**Отчёт**

**Практическая работа №9 Разработка классов в C#**

**Цель: научиться разрабатывать классы, используя конструкторы, свойства, операторы и разные модификаторы доступа**

ПОРЯДОК ВЫПОЛНЕНИЯ

* Создать новое решение с проектом консольного приложения на основании языка C#.
* Создайте класс и объект из «пример создания класса»
* Показать преподавателю выполненное задание.
* Выполнить задания из блока Практические задания создавая для каждого класса новый файл .cs (класс консольного приложения) консольного приложения C#.
* После выполнения каждого задания показать результат преподавателю.

ПРИМЕР СОЗДАНИЯ КЛАССА



Для класса **Hospital** добавить поля:

* hospitalId (int, только для чтения): Уникальный идентификатор больницы.
* name (string): Название больницы.
* address (string): Адрес больницы.
* departments (List): Список отделений больницы (содержит id отделений).

Для класса **Hospital** добавить конструктор с 1 параметром (hospitalId) и его перегрузки с большим количеством параметров. Вторая перегрузка должна вызывать прошлую через this. Все последующие также.

Сделать все поля приватными и создать для них свойства, обрабатывающие эти поля.

1a **Создание класса Hospital с полями и свойствами**

using System;

using System.Collections.Generic;

public class Hospital

{

private readonly int hospitalId;

private string name;

private string address;

private List<int> departments;

// Свойства

public int HospitalId => hospitalId;

public string Name { get => name; set => name = value; }

public string Address { get => address; set => address = value; }

public List<int> Departments { get => departments; set => departments = value; }

// Конструктор с 1 параметром

public Hospital(int hospitalId)

{

this.hospitalId = hospitalId;

this.departments = new List<int>();

}

}

public class Program

{

public static void Main()

{

Hospital hospital = new Hospital(1);

hospital.Name = "Городская больница";

hospital.Address = "ул. Ленина, 10";

hospital.Departments.Add(101);

Console.WriteLine($"ID: {hospital.HospitalId}, Название: {hospital.Name}, Адрес: {hospital.Address}");

Console.WriteLine("Отделения: " + string.Join(", ", hospital.Departments));

}

}

1b. **Добавление перегруженных конструкторов**

using System;

using System.Collections.Generic;

public class Hospital

{

private readonly int hospitalId;

private string name;

private string address;

private List<int> departments;

// Свойства (аналогично 1a)

public int HospitalId => hospitalId;

public string Name { get => name; set => name = value; }

public string Address { get => address; set => address = value; }

public List<int> Departments { get => departments; set => departments = value; }

// Конструкторы с перегрузкой через this

public Hospital(int hospitalId) : this(hospitalId, "Неизвестно", "Неизвестно", new List<int>()) { }

public Hospital(int hospitalId, string name) : this(hospitalId, name, "Неизвестно", new List<int>()) { }

public Hospital(int hospitalId, string name, string address) : this(hospitalId, name, address, new List<int>()) { }

public Hospital(int hospitalId, string name, string address, List<int> departments)

{

this.hospitalId = hospitalId;

this.name = name;

this.address = address;

this.departments = departments;

}

}

public class Program

{

public static void Main()

{

// Использование разных конструкторов

Hospital hospital1 = new Hospital(1);

Hospital hospital2 = new Hospital(2, "Центральная больница");

Hospital hospital3 = new Hospital(3, "Детская больница", "ул. Пушкина, 5");

Hospital hospital4 = new Hospital(4, "Районная больница", "ул. Гагарина, 3", new List<int> { 401, 402 });

Console.WriteLine($"Больница 3: {hospital3.Name}, Адрес: {hospital3.Address}");

}

}

1c. **Проверка свойств с валидацией (дополнение к 1a)**

using System;

using System.Collections.Generic;

public class Hospital

{

private readonly int hospitalId;

private string name;

private string address;

private List<int> departments;

// Свойства с проверками

public int HospitalId => hospitalId;

public string Name

{

get => name;

set => name = !string.IsNullOrWhiteSpace(value) ? value : throw new ArgumentException("Название не может быть пустым");

