МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева»**

Филиал СибГУ в г. Лесосибирске

Кафедра информационных и технических систем

КУРСОВАЯ РАБОТА

Программирование
наименование дисциплины

Разработка приложения с графическим интерфейсом
«Деканат»
тема работы

Руководитель — оценка: удовлетворительно; подпись, дата: 23.06.25; П.А. Егармин (инициалы, фамилия)

Обучающийся — БИТ23-11 № 232920900 26 (номер группы, зачетной книжки); подпись, дата: 23.06.25; С.А. Сейденцаль (инициалы, фамилия)

Лесосибирск 2025

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

1. Ознакомиться с текстом задания. Выполнить объектно-ориентированный анализ и проектирование программной системы.
2. Выполнить реализацию объектной модели средствами объектно-ориентированного программирования.
3. Разработать и реализовать графический интерфейс для программной системы.
4. Выполнить отладку программы и получить выходные результаты.
5. Оформить отчет в виде пояснительной записки по курсовой работе.

Постановка задачи. В отделе кадров вуза хранится следующая информация о педагогических работниках:

- фамилия;
- имя;
- должность;
- дата рождения;
- дата принятия на работу.

Ежегодно вузом формируется ведомость о поощрении работников по итогам учебного года, о которой известна следующая информация:

- год формирования ведомости;
- количество работников вуза.

Необходимо:

1. Построить модель предметной области.
2. Построить диаграмму классов проектирования.
3. Разработать приложение в соответствии с диаграммой классов проектирования, реализующее следующие задачи:
   - определить стаж каждого работника;
   - определить возраст каждого работника;
   - вывести на экран сведения о работнике с минимальным и максимальным стажем;
   - вывести на экран список работников на поощрение (стаж более десяти лет);
   - вывести на экран список всех работников вуза, отсортированных по возрасту.

В программе предусмотреть:

- использование свойств, индексаторов, коллекций, обработку исключительных ситуаций;
- организацию диалога с пользователем во время работы программы;
- сохранение данных в файл, загрузку данных их файла.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

C# (*си шарп*) – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров компании *Microsoft* под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы *Microsoft .NET Framework* [3].

.NET Framework – программная платформа, выпущенная компанией *Microsoft* в 2002 году. Основой платформы является общеязыковая среда исполнения *Common Language Runtime (CLR)*, которая подходит для разных языков программирования. Функциональные возможности *CLR* доступны в любых языках программирования, использующих эту среду.

Считается, что платформа *.NET Framework* явилась ответом компании *Microsoft* на набравшую к тому времени большую популярность платформу Java компании *Sun Microsystems* (ныне принадлежит *Oracle*).

Хотя *.NET* является патентованной технологией корпорации *Microsoft* и официально рассчитана на работу под операционными системами семейства *Microsoft Windows*, существуют независимые проекты (например, *Mono* и *Portable.NET*), позволяющие запускать программы *.NET* на некоторых других операционных системах.

Цель выполнения курсовой работы: проектирование и реализация приложения с графическим интерфейсом «Деканат» на языке C#.

Задачи курсовой работы:

- изучение литературы по методологии объектно-ориентированного программирования и проектирования;
- построение объектной модели предметной области;
- реализация объектной модели средствами языка C#;
- разработка графического интерфейса программной системы;
- тестирование и отладка приложения.

Вариант выполнения курсовой работы – 8.

1 ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ

Поскольку формулировка задач, решаемых с помощью компьютера, все ближе приближается к формулировкам реальных жизненных процессов, необходимо представить программу в виде множества объектов, каждый из которых обладает своими свойствами и поведением, но его внутреннее устройство скрыто от других объектов. Тогда решение задачи сводится к моделированию взаимодействия этих объектов. Построенная таким образом модель задачи называется **объектной**.

Для того, чтобы построить объектную модель, необходимо:

– выделить взаимодействующие объекты, с помощью которых можно достаточно полно описать поведение моделируемой системы;

– определить свойства объектов, существенных в данной задаче;

– описать поведение (возможные действия) объектов, то есть команды, которые объекты могут выполнять.

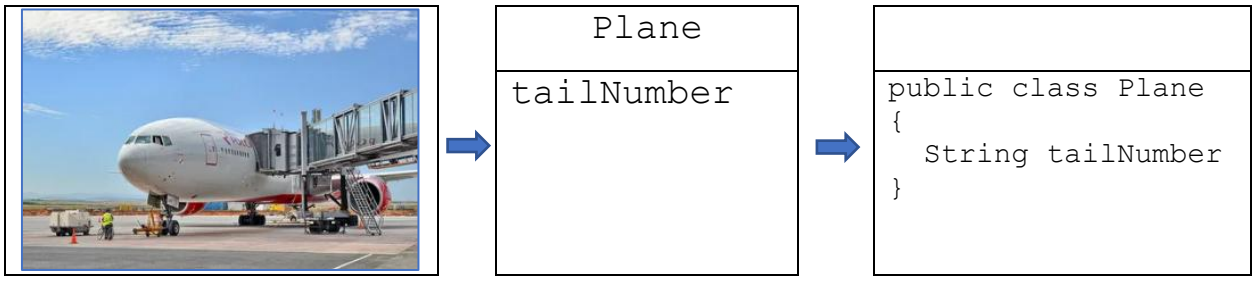
Этап разработки модели, на котором решаются перечисленные выше задачи, называется **объектно-ориентированным анализом**. Он выполняется до того, как будет написана первая строчка кода, и во многом определяет качество и надежность будущей программы.

В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию объектов (или понятий) в терминах предметной области. Например, в случае информационной системы аэропорта среди понятий должны присутствовать `Plane` (самолет), `Flight` (рейс) и `Pilot` (пилот). В процессе **объектно-ориентированного проектирования** определяются программные объекты и способы их взаимодействия с целью выполнения системных требований. Например, в системе аэропорта программный объект `Plane` может содержать атрибут `tailNumber` (бортовой номер) и метод `getFlightHistory` (получить историю полетов) [4].

