
    static void Main()
    {
        var manager = new HospitalManager();

        var drJohn = new Doctor(1, "Smith", "John", new DateTime(1975, 3, 15));
        drJohn.AddSpecialization(new Specialization("Pediatrician"));

        var drEmily = new Doctor(2, "Jones", "Emily", new DateTime(1980, 6, 20));
        drEmily.AddSpecialization(new Specialization("Cardiologist"));

        var drMichael = new Doctor(3, "Williams", "Michael", new DateTime(1978, 11,
5));
        drMichael.AddSpecialization(new Specialization("Pediatrician"));

        var drOlivia = new Doctor(4, "Brown", "Olivia", new DateTime(1985, 1, 25));
        drOlivia.AddSpecialization(new Specialization("Orthopedist"));

        var drJames = new Doctor(5, "Taylor", "James", new DateTime(1970, 9, 10));
        drJames.AddSpecialization(new Specialization("Ophthalmologist"));

        manager.AddDoctor(drJohn);
        manager.AddDoctor(drEmily);
        manager.AddDoctor(drMichael);
        manager.AddDoctor(drOlivia);
        manager.AddDoctor(drJames);

        Console.WriteLine("Initial list of doctors:");
        PrintDoctors(manager.GetAllDoctors());

        Console.WriteLine("\nAdding 'Internal Medicine' specialization to Dr. Emily
Jones");
        drEmily.AddSpecialization(new Specialization("Internal Medicine"));

        Console.WriteLine("\nDoctors list after updating Dr. Emily's
specializations:");
        PrintDoctors(manager.GetAllDoctors());

        var drWill = new Doctor(6, "Peterson", "Will", new DateTime(1988, 4, 12));
        drWill.AddSpecialization(new Specialization("Pediatrician"));
        manager.AddDoctor(drWill);
    }
}

```

```

Console.WriteLine("\nAfter adding Dr. Will Peterson:");
PrintDoctors(manager.GetAllDoctors());

bool removed = manager.RemoveDoctor(drJohn.Id);
Console.WriteLine($"Removing Dr. John Smith: {(removed ? "Success" :
"Failed")}");

Console.WriteLine("\nDoctors list after removal:");
PrintDoctors(manager.GetAllDoctors());

var patient1 = new Patient(1, "Johnson", "Anna", new DateTime(2010, 7, 4));
var patient2 = new Patient(2, "Williams", "David", new DateTime(1985, 2, 16));
var patient3 = new Patient(3, "Miller", "Sophia", new DateTime(1990, 12, 30));

manager.AddPatient(patient1);
manager.AddPatient(patient2);
manager.AddPatient(patient3);

Console.WriteLine("\nInitial list of patients:");
PrintPatients(manager.GetAllPatients());

var patient4 = new Patient(4, "Taylor", "Chris", new DateTime(2000, 5, 21));
manager.AddPatient(patient4);

Console.WriteLine("\nAfter adding new patient Chris Taylor:");
PrintPatients(manager.GetAllPatients());

bool patientRemoved = manager.RemovePatient(patient2.Id);
Console.WriteLine($"Removing patient David Williams: {(patientRemoved ?
"Success" : "Failed")}");

Console.WriteLine("\nPatients list after removal:");
PrintPatients(manager.GetAllPatients());

var diagnosis1 = new Diagnosis("J01", "Flu", "Hight fever");
var diagnosis2 = new Diagnosis("I10", "Hypertension", "High blood pressure");

var visit1 = new VisitRecord(patient1, drWill, DateTime.Today.AddDays(-10),
diagnosis1, "Prescribed antibiotics");
var visit2 = new VisitRecord(patient1, drEmily, DateTime.Today.AddDays(-5),
diagnosis2, "Recommended lifestyle changes");

manager.AddVisitRecordToPatient(patient1.Id, visit1);
manager.AddVisitRecordToPatient(patient1.Id, visit2);

Console.WriteLine($"Medical records for patient {patient1.GetFullName()}
(before update):");
PrintVisitRecords(manager.GetVisitRecordsForPatient(patient1.Id));

var diagnosis3 = new Diagnosis("R50", "Cold", "Light coughing");
var visit3 = new VisitRecord(patient1, drMichael, DateTime.Today, diagnosis3,
"Advised rest and hydration");
manager.AddVisitRecordToPatient(patient1.Id, visit3);