}

public string Address

{

get => address;

set => address = !string.IsNullOrWhiteSpace(value) ? value : "Адрес не указан";

}

public List<int> Departments

{

get => departments;

set => departments = value ?? new List<int>();

}

// Конструктор из 1a

public Hospital(int hospitalId)

{

this.hospitalId = hospitalId;

this.departments = new List<int>();

}

}

public class Program

{

public static void Main()

{

Hospital hospital = new Hospital(1);

hospital.Name = " "; // Выбросит исключение

hospital.Address = null; // Установит "Адрес не указан"

Console.WriteLine(hospital.Address);

}

}

Для класса **Cardiology** добавить поля:

* departmentId (int, только для чтения): Уникальный идентификатор отделения.
* name (string): Название отделения.
* headOfDepartment (Doctor): Заведующий отделением.
* doctors (List): Список врачей.
* beds (bool[]): Массив коек (True — свободна, False — занята).

Добавить конструктор без параметров и его перегрузки, вызывающие предыдущие через this.

Реализовать методы:

* AddDoctor, RemoveDoctor, FindDoctor, GetAvailableBeds.

#### **2a: Создание класса Cardiology с полями, свойствами и конструкторами**

using System;

using System.Collections.Generic;

public class Doctor

{

public int DoctorId { get; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Specialization { get; set; }

public Doctor(int doctorId, string firstName, string lastName, string specialization)

{

DoctorId = doctorId;

FirstName = firstName;

LastName = lastName;

Specialization = specialization;

}

}

public class Cardiology

{

private readonly int departmentId;

private string name;

private Doctor headOfDepartment;

private List<Doctor> doctors;

private bool[] beds;

// Свойства

public int DepartmentId => departmentId;

public string Name { get => name; set => name = value; }

public Doctor HeadOfDepartment { get => headOfDepartment; set => headOfDepartment = value; }

public List<Doctor> Doctors { get => doctors; set => doctors = value; }

public bool[] Beds { get => beds; set => beds = value; }

// Конструкторы

public Cardiology()

{

this.doctors = new List<Doctor>();

this.beds = new bool[10]; // Пример: 10 коек, изначально все свободны (true)

Array.Fill(this.beds, true);

}

public Cardiology(int departmentId) : this()

{

this.departmentId = departmentId;

}

public Cardiology(int departmentId, string name) : this(departmentId)

{

this.name = name;

}

public Cardiology(int departmentId, string name, Doctor head) : this(departmentId, name)

{

this.headOfDepartment = head;

}

}

public class Program

{

public static void Main()

{

Doctor headDoctor = new Doctor(1, "Иван", "Петров", "Кардиолог");

Cardiology cardiology = new Cardiology(101, "Кардиологическое отделение", headDoctor);

Console.WriteLine($"Отделение: {cardiology.Name}, Заведующий: {cardiology.HeadOfDepartment.LastName}");

Console.WriteLine($"Свободных коек: {cardiology.Beds.Length}");

}

}

#### **2b: Реализация методов для работы с врачами и койками**

using System;

using System.Collections.Generic;

using System.Linq;

public class Doctor

{

public int DoctorId { get; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Specialization { get; set; }

public Doctor(int doctorId, string firstName, string lastName, string specialization)

{

DoctorId = doctorId;

FirstName = firstName;

LastName = lastName;

Specialization = specialization;

}

// Добавлен метод ToString для удобного вывода информации о враче

public override string ToString()

{

return $"{LastName} {FirstName} ({Specialization})";

}

}

public class Cardiology

{

private readonly int departmentId;

private string name;

private Doctor headOfDepartment;

private List<Doctor> doctors;

private bool[] beds;

// Свойства

public int DepartmentId => departmentId;

public string Name { get => name; set => name = value; }

public Doctor HeadOfDepartment

{

get => headOfDepartment;

set => headOfDepartment = value ?? throw new ArgumentNullException("Заведующий не может быть null");