Построение объектной модели удобно выполнять с помощью языка ***UML***[1].

И наконец, на этапе **реализации** или **объектно-ориентированного программирования** обеспечивается реализация разработанных компонентов, таких как класс `Plane`, на языке C#.

[1] *UML* (*Unified Modeling Language* – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения.



Понятие предметной области | Визуализация понятия предметной области | Представление средствами объектно-ориентированного языка программирования

Рисунок 1 – Представление объектов с использованием объектно-ориентированного подхода

Рассмотрим основные действия, выполняемые в процессе разработки программной системы, и создаваемые при этом диаграммы [1]. Для построения диаграмм следует использовать один из *UML*-инструментов, например, *Microsoft Visio* [2].

1.1 Определение модели предметной области

Объектно-ориентированный анализ связан с описанием предметной области с точки зрения классификации объектов. Результат анализа выражается в модели предметной области (*domain model*), которая иллюстрируется с помощью набора диаграмм с изображенными на них объектами предметной области.

Модель предметной области нашей задачи представлена на рисунке 2.

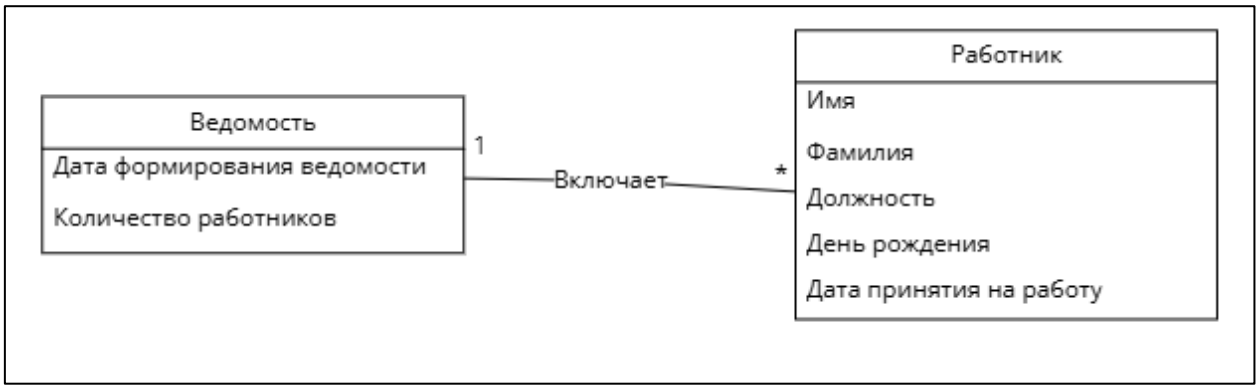


Рисунок 2 – Модель предметной области

Эта модель иллюстрирует понятия «`Ведомость`» и «`Работник`», а также их связи и атрибуты. Понятие «`Ведомость`» включает информацию о работниках вуза. Понятие «`Работник`» включает подробную информацию о каждом работнике вуза.

Модель предметной области – это представление понятий, выраженных в терминах реального мира. Эта модель также называется **концептуальной объектной моделью** (*conceptual object model*).

1.2 Разработка диаграмм классов проектирования

Для описания поведения объектов полезно строить статическое представление системы в виде диаграммы классов проектирования (*design class diagram*). На такой диаграмме отображаются атрибуты и методы классов. На рисунке 3 представлены классы разрабатываемого приложения. Стрелка между классами показывает, что студенческая группа включает несколько студентов.

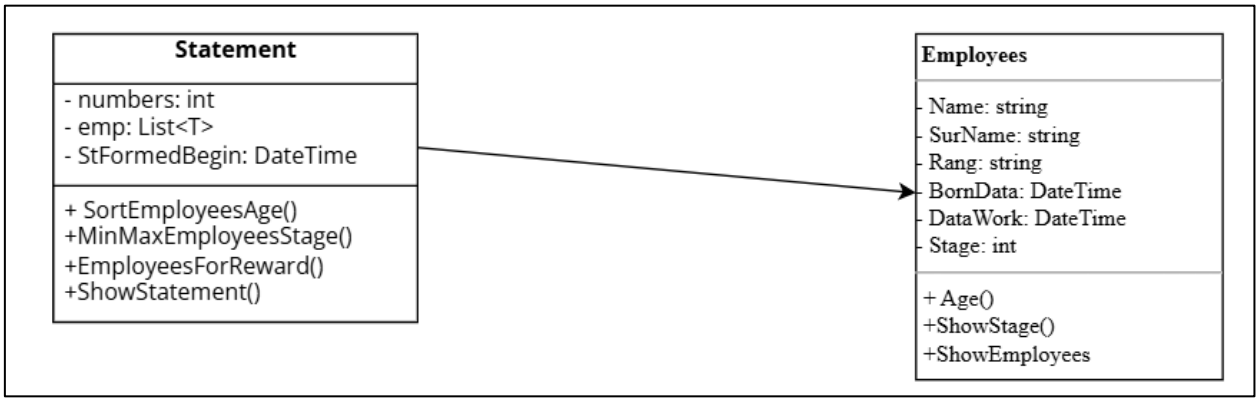


Рисунок 3 – Диаграмма классов проектирования

К классу `Employees` относятся следующие методы:

– `ShowEmployees()` – необходим для отображения информации о конкретном работнике;

– `Age()` – необходим для вычисления возраста работника;

– `ShowStage()`- необходим для вычисления стажа работы каждого работника;

К классу `Statement` относятся следующие методы:

– `SortEmployeesAge()` – сортирует работников по возрасту;

– `MinMaxEmployeesStage()` – выводит работников с минимальным и максимальным стажем;

– `EmployeesForReward()` – выводит список работников, назначенных на поощрение:

– `ShowStatement()` – выводит информацию обо всех работниках.

2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

2.1 Создание и настройка проекта

Реализацию объектной модели начнем с создания проекта.

1. Запускаем интегрированную среду разработки программного обеспечения *Visual Studio*.