Console.WriteLine($"Medical records for patient {patient1.GetFullName()}
(after update):");
PrintVisitRecords(manager.GetVisitRecordsForPatient(patient1.Id));

var appointment1 = new Appointment(patient1, drWill,
DateTime.Today.AddDays(1).AddHours(9));
var appointment2 = new Appointment(patient4, drEmily,
DateTime.Today.AddDays(2).AddHours(10));

manager.AddAppointment(appointment1);
manager.AddAppointment(appointment2);

Console.WriteLine("\nSchedule with two appointments:");
PrintAllAppointments(manager);

```

```

        var updatedAppointment1 = new Appointment(patient1, drWill,
DateTime.Today.AddDays(1).AddHours(11));
        bool updated = manager.UpdateAppointment(appointment1, updatedAppointment1);
        Console.WriteLine($"\\nUpdating appointment for {patient1.GetFullName()} with
Dr. Will Peterson: {(updated ? "Success" : "Failed")}");

        Console.WriteLine("\\nSchedule after updating appointment:");
        PrintAllAppointments(manager);

        var newAppointment = new Appointment(patient4, drOlivia,
DateTime.Today.AddDays(3).AddHours(14));
        manager.AddAppointment(newAppointment);

        Console.WriteLine($"\\nAfter scheduling {patient4.GetFullName()} with Dr.
Olivia Brown:");
        PrintAllAppointments(manager);

        var searchPatients = manager.SearchPatientsByName("Taylor", "Chris");
        Console.WriteLine("\\nSearch patients by name 'Taylor Chris:");
        PrintPatients(searchPatients);

        var searchDoctors = manager.SearchDoctors(specialization: "Pediatrician");
        Console.WriteLine("\\nSearch doctors with specialization 'Pediatrician:");
        PrintDoctors(searchDoctors);

        var startRange = DateTime.Today;
        var endRange = DateTime.Today.AddDays(5);
        var drWillSchedule = manager.GetAppointmentsForDoctorInRange(drWill,
startRange, endRange);
        Console.WriteLine($"\\nDr. Will Peterson's schedule from
{startRange.ToShortDateString()} to {endRange.ToShortDateString():");
        PrintAppointments(drWillSchedule);
    }

    static void PrintDoctors(IEnumerable<Doctor> doctors)
    {
        int i = 1;
        foreach (var d in doctors)
        {
            Console.WriteLine($"{{i}}. Dr. {{d.GetFullName()}} - Specializations:
{string.Join(", ", d.Specializations.Select(s => s.Name))}");
            i++;
        }
    }

    static void PrintPatients(IEnumerable<Patient> patients)
    {
        int i = 1;
        foreach (var p in patients)
        {
            Console.WriteLine($"{{i}}. {{p.GetFullName()}}, BirthDate:
{{p.BirthDate.ToShortDateString()}}");
            i++;
        }
    }

    static void PrintVisitRecords(IEnumerable<VisitRecord> records)
    {
        foreach (var rec in records)
        {
            Console.WriteLine($"- Date: {{rec.VisitDate.ToShortDateString()}} , Doctor:
Dr. {{rec.Doctor.GetFullName()}} , Diagnosis: {{rec.Diagnosis}} , Notes: {{rec.Notes}}");
        }
    }