}

public List<Doctor> Doctors { get => doctors; set => doctors = value ?? new List<Doctor>(); }

public bool[] Beds { get => beds; set => beds = value ?? new bool[0]; }

// Конструкторы

public Cardiology()

{

this.doctors = new List<Doctor>();

this.beds = new bool[10];

Array.Fill(this.beds, true);

}

public Cardiology(int departmentId) : this()

{

this.departmentId = departmentId;

}

public Cardiology(int departmentId, string name) : this(departmentId)

{

this.name = name;

}

public Cardiology(int departmentId, string name, Doctor head) : this(departmentId, name)

{

this.HeadOfDepartment = head; // Используем свойство для проверки null

}

// Методы

public void AddDoctor(Doctor doctor)

{

if (doctor == null)

throw new ArgumentNullException(nameof(doctor), "Врач не может быть null");

// Проверка на дублирование ID

if (doctors.Any(d => d.DoctorId == doctor.DoctorId))

throw new ArgumentException($"Врач с ID {doctor.DoctorId} уже существует");

doctors.Add(doctor);

}

public bool RemoveDoctor(int doctorId)

{

var doctor = doctors.FirstOrDefault(d => d.DoctorId == doctorId);

if (doctor != null)

{

return doctors.Remove(doctor);

}

return false;

}

public Doctor FindDoctor(int doctorId)

{

return doctors.FirstOrDefault(d => d.DoctorId == doctorId);

}

public int GetAvailableBeds()

{

return beds.Count(b => b);

}

}

public class Program

{

public static void Main()

{

try

{

Cardiology cardiology = new Cardiology(101, "Кардиология");

// Добавляем врачей

cardiology.AddDoctor(new Doctor(1, "Анна", "Сидорова", "Кардиолог"));

cardiology.AddDoctor(new Doctor(2, "Сергей", "Иванов", "Кардиохирург"));

// Пытаемся добавить врача с существующим ID (вызовет исключение)

// cardiology.AddDoctor(new Doctor(2, "Иван", "Петров", "Терапевт"));

// Тестируем методы

Console.WriteLine($"Найден врач: {cardiology.FindDoctor(1)?.ToString() ?? "Не найден"}");

Console.WriteLine($"Свободных коек: {cardiology.GetAvailableBeds()}");

// Удаление врача

bool removed = cardiology.RemoveDoctor(2);

Console.WriteLine($"Врач удалён: {removed}");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

}

Создать классы **Surgery** и **Therapeutic**, аналогичные классу **Cardiology**, с соответствующими полями и методами.

using System;

using System.Collections.Generic;

// Базовый класс для отделения (чтобы избежать дублирования кода)

public class MedicalDepartment

{

public string DepartmentName { get; set; }

public List<string> Patients { get; set; }

public MedicalDepartment(string name)

{

DepartmentName = name;

Patients = new List<string>();

}

public void AddPatient(string patientName)

{

Patients.Add(patientName);

}

public virtual void DisplayInfo()

{

Console.WriteLine($"Отделение: {DepartmentName}");

if (Patients.Count > 0)

{

Console.WriteLine("Пациенты:");

foreach (string patient in Patients)

{

Console.WriteLine($"- {patient}");

}

}

else

{

Console.WriteLine("В отделении нет пациентов.");

}

}

}

// Класс Cardiology (Кардиология) - для примера

public class Cardiology : MedicalDepartment

{

public Cardiology() : base("Кардиология") { }

// Дополнительные поля и методы, специфичные для кардиологии

public string DoctorName { get; set; }

public Cardiology(string doctorName) : base("Кардиология") {

DoctorName = doctorName;

}

public override void DisplayInfo()

{

base.DisplayInfo();

Console.WriteLine($"Имя врача: {DoctorName}");

}

}

// Класс Surgery (Хирургия)

public class Surgery : MedicalDepartment

{

public string SurgeonName { get; set; } // Имя хирурга

public string Specialty { get; set; } // Специализация (например, общая хирургия, нейрохирургия)

public Surgery(string surgeonName, string specialty) : base("Хирургия")

{

SurgeonName = surgeonName;

