МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

Автоматизованих Систем Обробки Інформації та Управління

**КУРСОВА РОБОТА**

з Програмування. «Структури даних та алгоритми»

на тему:  
Додаток для реєстратури лікарні

Студента 1-го курсу, групи ІС-91

Мініна Павла Андрійовича

Спеціальності 126  
«Інформаційні управляючі системи та технології»

Керівник ст.вик. Вітковська І. І.

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ – 2020 рік

# 

# **ЗМІСТ**

[ЗМІСТ 2](#_Toc40523792)

[ПОСТАНОВКА ЗАДАЧІ](#_Toc40523793) 3

[ВСТУП](#_Toc40523793) 4

[ОСНОВНА ЧАСТИНА](#_Toc40523793) 5-12

[1.Аналіз предметної області](#_Toc40523793) 5

[[2.Об’єктно орієнтована модель програми](#_Toc40523793)](#_Toc40523793) 5

[[3.Функціональні вимоги програми](#_Toc40523793)](#_Toc40523793) 6

[[4.Опис методів програми](#_Toc40523793)](#_Toc40523793) 8

[[5.Календарний план](#_Toc40523793)](#_Toc40523793) 11

6.Діаграма класів програмного забезпечення12

[7.Інструкція по використанню 13](#_Toc40523793)

[8.ТЕСТУВАННЯ 14](#_Toc40523793)

[ЛІСТИНГ ПРОГРАМИ 1](#_Toc40523793)7

[АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ](#_Toc40523793) 28

[ВИКОРИСТАНА ЛІТЕРАТУРА](#_Toc40523793) 29

**ПОСТАНОВКА ЗАДАЧІ**

1. Спроектувати об’єктно-орієнтовану модель предметної галузі згідно з варіантом, визначивши необхідні для цього класи та їх структуру. При об’єктно-орієнтованому проектуванні продемонструвати застосування всіх видів відношень.
2. Написати програму мовою С#, у якій реалізувати попередньо спроектовану об’єктно-орієнтовану модель.
3. Предметну галузь реалізувати окремим проектом – динамічною бібліотекою (С#).
4. Інтерфейс користувача, наприклад, введення\виведення з консолі, реалізовувати окремим проектом (С#). Код інтерфейсу має бути простим (демонструється використання класів предметної галузі шляхом створення об’єктів та їх застосування).

Реєстратура лікарні. Запис до лікарів на прийом.

Реєстратура надає дані стосовно наявності лікарів та розкладу

прийому хворих. Хворим можливо записатись на прийом до

лікаря, якщо є вільний час у розкладі лікаря. В реєстратурі

ведуться картки відвідування хворими лікарні, в які записується

час відвідання лікаря, діагноз та лікар, що його поставив.

**ВСТУП**

Підсумовуючи задачу, на виході нам потрібна працююча програма, яка не тільки повинна бути логічно виконана та бути зручною для використання, а ще й включати у собі значну частину вивченого матеріалу за курс. Ціллю цієї КР я бачу узагальнення та закріплення отриманих знань з ООП та перший досвід у створенні реально працюючої програми для реального об’єкту у світі.

**ОСНОВНА ЧАСТИНА**

**1. Аналіз предметної області**

Додаток “Registry Helper” – спеціально розроблена програма для використання у реєстратурі лікарні. Вона повинна мати функції, які значно полегшують роботу працівників шляхом автоматизації ручної роботи. Вже на етапі планування зрозуміло, що в програмі повинні бути реалізовані класи доктора, пацієнта та робочого графіку (у процесі до них долучилися ще й класи лікарні та людини).

Перелік основних завдань програми:

1. Додавання лікарів до бази даних;
2. Перевірка наявності запису на прийом у БД;
3. Надавання графіку роботи лікарів;
4. Запис на прийом до лікаря;
5. Надання інформації про записи до лікаря;
6. Створення карток відвідування хворими лікарні і їх збереження у БД.

Ці дії для зручності повинні бути легкодоступними, та потрібна можливість необмеженого їх використання протягом одного запуску програми. На етапі планування також була розроблена система запису та зчитування даних з текстових файлів з метою запобігти втраті даних при перезапуску програми .

**2. Об’єктно-орієнтоване проектування програми**

Об’єктами у програмі вирішено було використовувати екземпляри класів Doctor, Patient, WorkSchedule. Клас Person – абстрактний, а Doctor і Patient – від нього наслідуються. Клас Hospital має тільки статичні поля та методи, так як нам не потрібна велика кількість екземплярів цього класу. Кожен доктор повинен мати свій графік роботи, який представлений об’єктом класа WorkSchedule.

Важливою задачею було збереження інформації про ці об’єкти не тільки в оперативній пам’яті. Тому, мною було реалізовано запис та зчитування інформації з файлів. Докладніше про це – у функціональних вимогах програми.

