

Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра ИИТ

Лабораторная работа №4

По дисциплине: "СПП"

Выполнил:
Студент 3 курса
Группы ПО-3
Лущ М. Г.
Проверил:
Крощенко А. А.

Брест 2019

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Вариант 14

Задание №1:

Реализовать указанный класс, включив в него вспомогательный внутренний класс или классы. Реализовать 2-3 метода (на выбор). Продемонстрировать использование реализованных классов.

Создать класс Department (отдел фирмы) с внутренним классом, с помощью объектов которого можно хранить информацию обо всех должностях отдела и обо всех сотрудниках, когда-либо занимавших конкретную должность.

Код программы:

```
using System;
```

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

```
namespace lab4_1
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            Department department = new Department();
```

```
            department.Statuses.AddRange(new List<Department.Status>()
```

```
            {
```

```
                new Department.Status() { StatusName = "Director" },
```

```
                new Department.Status() { StatusName = "Worker" },
```

```
                new Department.Status() { StatusName = "Manager" },
```

```
                new Department.Status() { StatusName = "Consulter" },
```

```
                new Department.Status() { StatusName = "Junior" },
```

```
                new Department.Status() { StatusName = "Middle" },
```

```
                new Department.Status() { StatusName = "Senior" },
```

```
                new Department.Status() { StatusName = "Tester" },
```

```
                new Department.Status() { StatusName = "DevOps" }
```

```
            });
```

```
department.Employees.AddRange(new List<Department.Employee>()
{
    new Department.Employee(),
    new Department.Employee(),
});
```

```
department.Employees[0].CurrentStatus = department.Statuses[7];
department.Employees[0].CurrentStatus = department.Statuses[4];
department.Employees[0].CurrentStatus = department.Statuses[5];
department.Employees[0].CurrentStatus = department.Statuses[6];
```

```
department.Employees[1].CurrentStatus = department.Statuses[7];
department.Employees[1].CurrentStatus = department.Statuses[2];
department.Employees[1].CurrentStatus = department.Statuses[1];
```

```
Console.ForegroundColor = ConsoleColor.Yellow;
```

```
Console.WriteLine("Список должностей:");
```

```
foreach (var status in department.Statuses)
```

```
{
    Console.WriteLine(status);
}
```

```
Console.ForegroundColor = ConsoleColor.Cyan;
```

```
Console.WriteLine("Предыдущие должности первого сотрудника:");
```

```
foreach (var status in department.Employees[0].PrevStatuses)
```

```
{
    Console.WriteLine(status);
}
```

```
Console.ForegroundColor = ConsoleColor.Magenta;
```

```
Console.WriteLine("Предыдущие должности второго сотрудника:");
```

```
foreach (var status in department.Employees[1].PrevStatuses)
```

```
{
```

```

        Console.WriteLine(status);
    }

    Console.ForegroundColor = ConsoleColor.White;
}
}

class Department
{
    public List<Employee> Employees { get; set; }
    public List<Status> Statuses { get; set; }
    public Department()
    {
        Statuses = new List<Status>();
        Employees = new List<Employee>();
    }

    internal class Employee
    {
        private List<Status> _prevStatuses;
        private Status _currentStatus;

        public string Info { get; set; }
        public IReadOnlyCollection<Status> PrevStatuses { get => _prevStatuses; }

        public Status CurrentStatus
        {
            {
                get => _currentStatus;
                set
                {
                    if (_currentStatus != null)
                        _prevStatuses.Add(_currentStatus);

                    _currentStatus = value;
                }
            }
        }
    }
}

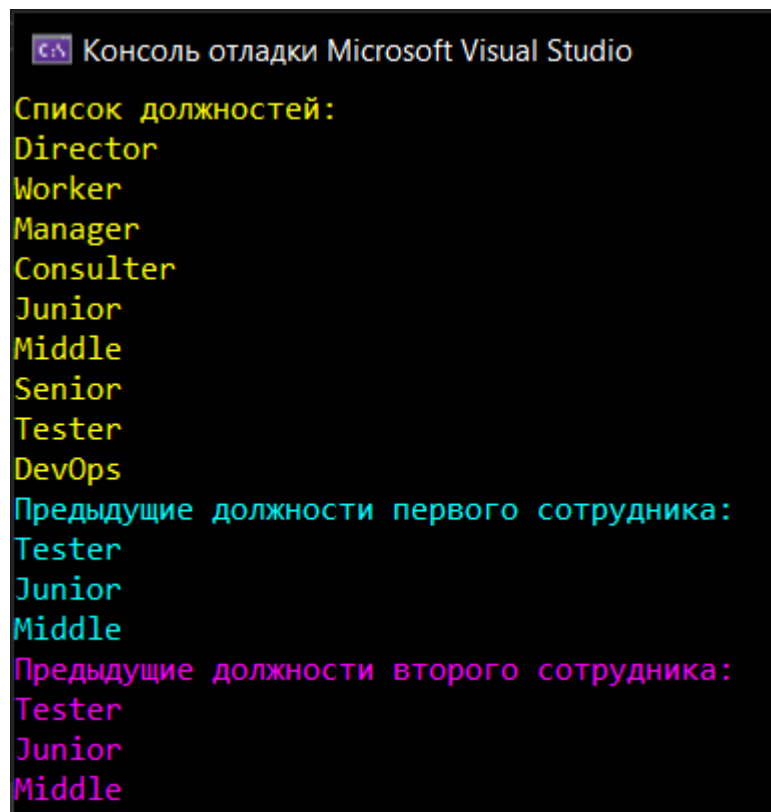
```

```

        }
    }
    public Employee()
    {
        _prevStatuses = new List<Status>();
    }
}
internal class Status
{
    public string StatusName { get; set; }
    public override string ToString()
    {
        return StatusName;
    }
}
}
}