    static void PrintAppointments(IEnumerable<Appointment> appointments)

```

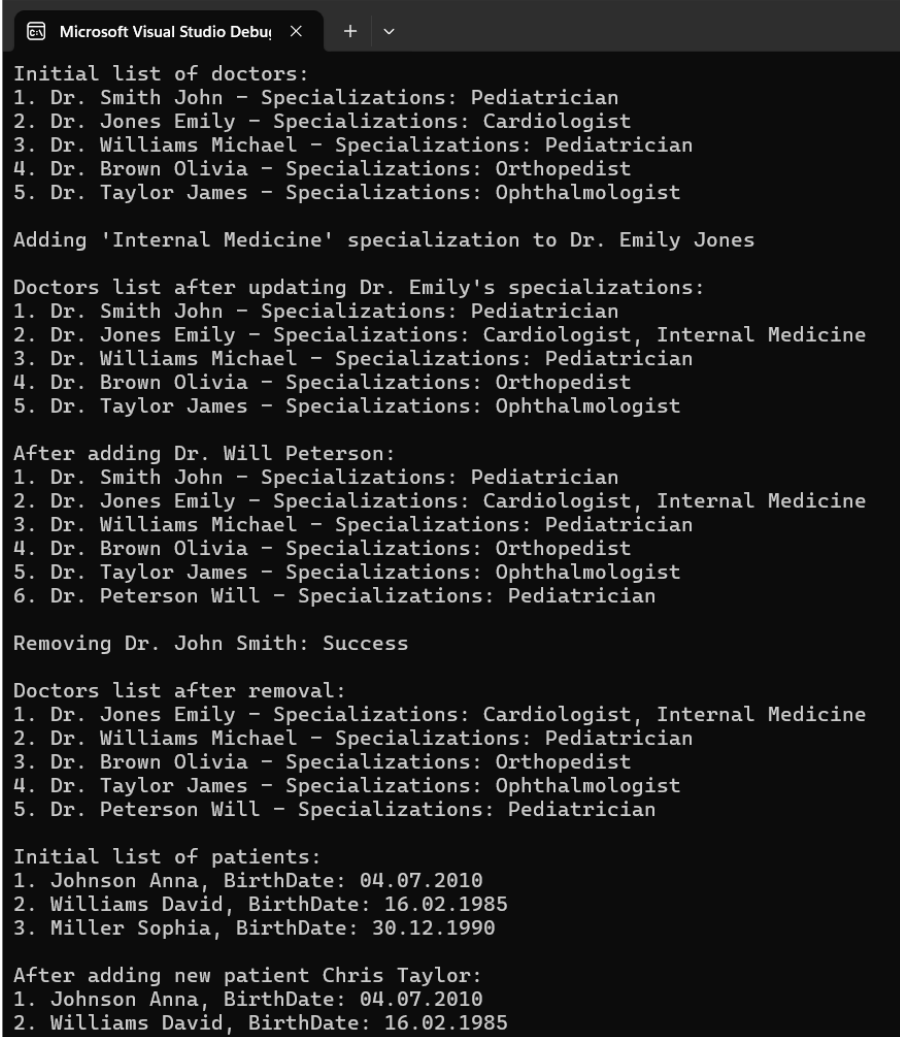
```
{
    foreach (var appt in appointments.OrderBy(a => a.AppointmentDate))
    {
        Console.WriteLine($"- {appt.AppointmentDate}: Patient
{appt.Patient.GetFullName()} with Dr. {appt.Doctor.GetFullName()}");
    }
}

static void PrintAllAppointments(HospitalManager manager)
{
    var allAppointments = manager.GetAllAppointments();
    Console.WriteLine("Full schedule of all appointments:");
    PrintAppointments(allAppointments);
}
}
```



## Приклад роботи програми:

На рисунках 2, 3 та 4 зображено приклад роботи програми під час тестування її функціоналу.



```
Microsoft Visual Studio Debug Console
Initial list of doctors:
1. Dr. Smith John - Specializations: Pediatrician
2. Dr. Jones Emily - Specializations: Cardiologist
3. Dr. Williams Michael - Specializations: Pediatrician
4. Dr. Brown Olivia - Specializations: Orthopedist
5. Dr. Taylor James - Specializations: Ophthalmologist

Adding 'Internal Medicine' specialization to Dr. Emily Jones

Doctors list after updating Dr. Emily's specializations:
1. Dr. Smith John - Specializations: Pediatrician
2. Dr. Jones Emily - Specializations: Cardiologist, Internal Medicine
3. Dr. Williams Michael - Specializations: Pediatrician
4. Dr. Brown Olivia - Specializations: Orthopedist
5. Dr. Taylor James - Specializations: Ophthalmologist

After adding Dr. Will Peterson:
1. Dr. Smith John - Specializations: Pediatrician
2. Dr. Jones Emily - Specializations: Cardiologist, Internal Medicine
3. Dr. Williams Michael - Specializations: Pediatrician
4. Dr. Brown Olivia - Specializations: Orthopedist
5. Dr. Taylor James - Specializations: Ophthalmologist
6. Dr. Peterson Will - Specializations: Pediatrician