**3. Функціональні вимоги програми**

По-перше, додаток повинен при запуску зчитувати вже існуючу інформацію про докторів та їх заплановані зустрічі. Для цього було використано статичний конструктор класу Hospital, у якому викликаються об’єкти класу StreamReader, який знаходиться у просторі System.IO. Спочатку з файлу Doctors.txt зчитуються імена докторів, що працюють у лікарні. Потім для кожного з цих лікарів отримуємо інформацію про них (наприклад ім’я, перелік робочих днів тощо) з файлу info.txt. Далі, якщо існує файл з іменем доктора у папці Doctors, зчитуємо його заплановані зустрічі. Уся ця інформація передається у конструктори класу Doctor, а усі створені екземпляри записуються у статичний список.

Абстрактний клас Person має простий конструктор та 2 поля: ім’я людини та її вік. Ці якості має і доктор, і пацієнт.

Доктор, у своєму класі, має декілька перевантажених конструкторів та методи перевірки присутності на робочому місці, планування прийомів хворих, також за допомогою методів можна перевірити, чи можливо записатися на вказану дату, та прийняти пацієнта.

Клас WorkSchedule, крім полів початку-кінця робочого дня, зміни, кількості робочих годин та масиву робочих днів неділі, має метод, що перевіряє чи працює власник цього графіку роботи у вказаний день.

Patient, в свою чергу, крім успадкованих полів name і age, має поля з діагнозом, та відомими скаргами на здоров’я. Реалізовано метод SeeDoctor. За замовченням діагноз пацієнта невідомий.

Ну і нарешті, клас Hospital, окрім вже сказаного раніше, має методи найняття лікарів та запуску основної програми.

Зовнішній вигляд “Registry Helper” ’а реалізовано у вигляді консольного додатку з простим інтерфейсом. Користувач вводить різні символи, значення котрих відповідають виклику потрібних функцій програми. У кінці кожної дії користувачу показується підказка, за допомогою якої він ніколи не заплутається в інтерфейсі.

Через те, що індексатор не може бути статичним, для звернення до доктору за іменем, а не чисельним індексом, було реалізовано метод Doc.

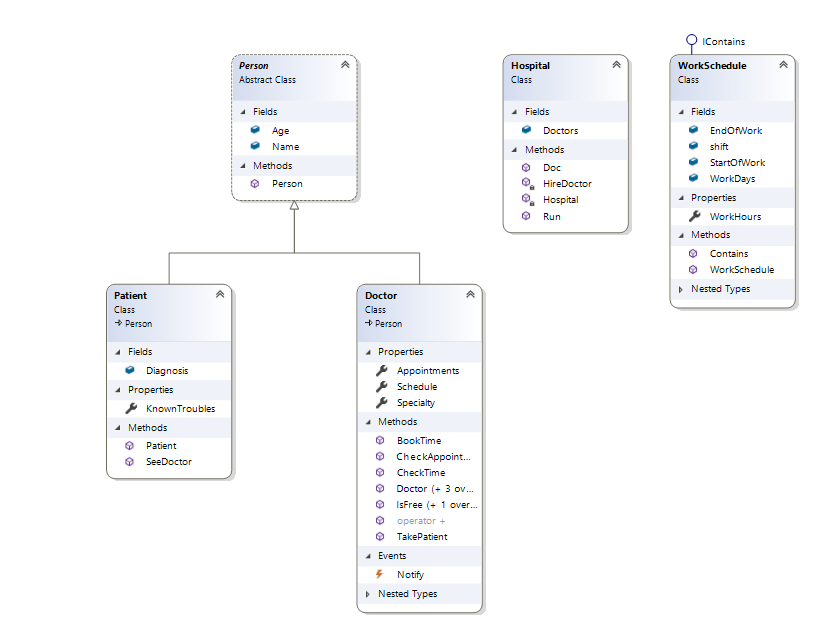
**4. Опис методів програми**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Назва  класу | Назва функції | Призначення  функції | Опис  вхідних  параметрів | Опис  вихідних  параметрів |
| 1. | Person | - | - | - | - |
| 2. | Doctor : Person | 1. IsFree() 2. IsFree(DateTime) 3. TakePatient(Patient) 4. CheckTime(DateTime)     e) BookTime(DateTime),  operator +(DateTime)  f) CheckAppointment(DateTime) | a) Перевіряється чи працює зараз доктор.  b) Перевіряється чи працює доктор у наданий момент.  с) Якщо всі умови виконуються, то доктор приймає пацієнта.  d) Перевіряється, чи можна записатися на прийом до доктора у наданий час  e) якщо це можливо, записуємо пацієнта до доктора на прийом  f) перевіряємо чи є запис у доктора на наданий час | a) без параметрів  b) дата DateTime  c) пацієнт Patient   1. дата DateTime 2. дата DateTime 3. дата DateTime | a) значення bool: true або false  b) значення bool: true або false  c) void; створюється картка відвідування  d) значення bool: true або false  e) void; додаємо запис до списку, робимо запис у файл  f) значення bool: true або false |
| 3. | WorkSchedule | a) Contains(DayOfWeek) | a) перевіряємо чи працює доктор у цей день | a) день неділі DayOfWeek | a) значення bool: true або false |
| 4. | Patient : Person | a) SeeDoctor(Doctor) | a) Якщо всі умови виконуються, то доктор приймає пацієнта. | a) доктор Doctor | a)void; створюється картка відвідування |
| 5. | Hospital | a)HireDoctor()  b) Doc(string)  c) Run() | a) Додається новий доктор, створений з даних, отриманих з консолі.  b) Знаходить доктора у списку за іменем  c) Запускає консольний додаток | a) без параметрів  b) string  с) без параметрів | a) void; створюється новий доктор  b) доктора Doctor  с) void |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Назва етапів курсової роботи | Термін виконання етапів роботи | Підпис керівника, студента |
| 1 | Отримання теми курсової роботи |  |  |
| 2 | Підготовка ТЗ |  |  |
| 3 | Аналіз предметної області |  |  |
| 4 | Проектування архітектури програмної системи |  |  |
| 5 | Розробка сценарію роботи програми |  |  |
| 6 | Узгодження з керівником інтерфейсу користувача |  |  |
| 7 | Розробка програмного забезпечення |  |  |
| 8 | Узгодження з керівником плану тестування |  |  |
| 9 | Тестування програми |  |  |
| 10 | Підготовка пояснювальної записки |  |  |
| 11 | Здача курсової роботи на перевірку |  |  |
| 12 | Захист курсової роботи |  |  |