2. В качестве типа проекта выбираем проект *Windows Form (.NET Framework)*.

3. Далее в обозревателе решений создаем две папки: *Model* и *View*. Таким образом мы физически выполним разделение файлов проекта на **модель** (папка *Model*, содержащая данные и методы их обработки) и **представление** (папка *View*, включающая данные для взаимодействия модели с пользователем (графический интерфейс)). В папку *View* перемещаем файл *Form1.cs* (рисунок 4).

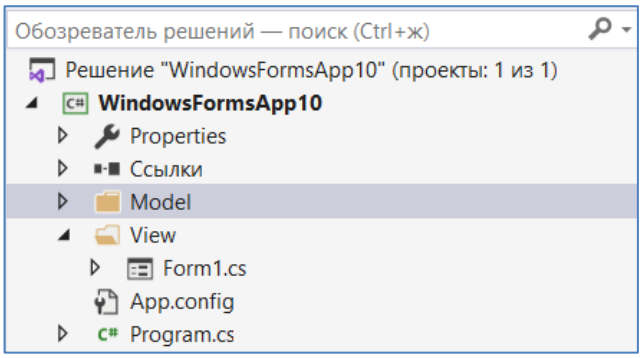


Рисунок 4 – Физическое разделение файлов проекта на модель и представление

4. В качестве типа проекта мы выбрали проект *Windows Form (.NET Framework)*, однако до создания графического интерфейса тестирование приложения пройдет в консоли. Вызовем контекстное меню проекта в обозревателе решений и выберем пункт **Свойства**. В появившемся окне в качестве выходных данных выберем пункт **Консольное приложение**.

2.2 Создание классов

2.2.1 Создание класса *Employees*

1. Создание классов начнем с класса `Employees` (работник). Для этого в контекстном меню папки *Model* выбираем пункт **Добавить**, а затем пункт **Класс** (рисунки 5-6).