```

Результаты работы программы:



```

Консоль отладки Microsoft Visual Studio
Список должностей:
Director
Worker
Manager
Consulter
Junior
Middle
Senior
Tester
DevOps
Предыдущие должности первого сотрудника:
Tester
Junior
Middle
Предыдущие должности второго сотрудника:
Tester
Junior
Middle

```

Задание №2:

Реализовать агрегирование. При создании класса агрегируемый класс объявляется как атрибут (локальная переменная, параметр метода). Включить в каждый класс 2-3 метода на выбор. Продемонстрировать использование разработанных классов.

Создать класс Абзац, используя класс Строка.

Код программы:

```
using System;

using System.Collections.Generic;

using System.Linq;

namespace lab4_2
{
    class Program
    {
        static void Main(string[] args)
        {
            Paragraph paragraph = new Paragraph();
            paragraph.Append("some text ");
            paragraph.AppendLine("End of line");
            paragraph.AppendLine("Second line");
            paragraph.Append("Some text");
            Console.WriteLine(paragraph.Text);
            paragraph.Clear();
            Console.WriteLine("~~~~~");
            Console.WriteLine("After clear");
            Console.WriteLine(paragraph.Text);
        }
    }

    class Paragraph
    {
        private List<String> _strings;

        public string Text
```

```
{  
    get => string.Concat(_strings.Select(x => x.Str));  
}
```

```
public void AppendLine(string newLine)  
{  
    Append(newLine + "\n");  
}
```

```
public void Append(string newLine)  
{  
    _strings.Add(new String(newLine));  
}
```

```
public void Clear()  
{  
    _strings = new List<String>();  
}
```

```
public Paragraph()  
{  
    _strings = new List<String>();  
}
```

```
class String  
{  
    public string Str { get; set; }  
    public String(string str)  
    {  
        Str = str;  
    }  
    public override string ToString()  
    {  
        return Str;  
    }  
}
```

$$\left. \begin{array}{l} \} \\ \} \\ \} \\ \} \end{array} \right\}$$

Результаты работы программы:

```

some text End of line
Second line
Some text
~~~~~
After clear

E:\NetCoreConsoleApp\lab4_2\bin\Debug\netcoreapp3.1\lab4_2.exe (процесс 17648) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Задание №3:

Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы.

Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или иных обстоятельствах.

Код программы:

Файл Hospital.cs:

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

namespace lab4_3
{
    class Hospital
    {
        private List<Doctor> _doctors;
        private List<HospitalWorker> _nurses;

        private Queue<IAppointable> _patients;
        private Queue<IAppointable> _appointPatients;

        public IReadOnlyCollection<Doctor> Doctors { get => _doctors; }
        public IReadOnlyCollection<HospitalWorker> Nurces { get => _nurses; }
        public IReadOnlyCollection<IAppointable> Patients { get => _patients.ToList(); }
        public IReadOnlyCollection<IAppointable> AppointPatients { get => _appointPatients.ToList(); }

        public Hospital()
        {
            _doctors = new List<Doctor>();
            _nurses = new List<HospitalWorker>();
            _patients = new Queue<IAppointable>();
            _appointPatients = new Queue<IAppointable>();
        }
    }
}
```