**5. Календарний план**

**6. Діаграма класів програмного забезпечення**

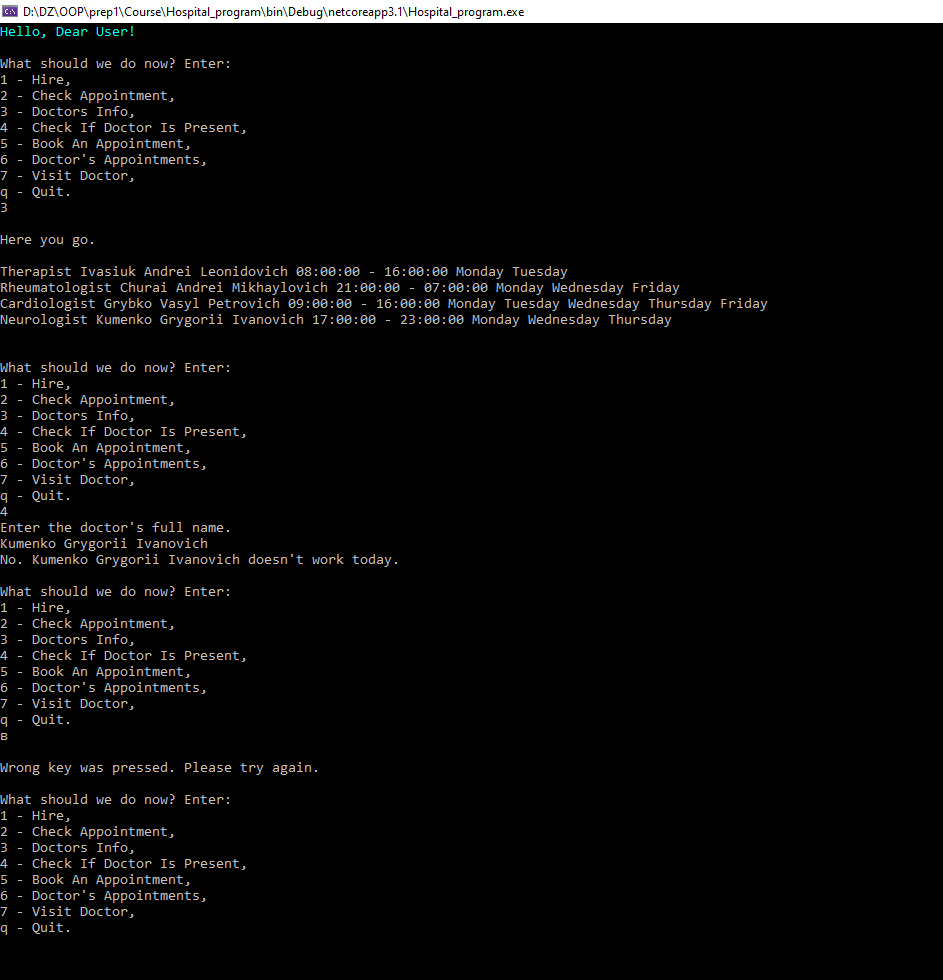


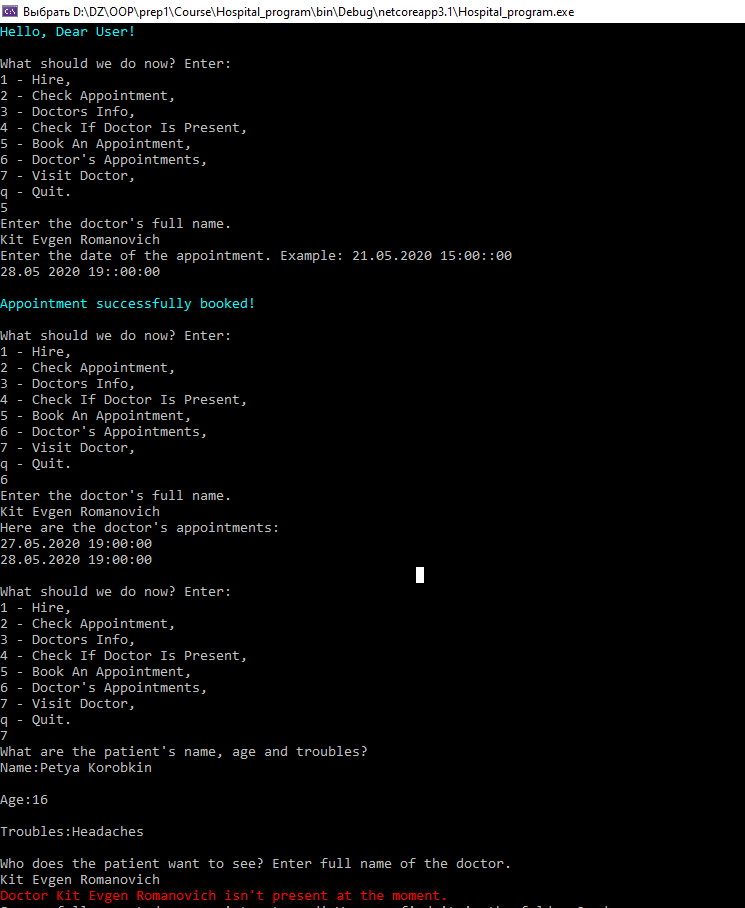
**7. Інструкція по використанню**

Слідувати рекомендаціям інтерфейсної частини програми. Не варто модифікувати файли без досконалого знання логіки програми.

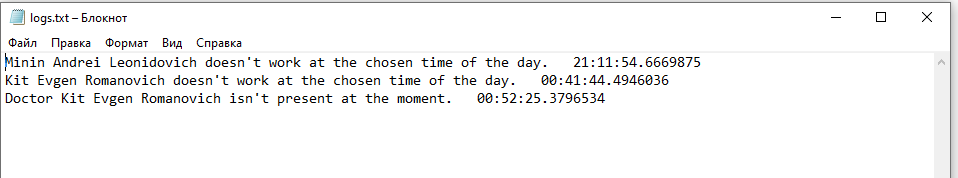
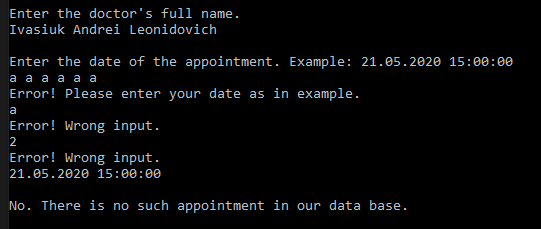
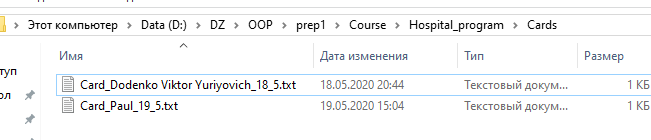
**8.ТЕСТУВАННЯ**

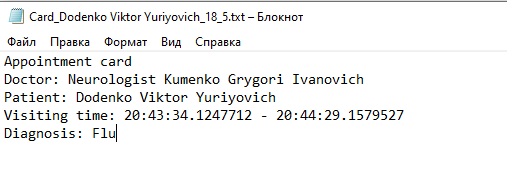
Програма працює коректно, усі файли зчитуються і заповнюються без проблем. Методи усіх класів виконують задумані дії. Також оброблені усі випадки неправильно введених даних.









**ЛІСТИНГ ПРОГРАМИ**

**Hospital\_Lib.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace Hospital\_Lib

{

public abstract class Person

{

public Person(string name, int age)

{

Name = name;

Age = age;

}

public string Name;

public int Age;

}

public class Patient : Person

{

public string Diagnosis;

public string KnownTroubles { get; }

public Patient(string name, int age, string troubles) : base(name, age)

{

KnownTroubles = troubles;

Diagnosis = "Unknown";

}

public void SeeDoctor(Doctor dr)

{

dr.TakePatient(this);

}

}

public class Doctor : Person

{

public WorkSchedule Schedule { get; }

public string Specialty { get; }

public List<DateTime> Appointments { get; }

public Doctor(string name, int age, string Specialty, WorkSchedule graphic) : base(name, age)

{

Appointments = new List<DateTime>();

this.Specialty = Specialty;

Schedule = graphic;