Specialty = specialty;

}

public override void DisplayInfo()

{

base.DisplayInfo();

Console.WriteLine($"Хирург: {SurgeonName}");

Console.WriteLine($"Специализация: {Specialty}");

}

}

// Класс Therapeutic (Терапия)

public class Therapeutic : MedicalDepartment

{

public string HeadNurse { get; set; } // Имя старшей медсестры

public int BedCount { get; set; } // Количество коек в отделении

public Therapeutic(string headNurse, int bedCount) : base("Терапия")

{

HeadNurse = headNurse;

BedCount = bedCount;

}

public override void DisplayInfo()

{

base.DisplayInfo();

Console.WriteLine($"Старшая медсестра: {HeadNurse}");

Console.WriteLine($"Количество коек: {BedCount}");

}

}

public class Program

{

public static void Main(string[] args)

{

// Пример использования

Cardiology cardio = new Cardiology("Иванов И.И.");

cardio.AddPatient("Петров П.П.");

cardio.DisplayInfo();

Console.WriteLine();

Surgery surgery = new Surgery("Сидоров А.А.", "Общая хирургия");

surgery.AddPatient("Смирнов В.В.");

surgery.AddPatient("Козлов И.П.");

surgery.DisplayInfo();

Console.WriteLine();

Therapeutic therapeutic = new Therapeutic("Елена Сергеевна", 30);

therapeutic.AddPatient("Новиков Р.Д.");

therapeutic.DisplayInfo();

}

}

Для класса **Doctor** добавить поля:

* doctorId (int, только для чтения).
* firstName, lastName, specialization (string).
* patients (List).

Конструкторы с перегрузкой через this.

Методы:

* AddPatient, RemovePatient, FindPatient, GetPatientCount.

Свойства с проверкой специализации (например, хирург не может быть в кардиологии).

**1a. Создание класса Doctor**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**public class Doctor**

**{**

**public int DoctorId { get; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**private string \_specialization;**

**public string Specialization**

**{**

**get { return \_specialization; }**

**set { \_specialization = value; } // Упрощенная версия, пока без проверки**

**}**

**public List<string> Patients { get; set; }**

**// Конструкторы**

**public Doctor(int doctorId, string firstName, string lastName, string specialization)**

**{**

**DoctorId = doctorId;**

**FirstName = firstName;**

**LastName = lastName;**

**Specialization = specialization;**

**Patients = new List<string>();**

**}**

**public Doctor(int doctorId, string firstName, string lastName) : this(doctorId, firstName, lastName, "Не указана") { }**

**public Doctor(int doctorId, string firstName) : this(doctorId, firstName, "Не указана") { }**

**// Методы**

**public void AddPatient(string patientName)**

**{**

**Patients.Add(patientName);**

**}**

**public void RemovePatient(string patientName)**

**{**

**Patients.Remove(patientName);**

**}**

**public string FindPatient(string patientName)**

**{**

**return Patients.FirstOrDefault(p => p == patientName);**

**}**

**public int GetPatientCount()**

**{**

**return Patients.Count;**

**}**

**public virtual void DisplayInfo()**

**{**

**Console.WriteLine($"Доктор (ID: {DoctorId}): {FirstName} {LastName}, Специализация: {Specialization}");**

**Console.WriteLine($"Количество пациентов: {GetPatientCount()}");**

**}**

**}**

**// Пример использования (проверка класса Doctor)**

**public class Program**

**{**

**public static void Main(string[] args)**

**{**

**Doctor doc1 = new Doctor(1, "Иван", "Иванов", "Кардиолог");**

**doc1.AddPatient("Петров П.П.");**

**doc1.AddPatient("Сидоров А.А.");**

**doc1.DisplayInfo();**

**string foundPatient = doc1.FindPatient("Петров П.П.");**

**Console.WriteLine($"Найден пациент: {foundPatient}");**

**}**

**}**

**1b. Добавление проверки специализации в класс Doctor**

Using System;

Using System.Collections.Generic;

Using System.Linq;