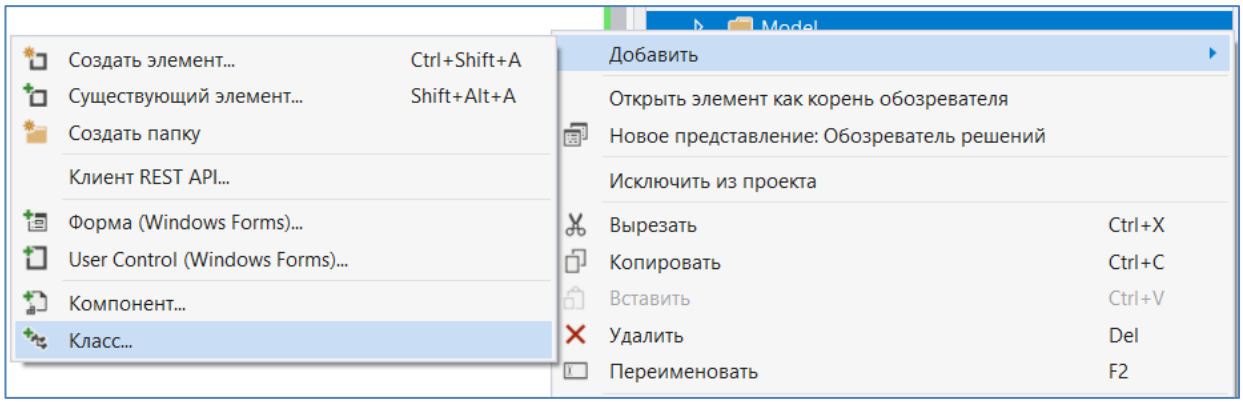


Рисунок 5 – Создание нового класса

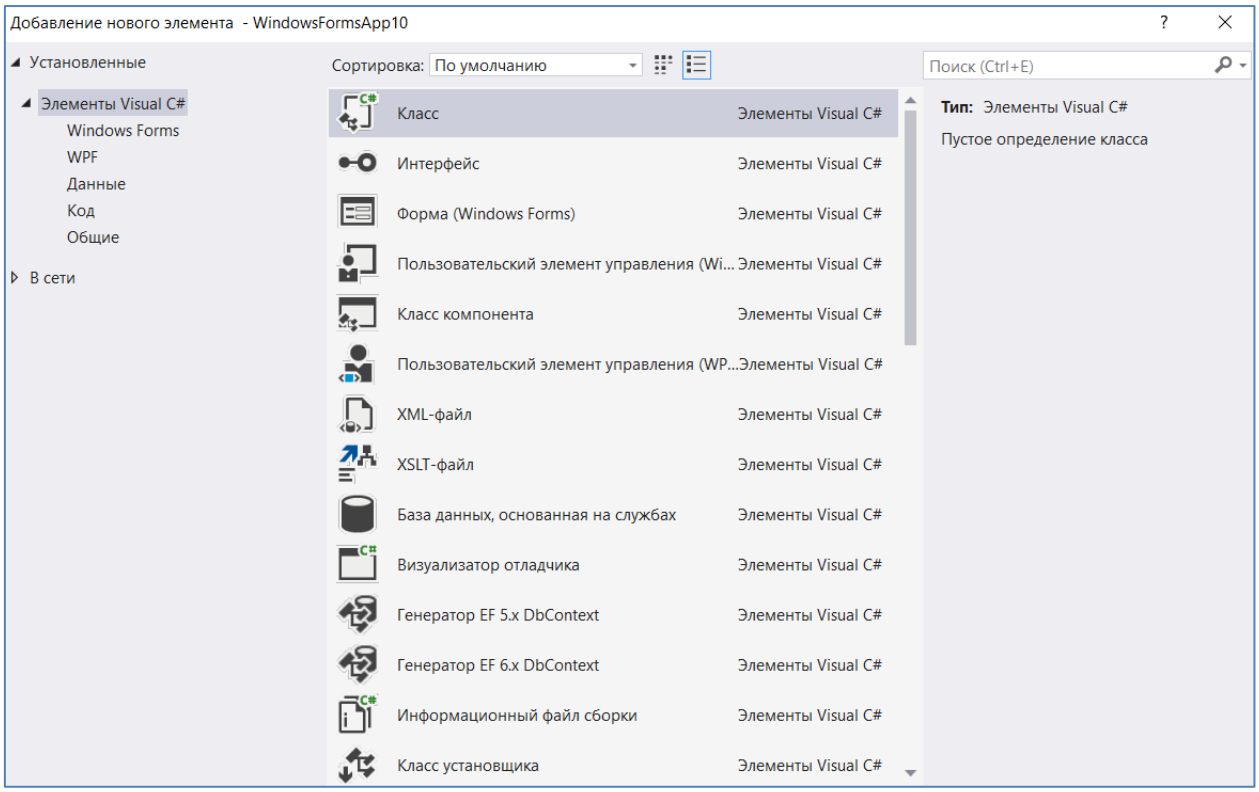


Рисунок 6 – Задание имени нового класса

2. Затем указываем имя класса `Employees.cs` и нажимаем кнопку Добавить. Файл `Employees.cs` появляется в папке *Model*.

3. Создаем поля класса `Employees` (рисунок 7). Все поля класса `Employees` закрытые.

```
internal class Employees
{
    string Name;
    string SurName;
    string Rang;
    DateTime BornData;
    DateTime DataWork;
    int Stage;
```

Рисунок 7 – Поля класса `Employees`

4. Далее необходимо описать конструктор для класса `Employees`, который будет использован при создании экземпляра класса (рисунок 8).

```
public Employees(string Name, string SurName, string Rang, DateTime BornData, DateTime DataWork)
{
    this.Name = Name;
    this.SurName = SurName;
    this.Rang = Rang;
    this.BornData = BornData;
    this.DataWork = DataWork;
}
```

Рисунок 8 – Конструктор класса `Employees`

5. Далее создадим метод `Age()` для определения возраста работника (рисунок 9).

```
public int Age()
{
    DateTime now = DateTime.Today;
    int age = now.Year - BornData.Year;
    now = now.AddYears(-age);
    if (BornData > now) age--;
    return age;
}
```

Рисунок 9 – Метод `Age`

6. В завершении в классе `Employees` необходимо описать метод `ShowEmployees()`, вызывая который мы получим текстовую строку с информацией о работнике (рисунок 10).

```
public string ShowEmployees()
{
    string s = $"{Name}\n";
    s += $"{SurName}\n";
    s += $"{Rang}\n";
    s += $"Возраст: {Age()} лет\n";
    s += $"Стаж работы: {ShowStage()} лет\n";
    s += $"Дата рождения: {BornData.Day}.{BornData.Month}.{BornData.Year}\n";
    s += $"Дата принятия на работу: {DataWork.Day}.{DataWork.Month}.{DataWork.Year}\n";
    return s ;
}
```

Рисунок 10 – Метод `ShowEmployees()`

2.2.2 Создание класса *Statement*

1. Создадим в папке *Model* файл `Statement.cs`.
2. Класс `Statement` будет содержать три закрытых поля (рисунок 11). Это поле `numbers` – количество работников, поле `StFormedBegin` – дата формирования ведомости и список объектов типа `Employees`.

```
internal class Statement
{
    int numbers;
    public List<Employees> emp = new List<Employees>();
    public DateTime StFormedBegin;
```

Рисунок 11 – Поля класса `Statement`

3. Далее необходимо описать конструктор для класса `Statement`, который будет использован при создании ведомости, содержащей информацию обо всех работниках вуза. Для упрощения ввода данных предварительно создадим текстовый файл, содержащий информацию о работниках вуза. Структура текстового файла представлена на рисунке 12.

```
День формирования ведомости
Месяц формирования ведомости
Год формирования ведомости
Количество работников вуза
---
Имя
Фамилия
Должность
День рождения
Месяц рождения
Год рождения
День принятия на работу
Месяц принятия на работу
Год принятия на работу
```

Рисунок 12 – Структура текстового файла с информацией о работниках

Создадим текстовый файл с информацией о работниках вуза и сохраним его в нужном каталоге (рисунок 13).

```
01
01
2000
2
---
Екатерина
Петрова
педагог
08
03
1985
03
08
1994
---
Павел
Егармин
педагог
23
03
1990
03
08
1999
```

Рисунок 13 – Текстовый файл с информацией о работниках вуза

Итоговый код для конструктора `Statement` представлен на рисунке 14.

```
public Statement(OpenFileDialog openFileDialog)
{
    try
    {
        string path = openFileDialog.FileName;
        //string path = "D:\\УЧЁБА\\Программирование\\Лабораторки\\Курсовая работа\\ПодробИнф.txt";
        StreamReader sr = new StreamReader(path, System.Text.Encoding.GetEncoding("UTF-8"));
        int dayborn,monthborn,yearborn,daywork,monthwork,yearwork,
            dayformSt,mounthformSt,yearformSt;
        string Name, SurName, Rang;
        dayformSt = int.Parse(sr.ReadLine());
        mounthformSt = int.Parse(sr.ReadLine());
        yearformSt = int.Parse(sr.ReadLine());
        StFormedBegin = new DateTime(yearformSt, mounthformSt, dayformSt);
        numbers = int.Parse(sr.ReadLine());
        sr.ReadLine();
        for (int i = 0; i < numbers; i++)
        {
            Name = sr.ReadLine();
            SurName = sr.ReadLine();
            Rang = sr.ReadLine();
            dayborn = int.Parse(sr.ReadLine());
            monthborn = int.Parse(sr.ReadLine());
            yearborn = int.Parse(sr.ReadLine());
            daywork = int.Parse(sr.ReadLine());
            monthwork = int.Parse(sr.ReadLine());
            yearwork = int.Parse(sr.ReadLine());
            DateTime dateborn = new DateTime(yearborn, monthborn, dayborn);
            DateTime datework = new DateTime(yearwork, monthwork, daywork);
            Employees employee = new Employees(Name, SurName, Rang, dateborn, datework);
            emp.Add(employee);
            sr.ReadLine();
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Рисунок 14 – Конструктор класса `Statement`

4. В классе `Statement` создадим метод `SortEmployeesAge()` для сортировки списка работников по возрасту. Метод представлен на рисунке 15.