}

public Doctor(string name, int age, string Specialty, int hours, TimeSpan time, DayOfWeek[] WorkDays) : base(name, age)

{

Appointments = new List<DateTime>();

this.Specialty = Specialty;

Schedule = new WorkSchedule(hours, time, WorkDays);

}

public Doctor(string name, int age, string Specialty, List<DateTime> appointments, WorkSchedule graphic) : base(name, age)

{

Appointments = new List<DateTime>();

for (int i = 0; i < appointments.Count; i++)

{

Appointments.Add(appointments[i]);

}

this.Specialty = Specialty;

Schedule = graphic;

}

public Doctor(string name, int age, string Specialty, List<DateTime> appointments, int hours, TimeSpan time, DayOfWeek[] WorkDays) : base(name, age)

{

Appointments = new List<DateTime>();

for (int i = 0; i < appointments.Count; i++)

{

Appointments.Add(appointments[i]);

}

this.Specialty = Specialty;

Schedule = new WorkSchedule(hours, time, WorkDays);

}

public bool IsFree()

{

if ((Schedule.shift == WorkSchedule.Shift.Day))

{

return ((DateTime.Now.TimeOfDay >= Schedule.StartOfWork) && (DateTime.Now.TimeOfDay <= Schedule.EndOfWork) && (Schedule.Contains(DateTime.Now.DayOfWeek)));

}

else

{

if ((DateTime.Now.TimeOfDay >= Schedule.StartOfWork)) return (Schedule.Contains(DateTime.Now.DayOfWeek));

return ((DateTime.Now.TimeOfDay <= Schedule.EndOfWork) && Schedule.Contains(DateTime.Now.DayOfWeek - 1));

}

}

public bool IsFree(DateTime dt)

{

if ((Schedule.shift == WorkSchedule.Shift.Day))

{

return ((dt.TimeOfDay >= Schedule.StartOfWork) && (dt.TimeOfDay <= Schedule.EndOfWork) && (Schedule.Contains(dt.DayOfWeek)));

}

else

{

if ((dt.TimeOfDay >= Schedule.StartOfWork)) return (Schedule.Contains(dt.DayOfWeek));

return ((dt.TimeOfDay <= Schedule.EndOfWork) && Schedule.Contains(dt.DayOfWeek - 1));

}

}

public void TakePatient(Patient patient)

{

try

{

if (IsFree())

{

bool appointment = true;

for (int i = 0; i < Appointments.Count; i++)

{

if (DateTime.Now > Appointments[i] - new TimeSpan(0, 30, 0) && DateTime.Now < Appointments[i] + new TimeSpan(0, 30, 0))

{

Notify?.Invoke("The doctor has an appointment at the moment. The patient can wait in the queue.");

appointment = false;

}

}

if (appointment)

{

Directory.CreateDirectory(@"D:\DZ\OOP\prep1\Course\Hospital\_program\Cards");

TimeSpan time1 = DateTime.Now.TimeOfDay;

Console.WriteLine("Type something in if you have returned.");

Console.ReadLine();

Console.WriteLine("What was the Diagnosis?");

string Diagnosis = Console.ReadLine();

File.WriteAllText(@"D:\DZ\OOP\prep1\Course\Hospital\_program\Cards\Card\_" + patient.Name + "\_" + DateTime.Now.Day + "\_" + DateTime.Now.Month + ".txt", "Appointment card\nDoctor: " + Specialty + " " + Name + "\nPatient: " + patient.Name + "\nVisiting time: " + time1 + " - " + DateTime.Now.TimeOfDay + "\nDiagnosis: " + Diagnosis);

patient.Diagnosis = Diagnosis;

Console.WriteLine("Successfully created an appointment card! You can find it in the folder Cards.");

}

}

else

{

throw new Exception("Doctor " + Name + " isn't present at the moment.");

}

}

catch (Exception e)

{

Notify?.Invoke(e.Message);

}

}

public static Doctor operator +(Doctor doctor, DateTime date)

{

doctor.BookTime(date);

return doctor;

}

public delegate void LogHandler(string message);

public static event LogHandler Notify = (string message) =>

{

Directory.CreateDirectory(@"D:\DZ\OOP\prep1\Hospital\_program\Program\logs");

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(message);

Console.ResetColor();

StreamWriter sw = new StreamWriter(@"D:\DZ\OOP\prep1\Course\Hospital\_program\logs\logs\_"+DateTime.Now.Day+"\_"+ DateTime.Now.Month+".txt", true, Encoding.Default);

sw.WriteLine(message + " " + DateTime.Now.TimeOfDay);

sw.Close();

};

public bool CheckTime(DateTime date)

{

if (IsFree(date) != true)

{

Notify?.Invoke(Name + " doesn't work at the chosen time of the day.");

return false;

}

int i = 0;

while (i < Appointments.Count)

{

if (DateTime.Now > Appointments[i])

{

Appointments.RemoveAt(i);

}

else { i++; }

}

for (i = 0; i < Appointments.Count; i++)