Public Class Doctor

{

Public Int DoctorId { Get; }

Public String FirstName { Get; Set; }

Public String LastName { Get; Set; }

Private String \_specialization;

Public String Specialization

{

Get { Return \_specialization; }

Set

{

// Проверка Специализации

If (value.ToLower() == "кардиология" && This Is SurgeryDoctor)

{

Throw New ArgumentException("Хирург Не Может Быть В Кардиологии!");

}

\_specialization = Value;

}

}

Public List<string> Patients { Get; Set; }

// Конструкторы И Методы (как В Задании 1)

Public Doctor(int DoctorId, String FirstName, String LastName, String Specialization)

{

DoctorId = DoctorId;

FirstName = FirstName;

LastName = LastName;

Specialization = Specialization;

Patients = New List<string>();

}

Public Doctor(int DoctorId, String FirstName, String LastName) : This(doctorId, FirstName, LastName, "Не Указана") { }

Public Doctor(int DoctorId, String FirstName) : This(doctorId, FirstName, "Не Указана") { }

// Методы

Public Void AddPatient(string PatientName)

{

Patients.Add(patientName);

}

Public Void RemovePatient(string PatientName)

{

Patients.Remove(patientName);

}

Public String FindPatient(string PatientName)

{

Return Patients.FirstOrDefault(p => P == PatientName);

}

Public Int GetPatientCount()

{

Return Patients.Count;

}

Public Virtual Void DisplayInfo()

{

Console.WriteLine($"Доктор (ID: {DoctorId}): {FirstName} {LastName}, Специализация: {Specialization}");

Console.WriteLine($"Количество Пациентов: {GetPatientCount()}");

}

}

// Для Теста Исключения

Public Class SurgeryDoctor : Doctor

{

Public SurgeryDoctor(int DoctorId, String FirstName, String LastName, String Specialization) : Base (doctorId, FirstName,lastName,specialization){

}

}

// Пример Использования (проверка Исключения)

Public Class Program

{

Public Static Void Main(string[] Args)

{

Try

{

SurgeryDoctor SurgeryDoc2 = New SurgeryDoctor(3, "Борис", "Кузнецов", "Кардиология"); // Вызовет Исключение

}

Catch (ArgumentException E)

{

Console.WriteLine($"Ошибка: {e.Message}");

}

}

}

Для класса **Patient** добавить поля:

* patientId (int, только для чтения).
* firstName, lastName, dateOfBirth, diagnosis (string).

Конструкторы с перегрузкой через this.

Методы:

* UpdateDiagnosis, GetAge, ToString.

1a. Создание класса Patient

using System;

public class Patient

{

public int PatientId { get; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string DateOfBirth { get; set; }

public string Diagnosis { get; set; }

// Конструкторы

public Patient(int patientId, string firstName, string lastName, string dateOfBirth, string diagnosis)

{

PatientId = patientId;

FirstName = firstName;

LastName = lastName;

DateOfBirth = dateOfBirth;

Diagnosis = diagnosis;

}

public Patient(int patientId, string firstName, string lastName, string dateOfBirth) : this(patientId, firstName, lastName, dateOfBirth, "Неизвестен") { }

public Patient(int patientId, string firstName, string lastName) : this(patientId, firstName, lastName, "01.01.1900", "Неизвестен") { }

// Методы

public void UpdateDiagnosis(string newDiagnosis)

{

Diagnosis = newDiagnosis;

}

public int GetAge()

{

if (DateTime.TryParse(DateOfBirth, out DateTime birthDate))

{

DateTime now = DateTime.Today;

int age = now.Year - birthDate.Year;

if (birthDate.Date > now.AddYears(-age))

{

age--;

}

return age;

}

else

{

return -1; // Некорректная дата рождения

}

}

public override string ToString()

{

return $"PatientId: {PatientId}, Имя: {FirstName} {LastName}, Дата рождения: {DateOfBirth}, Диагноз: {Diagnosis}, Возраст: {GetAge()} лет";

}

public virtual void DisplayInfo()

{

Console.WriteLine(ToString());