```

{
    _doctors = new List<Doctor>();
    _nurses = new List<HospitalWorker>();

    _patients = new Queue<IAppointable>();
    _appointPatients = new Queue<IAppointable>();
}

public Hospital(IEnumerable<Doctor> doctors, IEnumerable<Nurse> nurses) : this()
{
    _doctors.AddRange(doctors);
    _nurses.AddRange(nurses);
}

public void AddPatient(IAppointable patient)
{
    _patients.Enqueue(patient);
}

public void AddDoctor(Doctor doctor)
{
    _doctors.Add(doctor);
}

public void AddNurse(Nurse nurse)
{
    _nurses.Add(nurse);
}

public void AppointPatient(Appointment appointment)
{
    if (_patients.Count > 0)
    {
        IAppointable first = _patients.Dequeue();
        _doctors.Find(x => !x.IsBusy).MakeAppointment(first, appointment);
        _appointPatients.Enqueue(first);
        Console.WriteLine("");
    }
    else
        throw new Exception("В больнице отсутствуют пациенты");
}

public void TreatPatient()
{
    if (_appointPatients.Count > 0)
    {
        IAppointable first = _appointPatients.Dequeue();
        HospitalWorker worker = _nurses.Find(x => !x.IsBusy) ?? _doctors.Find(x => !x.IsBusy) as
HospitalWorker;

        if (worker == null)
            throw new Exception("В больнице отсутствуют свободные врачи и медсёстры");
    }
}

```

```

        else
        {
            worker.ExecuteAppointment(first);
            if (first.IsIll)
            {
                _patients.Enqueue(first);
                Console.WriteLine("Пациент не выздоровел и остаётся в больнице");
            }
            else
            {
                Console.WriteLine("Пациент выздоровел и выписан из больницы");
            }
        }
    }
    else
        throw new Exception("В больнице отсутствуют пациенты");
}

public void TreatPatient(IAppointable patient, string reason)
{
    var patients = _patients.ToList();
    var appointPatients = _appointPatients.ToList();

    patients.Remove(patient);
    appointPatients.Remove(patient);

    _patients = new Queue<IAppointable>(patients);
    _appointPatients = new Queue<IAppointable>(appointPatients);
}

public void DisposeDoctor(int index)
{
    if (index < 0 || index >= Doctors.Count)
        throw new ArgumentOutOfRangeException();
    else
        _doctors[index].Dispose();
}

public void DisposeNurce(int index)
{
    if (index < 0 || index >= Doctors.Count)
        throw new ArgumentOutOfRangeException();
    else
        _nurses[index].Dispose();
}
}
}

```

Файл HospitalFactory.cs:

```

using System;
using System.Collections.Generic;

```

```

using System.Runtime.InteropServices;
using System.Text;

namespace lab4_3
{
    static class HospitalFactory
    {
        public static Hospital CreateHospital(int numOfDoctors, int numOfNurses)
        {
            Hospital result = new Hospital();
            for (int i = 0; i < numOfDoctors; i++)
            {
                result.AddDoctor(new Doctor());
            }
            for (int i = 0; i < numOfNurses; i++)
            {
                result.AddNurse(new Nurse());
            }
            return result;
        }
    }
}

```

Файл HospitalWorkers.cs:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace lab4_3
{
    abstract class HospitalWorker : IDisposable
    {
        protected bool _isBusy = false;
        public bool IsBusy { get => _isBusy; }

        public virtual void ExecuteAppointment(IAppointable patient)
        {
            _isBusy = true;
            patient.Treat();
        }

        public void Dispose() => _isBusy = false;
    }

    sealed class Doctor : HospitalWorker
    {
        public void MakeAppointment(IAppointable patient, Appointment appointment)
        {
            _isBusy = true;
            patient.GiveAppointment(appointment);
        }
    }
}

```

```

        public override void ExecuteAppointment(IAppointable patient)
        {
            base.ExecuteAppointment(patient);
            Console.WriteLine("Врач выполнил назначение");
        }
    }
}
sealed class Nurse : HospitalWorker
{
    public override void ExecuteAppointment(IAppointable patient)
    {
        base.ExecuteAppointment(patient);
        Console.WriteLine("Медсестра выполнила назначение");
    }
}
}