{

if (date > Appointments[i] - new TimeSpan(0, 30, 0) && date < Appointments[i] + new TimeSpan(0, 30, 0))

{

Notify?.Invoke("The chosen time is already taken.");

return false;

}

}

return true;

}

public void BookTime(DateTime date)

{

if (CheckTime(date))

{

Appointments.Add(date);

StreamWriter sw = new StreamWriter(@"D:\DZ\OOP\prep1\Course\Hospital\_program\Doctors\" + Name + ".txt", true, Encoding.Default);

sw.WriteLine(date);

sw.Close();

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("\nAppointment successfully booked!");

Console.ResetColor();

}

}

public bool CheckAppointment(DateTime date)

{

return Appointments.Contains(date);

}

}

public class WorkSchedule:IContains

{

public enum Shift { Day, Night };

public Shift shift;

public int WorkHours { get; }

public TimeSpan StartOfWork, EndOfWork;

public DayOfWeek[] WorkDays;

public WorkSchedule(int hours, TimeSpan time, DayOfWeek[] WorkDays)

{

this.WorkDays = new DayOfWeek[WorkDays.Length];

this.WorkDays = WorkDays;

shift = Shift.Day;

WorkHours = hours;

StartOfWork = time;

EndOfWork = StartOfWork + new TimeSpan(WorkHours, 0, 0);

if (EndOfWork.Days > 0)

{

EndOfWork -= new TimeSpan(1, 0, 0, 0);

shift = Shift.Night;

}

}

public bool Contains(DayOfWeek day)

{

for (int i = 0; i < WorkDays.Length; i++)

{

if (WorkDays[i] == day) { return true; }

}

return false;

}

}

public class Hospital

{

public static List<Doctor> Doctors;

static Hospital()

{

Doctors = new List<Doctor> { };

using (StreamReader sr = new StreamReader(@"D:\DZ\OOP\prep1\Course\Hospital\_program/Doctors.txt", Encoding.Default))

{

using (StreamReader sr2 = new StreamReader(@"D:\DZ\OOP\prep1\Course\Hospital\_program/info.txt", Encoding.Default))

{

string name;

while ((name = sr.ReadLine()) != null)

{

if (File.Exists(@"D:\DZ\OOP\prep1\Course\Hospital\_program\Doctors/" + name + ".txt"))

{

using (StreamReader sr3 = new StreamReader(@"D:\DZ\OOP\prep1\Course\Hospital\_program\Doctors/" + name + ".txt", Encoding.Default))

{

string dates;

List<DateTime> dti = new List<DateTime> { };

while ((dates = sr3.ReadLine()) != null)

{

string[] dates\_int = dates.Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

dti.Add(new DateTime(int.Parse(dates\_int[2]), int.Parse(dates\_int[1]), int.Parse(dates\_int[0]), int.Parse(dates\_int[3]), int.Parse(dates\_int[4]), int.Parse(dates\_int[5])));

}

string info = sr2.ReadLine();

string[] info\_words = info.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

string[] Start\_time = info\_words[3].Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

DayOfWeek[] Days = new DayOfWeek[info\_words.Length - 4];

for (int i = 4; i < info\_words.Length; i++)

{

Days[i - 4] = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), info\_words[i]);

}

Doctor test = new Doctor(name, int.Parse(info\_words[0]), info\_words[1], dti, int.Parse(info\_words[2]), new TimeSpan(int.Parse(Start\_time[0]), int.Parse(Start\_time[1]), int.Parse(Start\_time[2])), Days);

Doctors.Add(test);

}

}

else

{

string info = sr2.ReadLine();

string[] info\_words = info.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

string[] Start\_time = info\_words[3].Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

DayOfWeek[] Days = new DayOfWeek[info\_words.Length - 4];

for (int i = 4; i < info\_words.Length; i++)

{

Days[i - 4] = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), info\_words[i]);

}

Doctor test = new Doctor(name, int.Parse(info\_words[0]), info\_words[1], int.Parse(info\_words[2]), new TimeSpan(int.Parse(Start\_time[0]), int.Parse(Start\_time[1]), int.Parse(Start\_time[2])), Days);

Doctors.Add(test);

}

}

}

}

}

private static void HireDoctor()

{

Console.WriteLine("What is the doctor's name?");

string name = Console.ReadLine();

Console.WriteLine("What is the doctor's age?");

string input = Console.ReadLine();

int age;

bool isParsable = int.TryParse(input, out age);

while (!isParsable)

{

Console.WriteLine("Error! Please enter a number. ");

input = Console.ReadLine();

isParsable = int.TryParse(input, out age);

}

Console.WriteLine("What is " + name + "'s specialty? Please enter 1 (one) word.");

string Specialty = Console.ReadLine();

while (Specialty.Contains(" "))

{

Console.WriteLine("Error! Please enter a single word. ");

Specialty = Console.ReadLine();

}

Console.WriteLine("How many work hours will they have?");

input = Console.ReadLine();

isParsable = int.TryParse(input, out int hours);

while (!isParsable)