```
public void SortEmployeesAge(bool flag = false)
{
    for (int i = 0; i < emp.Count - 1; i++)
    {
        int a = i;
        for (int j = i + 1; j < emp.Count; j++)
        {
            int b = emp[j].Age();
            int c = emp[a].Age();

            bool needSwap = flag ? b > c : b < c;

            if (needSwap) a = j;
        }
        if (a != i)
        {
            Employees tmp = emp[i];
            emp[i] = emp[a];
            emp[a] = tmp;
        }
    }
}
```

Рисунок 15 – Метод `SortEmployeesAge`

5. Также создадим метод `MinMaxEmployeesStage()` для вывода работников с минимальным и максимальным стажем работы. Обратите внимание, что для вывода, мы формируем текстовую строку (рисунок 16).

```
public string MinMaxEmployeesStage()
{
    Employees MinEmployeeStage = emp[0];
    Employees MaxEmployeeStage = emp[0];

    foreach (Employees emp in emp)
    {
        if (emp.ShowStage() < MinEmployeeStage.ShowStage())
            MinEmployeeStage = emp;

        if (emp.ShowStage() > MaxEmployeeStage.ShowStage())
            MaxEmployeeStage = emp;
    }

    return $"Сотрудник с минимальным стажем:\n{MinEmployeeStage.ShowEmployees()}\n\n" +
           $"Сотрудник с максимальным стажем:\n{MaxEmployeeStage.ShowEmployees()}";
}
```

Рисунок 16 – Метод `ShowGroup`

6. Ещё нам необходимо написать метод `EmployeesForReward()` для отображения работников вуза, представленных к поощрению (рисунок 17).

```
public string EmployeesForReward()
{
    string RewardTxt = "Сотрудники на поощрение (стаж > 10 лет):\n";

    foreach (Employees emp in emp)
    {
        if (emp.ShowStage() > 10)
        {
            RewardTxt += emp.ShowEmployees() + "\n";
        }
    }
    return RewardTxt;
}
```

Рисунок 17 – Метод `EmployeesForReward`

7. В заключение создадим метод `ShowStatement()` для вывода работников вуза (рисунок 18).

```
public string ShowStatement()
{
    string s = "Список работников: \n";
    SortEmployeesAge(true);
    s += "\n";
    for (int i = 0; i < emp.Count; i++)
    {
        s += emp[i].ShowEmployees();
        s += "\n";
    }
    s += EmployeesForReward();
    s += MinMaxEmployeesStage();
    s += "\n";
    s += "Дата формирования ведомости: ";
    s += StFormedBegin;
    return s;
}
```

Рисунок 18 – Метод `ShowStatement`

2.2.3 Тестирование приложения

1. Для тестирования приложения необходимо в главном методе с помощью программной команды `Statement workers = new Statement();` создать группу рабочих вуза (рисунок 19).

2. Затем командой `Console.WriteLine(workers.ShowStatement());` вывести на экран подробную информацию о каждом студенте. В коде также используем обработчик исключений (рисунок 19).

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    //Application.Run(new Form1());
    try
    {
        Statement workers = new Statement();
        Console.WriteLine(workers.ShowStatement());
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadKey();
}
```

Рисунок 19 – Главный метод программы

2.3 Создание графического интерфейса

2.3.1 Настройка проекта

1. Вызовем контекстное меню проекта в обозревателе решений и выберем пункт **Свойства**. В появившемся окне в качестве выходных данных выберем пункт **Приложение Windows**.

2. В главном методе программы включим запуск главной формы, отменив создание новой студенческой группы и вывода списка группы на экран (рисунок 20).