```

Файл Patient.cs:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace lab4_3
{
    interface IAppointable
    {
        void Treat();
        public void GiveAppointment(Appointment appointment);

        public bool IsIll { get; set; }
    }
    sealed class Patient: IAppointable
    {
        public readonly string fullname;
        private Appointment? _appointment;
        private bool _isIll;

        public bool IsIll
        {
            get => _isIll;
            set
            {
                if (_isIll == false && value == true)
                {
                    Console.WriteLine("Больница пытается сделать здорового пациента больным");
                    return;
                }

                _isIll = value;
            }
        }
    }
}

```

```

public Patient(string fullname)
{
    this.fullname = fullname;
    _isIll = true;
}

public void Treat()
{
    if (_appointment.HasValue && IsIll)
    {
        Console.WriteLine("Пациента лечат");
        IsIll = _appointment.Value.ExecuteAppointment(fullname);
        _appointment = null;
    }
    else
        throw new Exception($"У пациента {fullname} отсутствует направление.");
}

public void GiveAppointment(Appointment appointment)
{
    Console.WriteLine($"Пациенту {fullname} выдали направление {appointment}");
    _appointment = appointment;
}

}
}

```

Файл Program.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Security.Cryptography;

namespace lab4_3
{
    class Program
    {
        static void Main(string[] args)
        {
            Hospital hospital = HospitalFactory.CreateHospital(numOfDoctors: 4, numOfNurses: 2);
            Patient[] patients = new Patient[]
            {
                new Patient("Иван Петрович"),
                new Patient("Михаил Викотрович"),
                new Patient("Мария Викторовна"),
                new Patient("Владислав Валентинович"),
                new Patient("Александр Александрович"),
                new Patient("Анастасия Андреевна"),
                new Patient("Мария Андреевна")
            };
        }
    }
}

```

```

        for (int i = 0; i < patients.Length; i++)
        {
            hospital.AddPatient(patients[i]);
        }

        hospital.AppointPatient(Appointment.Medications);
        hospital.TreatPatient();

        hospital.TreatPatient(patients[2], "нарушение режима");

    }
}

```

Файл Utils.cs:

```

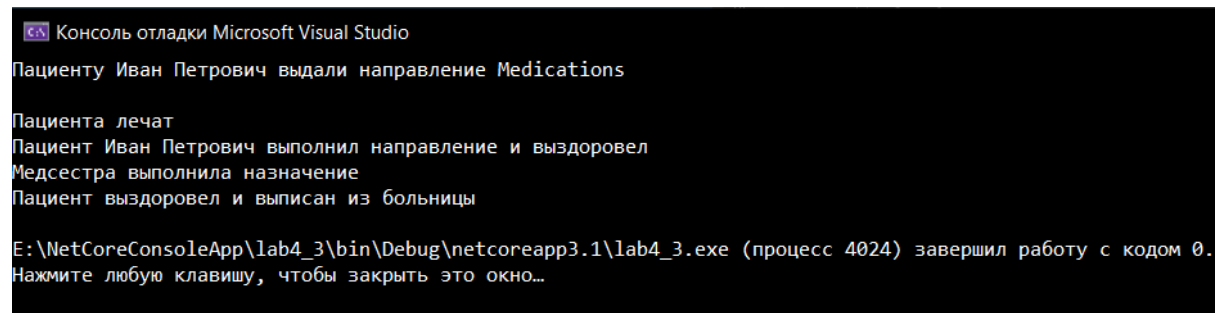
using System;
using System.Collections.Generic;
using System.Text;

namespace lab4_3
{
    public enum Appointment { Procedures, Medications, Operations }
    public static class AppointmentExtension
    {
        public static bool ExecuteAppointment(this Appointment appointment, string patientFullname,
        float healChance = 0.5f)
        {
            Random random = new Random();
            bool isIll = random.NextDouble() < healChance;

            Console.WriteLine("Пациент {0} выполнил направление и {1}", patientFullname, isIll ? "не
            выздоровел" : "выздоровел");
            return isIll;
        }
    }
}

```

Результаты работы программы:



Консоль отладки Microsoft Visual Studio

```

Пациенту Иван Петрович выдали направление Medications

Пациента лечат
Пациент Иван Петрович выполнил направление и выздоровел
Медсестра выполнила назначение
Пациент выздоровел и выписан из больницы

E:\NetCoreConsoleApp\lab4_3\bin\Debug\netcoreapp3.1\lab4_3.exe (процесс 4024) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...

```