{

Console.WriteLine("Error! Please enter a number. ");

input = Console.ReadLine();

isParsable = int.TryParse(input, out hours);

}

Console.WriteLine("At what time does the doctor's work start? Example: 08:30:00");

string[] Start\_time = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

bool num = true;

while (num)

{

num = false;

while (Start\_time.Length != 3)

{

Console.WriteLine("Error! Wrong input.");

Start\_time = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

}

for (int i = 0; i < 3; i++)

{

if (!int.TryParse(Start\_time[i], out \_))

{

Console.WriteLine("Error! Please enter your date as in example. ");

num = true;

break;

}

}

}

bool again;

Console.WriteLine("And finally on what days will they work?");

string[] input\_Days;

do

{

again = false;

input\_Days = Console.ReadLine().Split(new char[] { ' ', '.', ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < input\_Days.Length; i++)

{

if (input\_Days[i] != "Monday" && input\_Days[i] != "Tuesday" && input\_Days[i] != "Wednesday" && input\_Days[i] != "Thursday" && input\_Days[i] != "Friday" && input\_Days[i] != "Sunday" && input\_Days[i] != "Saturday")

{

Console.WriteLine("Wrong data entered! Please try again.");

again = true;

break;

}

}

} while (again) ;

DayOfWeek[] Days = new DayOfWeek[input\_Days.Length];

for (int i = 0; i < input\_Days.Length; i++)

{

Days[i] = (DayOfWeek)Enum.Parse(typeof(DayOfWeek), input\_Days[i]);

}

Doctors.Add(new Doctor(name, age, Specialty, hours, new TimeSpan(int.Parse(Start\_time[0]), int.Parse(Start\_time[1]), int.Parse(Start\_time[2])), Days));

StreamWriter sw = new StreamWriter(@"D:\DZ\OOP\prep1\Course\Hospital\_program/Doctors.txt", true, Encoding.Default);

sw.WriteLine(name);

sw.Close();

StreamWriter sw2 = new StreamWriter(@"D:\DZ\OOP\prep1\Course\Hospital\_program/info.txt", true, Encoding.Default);

string output = age + " " + Specialty + " " + hours + " " + new TimeSpan(int.Parse(Start\_time[0]), int.Parse(Start\_time[1]), int.Parse(Start\_time[2])) + " ";

for (int i = 0; i < input\_Days.Length; i++)

{

output += input\_Days[i] + " ";

}

sw2.WriteLine(output);

sw2.Close();

}

public static Doctor Doc(string name)

{

{

for (int i = 0; i < Doctors.Count; i++)

{

if (Doctors[i].Name == name) return Doctors[i];

}

return null;

}

}

public static void Run()

{

Console.ForegroundColor = ConsoleColor.Cyan;

Console.WriteLine("Hello, Dear User!");

Console.ResetColor();

bool next = true;

while (next)

{

Console.WriteLine("\nWhat should we do now? Enter:\n1 - Hire,\n2 - Check Appointment,\n3 - Doctors Info,\n4 - Check If Doctor Is Present,\n5 - Book An Appointment,\n6 - Doctor's Appointments,\n7 - Visit Doctor,\nq - Quit.");

string a = Console.ReadLine();

switch (a)

{

case "1":

HireDoctor();

Console.WriteLine("Successfully added the doctor!");

break;

case "2":

Console.WriteLine("\nEnter the doctor's full name.");

string doctor = Console.ReadLine();

if (Doc(doctor) == null)

{

Console.WriteLine("There is no such doctor " + doctor + "!");

break;

}

Console.WriteLine("\nEnter the date of the appointment. Example: 21.05.2020 15:00:00");

string[] dates\_int;

bool b;

do {

dates\_int = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

b = false;

while (dates\_int.Length != 6)

{

Console.WriteLine("Error! Wrong input.");

dates\_int = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

}

for (int i = 0; i < 6; i++)

{

if (!int.TryParse(dates\_int[i], out \_)){

Console.WriteLine("Error! Please enter your date as in example. ");

b = true;

break;

}

}

} while (b) ;

if (Doc(doctor).CheckAppointment(new DateTime(int.Parse(dates\_int[2]), int.Parse(dates\_int[1]), int.Parse(dates\_int[0]), int.Parse(dates\_int[3]), int.Parse(dates\_int[4]), int.Parse(dates\_int[5]))))

{

Console.WriteLine("\nYes. There is such an appointment.");

}

else { Console.WriteLine("\nNo. There is no such appointment in our data base."); }

break;

case "3":

Console.WriteLine("\nHere you go.\n");

for (int i = 0; i < Doctors.Count; i++)

{

Console.Write(Doctors[i].Specialty + " " + Doctors[i].Name + " " + Doctors[i].Schedule.StartOfWork + " - " + Doctors[i].Schedule.EndOfWork);

for (int j = 0; j < Doctors[i].Schedule.WorkDays.Length; j++)