Removing Dr. John Smith: Success

Doctors list after removal:
1. Dr. Jones Emily - Specializations: Cardiologist, Internal Medicine
2. Dr. Williams Michael - Specializations: Pediatrician
3. Dr. Brown Olivia - Specializations: Orthopedist
4. Dr. Taylor James - Specializations: Ophthalmologist
5. Dr. Peterson Will - Specializations: Pediatrician

Initial list of patients:
1. Johnson Anna, BirthDate: 04.07.2010
2. Williams David, BirthDate: 16.02.1985
3. Miller Sophia, BirthDate: 30.12.1990

After adding new patient Chris Taylor:
1. Johnson Anna, BirthDate: 04.07.2010
2. Williams David, BirthDate: 16.02.1985
```

Рисунок 2. Робота програми

```

Microsoft Visual Studio Debu  x  +  v
3. Miller Sophia, BirthDate: 30.12.1990
4. Taylor Chris, BirthDate: 21.05.2000

Removing patient David Williams: Success

Patients list after removal:
1. Johnson Anna, BirthDate: 04.07.2010
2. Miller Sophia, BirthDate: 30.12.1990
3. Taylor Chris, BirthDate: 21.05.2000

Medical records for patient Johnson Anna (before update):
- Date: 27.05.2025, Doctor: Dr. Peterson Will, Diagnosis: J01: Flu, Notes: Prescribed antibiotics
- Date: 01.06.2025, Doctor: Dr. Jones Emily, Diagnosis: I10: Hypertension, Notes: Recommended lifestyle changes

Medical records for patient Johnson Anna (after update):
- Date: 27.05.2025, Doctor: Dr. Peterson Will, Diagnosis: J01: Flu, Notes: Prescribed antibiotics
- Date: 01.06.2025, Doctor: Dr. Jones Emily, Diagnosis: I10: Hypertension, Notes: Recommended lifestyle changes
- Date: 06.06.2025, Doctor: Dr. Williams Michael, Diagnosis: R50: Cold, Notes: Advised rest and hydration

Schedule with two appointments:
Full schedule of all appointments:
- 07.06.2025 9:00:00: Patient Johnson Anna with Dr. Peterson Will
- 08.06.2025 10:00:00: Patient Taylor Chris with Dr. Jones Emily

Updating appointment for Johnson Anna with Dr. Will Peterson: Success

Schedule after updating appointment:
Full schedule of all appointments:
- 07.06.2025 11:00:00: Patient Johnson Anna with Dr. Peterson Will
- 08.06.2025 10:00:00: Patient Taylor Chris with Dr. Jones Emily

After scheduling Taylor Chris with Dr. Olivia Brown:
Full schedule of all appointments:
- 07.06.2025 11:00:00: Patient Johnson Anna with Dr. Peterson Will
- 08.06.2025 10:00:00: Patient Taylor Chris with Dr. Jones Emily
- 09.06.2025 14:00:00: Patient Taylor Chris with Dr. Brown Olivia

Search patients by name 'Taylor Chris':
1. Taylor Chris, BirthDate: 21.05.2000

Search doctors with specialization 'Pediatrician':

```

Рисунок 3. Работа программы

```

Search doctors with specialization 'Pediatrician':
1. Dr. Williams Michael - Specializations: Pediatrician
2. Dr. Peterson Will - Specializations: Pediatrician

Dr. Will Peterson's schedule from 06.06.2025 to 11.06.2025:
- 07.06.2025 11:00:00: Patient Johnson Anna with Dr. Peterson Will

```

Рисунок 4. Работа программы

**Висновок:** В процесі виконання лабораторної роботи було досліджено дослідити типи відношень між класами та об'єктами в ООП, було отримано навички проектування об'єктно-орієнтовану модель предметної галузі. Результатом лабораторної роботи стало написання програми, в якій було реалізовано попередньо спроектовану об'єктно орієнтовану модель. Вона представляє собою систему реєстратури в лікарні, яка веде облік хворих та запис до лікарів на прийом. Було розроблено 10 типів даних, серед яких Person, Patient, Doctor, Schedule, Appointment, VisitRecord, Diagnosis, Specialization, MedicalCard та HospitalManager, застосовано основні види відношень, такі як залежність, асоціація, узагальнення, реалізація та деякі їх підтипи, та оброблено виключення. Також було застосовано базові принципи ООП. Програмний інтерфейс представляє собою демонстрацію використання всіх класів предметної галузі шляхом створення об'єктів та їх застосування.