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
```

Рисунок 20 – Главный метод программы

Запустим приложение и убедимся, что главная форма активна и появляется на экране.

2.3.2 Настройка внешнего вида форм приложения

Интерфейс приложения будет состоять из 3-х форм (приложение А):

– *Form1* – главная форма приложения (рисунок А.1);

– *Form2* – форма для вывода результатов (рисунок А.2);

– *Form3* – форма для вывода информации о приложении (рисунок А.3).

1. Создание графического интерфейса начнем с настройки внешнего вида **главной формы**. Используя окно свойств, установим следующие параметры для формы:

– заголовок – Диалоговая программа «Деканат»;

– ширина – 1000 пикселей;

– высота – 600 пикселей;
– стиль рамки (*FormBorderStyle*) – *Fixed3D*;
– задний фон (*BackColor*) – *ActiveCaption*.

2. Настроим для формы кнопку выхода из приложения  . Для этого для обработчика события *FormClosed* формы добавим команду `Application.Exit();`

3. Добавим на форму метку – заголовок (*Label*). Установим для нее следующие свойства:
– имя (*Name*) – *label_caption*;
– заголовок (*Text*) – Деканат;
– шрифт (*Font*) – *Constantia*;
– полужирный (*Bold*) – *True*;
– размер шрифта (*Size*) – 34;
– цвет (*ForeColor*) – *Info*;
– расположение (*Location*) – 344; 9.

4. Добавим на форму метку – название вуза (рисунок А.1). Установим для нее следующие свойства:
– имя (*Name*) – *label_university*;
– заголовок (*Text*) – ФГБОУ ВО «Сибирский государственный университет имени М.Ф. Решетнева» Лесосибирский филиал;
– шрифт (*Font*) – *Microsoft Sans Serif*;
– полужирный (*Bold*) – *True*;
– размер шрифта (*Size*) – 8;
– цвет (*ForeColor*) – *Info*;
– расположение (*Location*) – 81; 516.

5. Разместим на форме компонент *Panel*, установив для него следующие свойства:
– имя (*Name*) – *panel_main*;
– цвет фона (*BackColor*) – *GradientActiveCaption*;
– размер (*size*) – 981; 66;
– расположение (*Location*) – 0; 425.

6. На панель добавим метку (*Label*) для перехода на форму с расчетами, установив для нее следующие свойства:
– имя (*Name*) – *label_start*;
– заголовок (*Text*) – Запуск;
– шрифт (*Font*) – *Constantia*;
– полужирный (*Bold*) – *True*;
– размер шрифта (*Size*) – 20;
– цвет (*ForeColor*) – *MenuText*;
– расположение (*Location*) – 44; 12.

7. Далее на панель добавим метку (*Label*) для перехода на форму с информацией, установив для нее следующие свойства:

– имя (*Name*) – *label_information*;
– заголовок (*Text*) – Информация;
– шрифт (*Font*) – *Constantia*;
– полужирный (*Bold*) – *True*;
– размер шрифта (*Size*) – 20;
– цвет (*ForeColor*) – *MenuText*;
– расположение (*Location*) – 365; 12.

8. Далее на панель добавим метку (*Label*) для выхода из приложения, установив для нее следующие свойства:

– имя (*Name*) – *label_exit*;
– заголовок (*Text*) – Выход;
– шрифт (*Font*) – *Constantia*;
– полужирный (*Bold*) – *True*;
– размер шрифта (*Size*) – 20;
– цвет (*ForeColor*) – *MenuText*;
– расположение (*Location*) – 780; 12.

9. Добавим на главную форму произвольный рисунок с помощью компонента *PictureBox* (рисунок А.1).

10. Аналогичным образом настраиваем внешний вид оставшихся форм (рисунок А.2, рисунок А.3). Для вывода результата (*Form2*) размещаем на форме компонент *RichTextBox*.

2.3.3 Настройка обработчиков событий

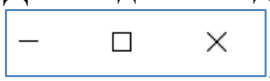
1. Настроим обработчики событий для меток главной формы («Запуск», «Информация», «Выход») таким образом, чтобы при наведении курсора мыши надпись была подчеркнута (рисунок 21), при смещении указателя – возвращалась в исходную форму (рисунок 22). Аналогичным образом настроим все соответствующие метки на всех формах.

```
private void label_start_MouseHover(object sender, EventArgs e)
{
    label_start.Font = new Font(label_start.Font, FontStyle.Underline | FontStyle.Bold);
}
```

Рисунок 21 – Событие *MouseHover* для текстовой метки

```
private void label_start_MouseLeave(object sender, EventArgs e)
{
    label_start.Font = new Font(label_start.Font, FontStyle.Bold);
}
```

Рисунок 22 – Событие *MouseLeave* для текстовой метки

11. Далее для каждой формы настроим кнопку выхода из приложения  . Для этого для обработчика события *FormClosed* каждой формы добавим команду `Application.Exit();`
12. Аналогичным образом настроим обработчик события *Click* для меток «Выход» каждой формы, добавив в него команду `Application.Exit();`
13. Далее настроим переход между формами. Для запуска формы с расчетами (*Form2*) из главной формы (*Form1*) в обработчике события *Click* для метки «Запуск» добавим соответствующий код (рисунок 23).

```
private void label_start_Click(object sender, EventArgs e)
{
    Form2 result = new Form2();
    result.Show();
    result.Location = this.Location;
    this.Hide();
}
```

Рисунок 23 – Настройка перехода между формами

Для обратного перехода (от формы с результатом к главной форме) код представлен на рисунке 24. Код отличается от предыдущего тем, что здесь не создается дополнительный экземпляр главной формы, а открывается существующий (скрытый). Настраиваем остальные переходы аналогично.

```
private void label_main_Click(object sender, EventArgs e)
{
    Form Form1 = Application.OpenForms[0];
    Form1.Show();
    Form1.Location = this.Location;
    this.Hide();
}
```

Рисунок 24 – Настройка перехода между формами

14. В заключении настроим обработчики событий *Click* для кнопок «Запуск расчетов» и «Сохранить данные», расположенных на форме для вывода результата (рисунки 25-26). Для класса `StreamWriter` необходимо подключить пространство имен `System.IO`.

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы (*.*)|*.*";
    openFileDialog.ShowDialog();
    Statement statement = new Statement(openFileDialog);
    richTextBox1.Text = statement.ShowStatement();
}
```

Рисунок 25 – Обработчик события для кнопки «Запуск расчетов»

```
private void label_save_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы (*.*)|*.*";
    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        StreamWriter streamWriter = new StreamWriter(saveFileDialog.FileName);
        streamWriter.WriteLine(richTextBox1.Text);
        streamWriter.Close();
    }

}
```

Рисунок 26 – Обработчик события для кнопки «Сохранить данные»

Результат работы приложения приведен на рисунке А.2.

ЗАКЛЮЧЕНИЕ

В курсовой работе спроектировано и реализовано на языке объектно-ориентированного программирования C# приложение с графическим интерфейсом «Деканат». В ходе работы:

- построена модель предметной области;
- построена диаграмма последовательностей;
- построена диаграмма классов проектирования;
- выполнено объектно-ориентированное программирование приложения;
- спроектирован и реализован графический интерфейс;
- выполнено тестирование и отладка приложения.

Таким образом, цель курсовой работы выполнена, задачи, поставленные в ходе ее выполнения, решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Простое руководство по UML-диаграммам и моделированию баз данных. – Текст : электронный // Microsoft 365 Team : [сайт]. – URL: https://www.microsoft.com/ru-ru/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling (дата обращения: 26.04.2021).

2. Visio: Работайте с наглядным представлением данных в любое время, где бы вы ни находились.. – Текст : электронный // Microsoft 365 Team : [сайт]. – URL: https://www.microsoft.com/ru-ru/microsoft-365/visio/flowchart-software (дата обращения: 26.04.2021).