{

Console.Write(" " + Doctors[i].Schedule.WorkDays[j]);

}

Console.WriteLine();

}

Console.WriteLine();

break;

case "4":

Console.WriteLine("Enter the doctor's full name.");

string doctor4 = Console.ReadLine();

if (Doc(doctor4) == null)

{

Console.WriteLine("There is no such doctor "+doctor4+"!");

break;

}

if (Doc(doctor4).IsFree())

{

Console.WriteLine("Yes, " + doctor4 + " is present. The patient may also see the doctor's appointments or visit the doctor.");

}

else { Console.WriteLine("No. " + doctor4 + " doesn't work today."); }

break;

case "5":

Console.WriteLine("Enter the doctor's full name.");

string doctor5 = Console.ReadLine();

if (Doc(doctor5) == null)

{

Console.WriteLine("There is no such doctor " + doctor5 + "!");

break;

}

Console.WriteLine("Enter the date of the appointment. Example: 21.05.2020 15:00::00");

string[] dates\_int5;

dates\_int5 = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

bool errors = false;

while (dates\_int5.Length != 6)

{

Console.WriteLine("Error! Wrong input.");

dates\_int5 = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

}

for (int i = 0; i < 6; i++)

{

if (!int.TryParse(dates\_int5[i], out \_))

{

Console.WriteLine("Error! Please enter your date as in example. ");

errors = true;

break;

}

}

while (errors)

{

dates\_int5= Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

errors = false;

while (dates\_int5.Length != 6)

{

Console.WriteLine("Error! Wrong input.");

dates\_int5 = Console.ReadLine().Split(new char[] { ' ', '.', ':' }, StringSplitOptions.RemoveEmptyEntries);

}

for (int i = 0; i < 6; i++)

{

if (!int.TryParse(dates\_int5[i], out \_))

{

Console.WriteLine("Error! Please enter your date as in example. ");

errors = true;

break;

}

}

}

\_ = Doc(doctor5) + new DateTime(int.Parse(dates\_int5[2]), int.Parse(dates\_int5[1]), int.Parse(dates\_int5[0]), int.Parse(dates\_int5[3]), int.Parse(dates\_int5[4]), int.Parse(dates\_int5[5]));

break;

case "6":

Console.WriteLine("Enter the doctor's full name.");

string doctor6 = Console.ReadLine();

if (Doc(doctor6) == null)

{

Console.WriteLine("There is no such doctor " + doctor6 + "!");

break;

}

if (Doc(doctor6).Appointments.Count == 0) { Console.WriteLine(doctor6 + " doesn't have any appointments yet!"); }

else

{

Console.WriteLine("Here are the doctor's appointments:");

for (int i = 0; i < Doc(doctor6).Appointments.Count; i++)

{

Console.WriteLine(Doc(doctor6).Appointments[i]);

}

}

break;

case "7":

Console.WriteLine("What are the patient's name, age and troubles?");

Console.Write("Name:");

string name = Console.ReadLine();

Console.Write("\nAge:");

string input7;

input7 = Console.ReadLine();

int age7;

bool isParsable7 = int.TryParse(input7, out age7);

while (!isParsable7)

{

Console.WriteLine("Error! Please enter a number. ");

input7 = Console.ReadLine();

isParsable7 = int.TryParse(input7, out age7);

}

Console.Write("\nTroubles:");

string troubles = Console.ReadLine();

Patient new\_patient = new Patient(name, age7, troubles);

Console.WriteLine("\nWho does the patient want to see? Enter full name of the doctor.");

string doctor7 = Console.ReadLine();

while (Doc(doctor7) == null)

{

Console.WriteLine("There is no such doctor " + doctor7 + "! Please try again.");

doctor7 = Console.ReadLine();

}

new\_patient.SeeDoctor(Doc(doctor7));

break;

case "q":

next = false;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nGoodbye. Have a nice day!");

Console.ResetColor();

break;

default:

Console.WriteLine("\nWrong key was pressed. Please try again.");

break;

}

}

}

}

}

**IContains.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Hospital\_Lib

{

interface IContains

{

bool Contains(DayOfWeek day);

}

}

**Hospital\_program.cs**

using System;

using Hospital\_Lib;

namespace Hospital\_program

{

class Program

{

static void Main(string[] args)

{

Hospital.Run();

}

}

}

# **АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ**

Я виконав поставлене завдання і створив додаток для реєстратури лікарні. Але головне те, що я зміг закріпити отримані знання, та ще й поширити їх на нові для себе теми (такі як класи DateTime, TimeSpan, та List<T>) Підводячи підсумок, ця КР поставила мені завдання, до якого я зміг підійти креативно та ще й віднайти нові можливості у мові C#.

**ВИКОРИСТАНА ЛІТЕРАТУРА**

1. <https://metanit.com/sharp/tutorial/>
2. <https://docs.microsoft.com/en-us/dotnet/csharp/>
3. Матеріали лекцій