}

}

// Пример использования (проверка класса Patient)

public class Program

{

public static void Main(string[] args)

{

Patient patient1 = new Patient(1, "Иван", "Иванов", "10.05.1980", "Грипп");

Console.WriteLine(patient1);

}

}

**1b. Добавление методов UpdateDiagnosis, GetAge, ToString в Patient**

using System;

public class Patient

{

public int PatientId { get; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string DateOfBirth { get; set; }

public string Diagnosis { get; set; }

// Конструкторы

public Patient(int patientId, string firstName, string lastName, string dateOfBirth, string diagnosis)

{

PatientId = patientId;

FirstName = firstName;

LastName = lastName;

DateOfBirth = dateOfBirth;

Diagnosis = diagnosis;

}

public Patient(int patientId, string firstName, string lastName, string dateOfBirth) : this(patientId, firstName, lastName, dateOfBirth, "Неизвестен") { }

public Patient(int patientId, string firstName, string lastName) : this(patientId, firstName, lastName, "01.01.1900", "Неизвестен") { }

// Методы

public void UpdateDiagnosis(string newDiagnosis)

{

Diagnosis = newDiagnosis;

}

public int GetAge()

{

if (DateTime.TryParse(DateOfBirth, out DateTime birthDate))

{

DateTime now = DateTime.Today;

int age = now.Year - birthDate.Year;

if (birthDate.Date > now.AddYears(-age))

{

age--;

}

return age;

}

else

{

return -1; // Некорректная дата рождения

}

}

public override string ToString()

{

return $"PatientId: {PatientId}, Имя: {FirstName} {LastName}, Дата рождения: {DateOfBirth}, Диагноз: {Diagnosis}, Возраст: {GetAge()} лет";

}

public virtual void DisplayInfo()

{

Console.WriteLine(ToString());

}

}

// Пример использования (проверка методов)

public class Program

{

public static void Main(string[] args)

{

Patient patient1 = new Patient(1, "Иван", "Иванов", "10.05.1980", "Грипп");

Console.WriteLine(patient1);

patient1.UpdateDiagnosis("Простуда");

Console.WriteLine(patient1);

Console.WriteLine($"Возраст patient1: {patient1.GetAge()}");

Patient patient2 = new Patient(2, "Петр", "Петров", "20.08.1990");

Console.WriteLine(patient2);

}

}

1c. **Создание классов MedicalDepartment, Cardiology, Surgery, Therapeutic**

using System;

using System.Collections.Generic;

using System.Linq;

public class Doctor { /\* Код класса Doctor из задания 2 \*/ }

// Модифицируем базовый класс MedicalDepartment

public class MedicalDepartment

{

public string DepartmentName { get; set; }

public List<Doctor> Doctors { get; set; }

public MedicalDepartment(string name)

{

DepartmentName = name;

Doctors = new List<Doctor>();

}

public void AddDoctor(Doctor doctor)

{

Doctors.Add(doctor);

}

public virtual void DisplayInfo()

{

Console.WriteLine($"Отделение: {DepartmentName}");

if (Doctors.Count > 0)

{

Console.WriteLine("Доктора:");

foreach (Doctor doctor in Doctors)

{

doctor.DisplayInfo();

}

}

else

{

Console.WriteLine("В отделении нет докторов.");

}

}

}

// Модифицируем классы отделений для использования Doctor

public class Cardiology : MedicalDepartment

{

public Cardiology() : base("Кардиология") { }

}

public class Surgery : MedicalDepartment

{

public Surgery() : base("Хирургия") { }

}

public class Therapeutic : MedicalDepartment

{

public Therapeutic() : base("Терапия") { }

}

// Пример использования

public class Program

{

public static void Main(string[] args)

{

Cardiology cardio = new Cardiology();

Doctor cardioDoc1 = new Doctor(1, "Иван", "Иванов", "Кардиолог");

cardio.AddDoctor(cardioDoc1);

cardioDoc1.AddPatient("Петров П.П.");

cardio.DisplayInfo();

Console.WriteLine();

}

}