3. C Sharp. – Текст : электронный // Материал из Википедии — свободной энциклопедии. – URL: https://ru.wikipedia.org/wiki/C_Sharp (дата обращения: 26.04.2021).

4. Ларман К. Применение UML 2.0 и шаблонов проектирования – Москва : Издательский дом «Вильямс», 2013. – 737 с. Текст : непосредственный.

ПРИЛОЖЕНИЕ А. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

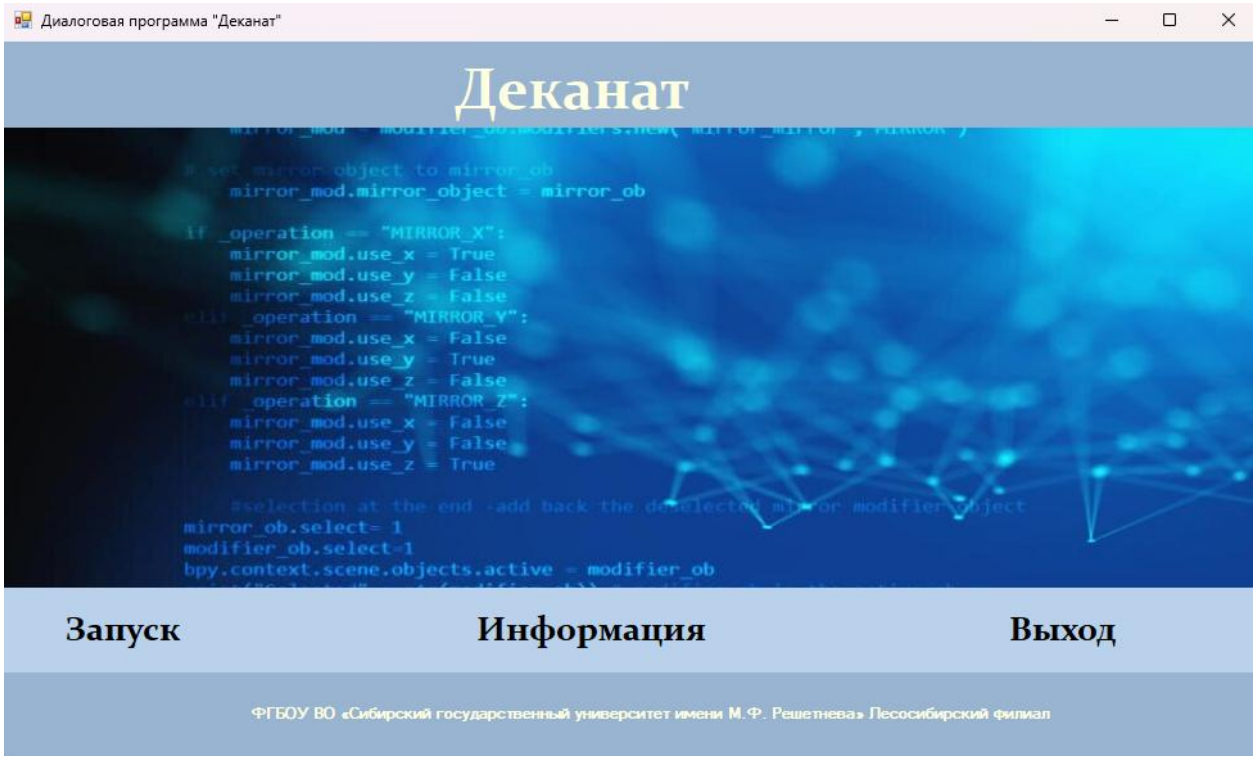


Рисунок А.1 – Главная форма приложения

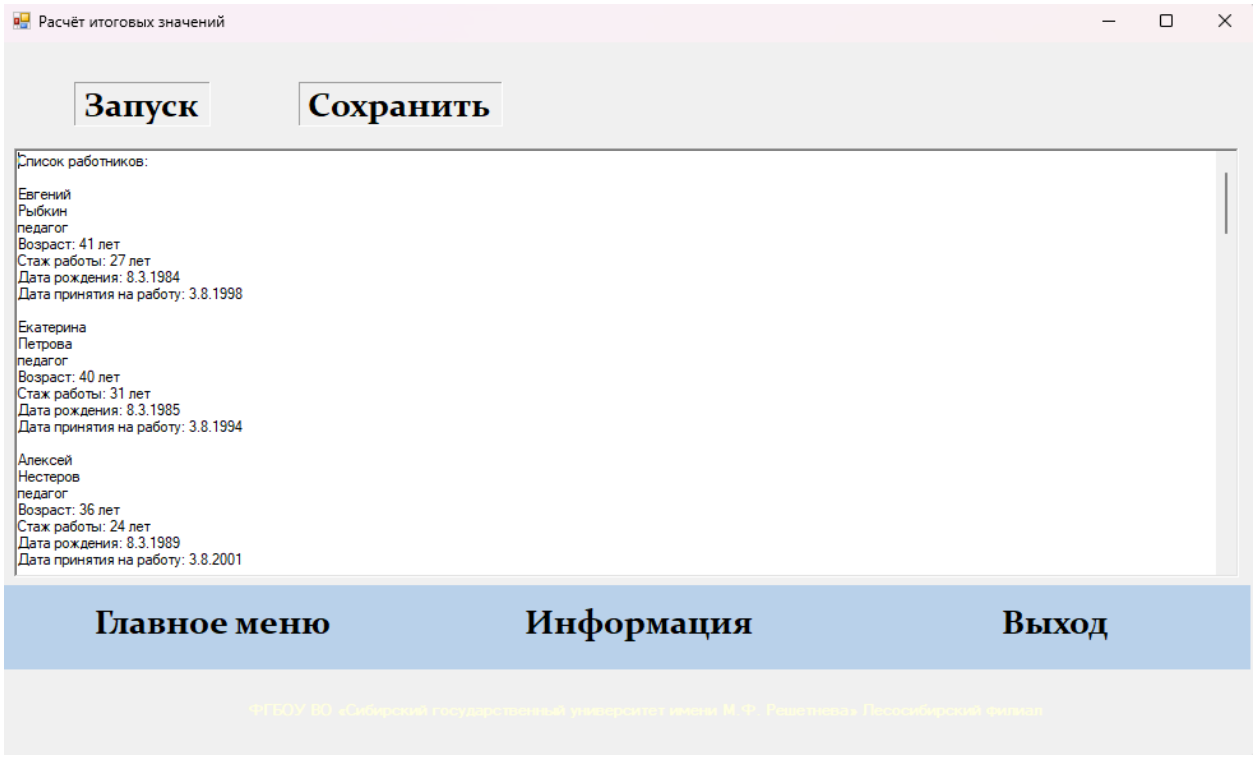


Рисунок А.2 – Расчет итоговых значений

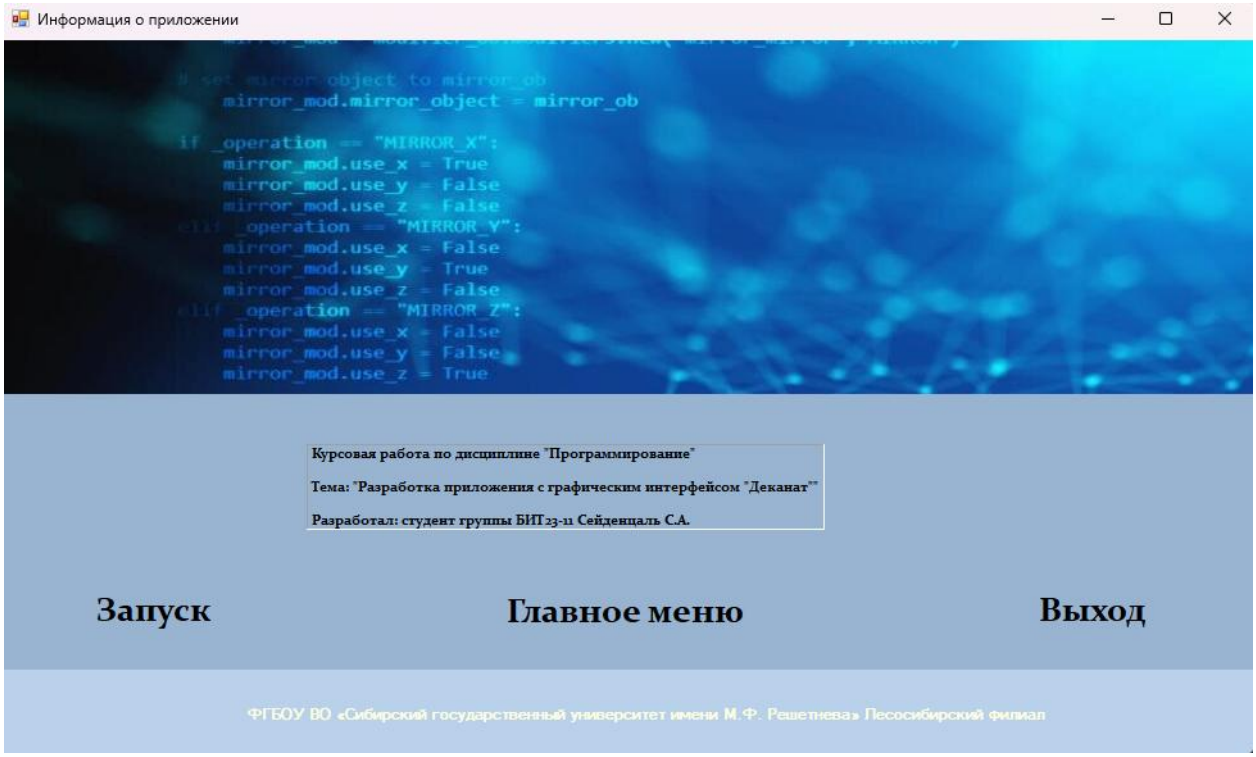


Рисунок А.3 – Информация о приложении

ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД

Б1. Главный класс приложения

```
namespace Курсовая_работа
{
    internal static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Б2. Класс *Employees*

```
    internal class Employees
{
    string Name;
    string SurName;
    string Rang;
    DateTime BornData;
    DateTime DataWork;
    int Stage;
    public Employees(string Name, string SurName, string Rang, DateTime BornData,
DateTime DataWork)
    {
        this.Name = Name;
        this.SurName = SurName;
        this.Rang = Rang;
        this.BornData = BornData;
        this.DataWork = DataWork;
    }
    public DateTime datawork
    {
        get
        {
            return DataWork;
        }
    }
    public DateTime databorn
    {
        get
        {
            return BornData;
        }
    }
    public int Age()
    {
        DateTime now = DateTime.Today;
        int age = now.Year - BornData.Year;
        now = now.AddYears(-age);
        if (BornData > now) age--;
        return age;
    }
    public int ShowStage()
```

```
    {
        int stage;
        DateTime now = DateTime.Today;
        return stage = now.Year - datawork.Year;
    }
    public string ShowEmployees()
    {
        string s = $"{Name}\n";
        s += $"{SurName}\n";
        s += $"{Rang}\n";
        s += $"Возраст: {Age()} лет\n";
        s += $"Стаж работы: {ShowStage()} лет\n";
        s += $"Дата рождения: {BornData.Day}.{BornData.Month}.{BornData.Year}\n";
        s += $"Дата принятия на работу:
{DataWork.Day}.{DataWork.Month}.{DataWork.Year}\n";
        return s ;
    }
}
```

Б3. Класс *Statement*

```
    internal class Statement
{
    int numbers;
    public List<Employees> emp = new List<Employees>();
    public DateTime StFormedBegin;

    public Statement(FileDialog openFileDialog)
    {
        try
        {
            string path = openFileDialog.FileName;
            //string path = "D:\\УЧЁБА\\Программирование\\Лабораторки\\Курсовая
работа\\ПодробИнф.txt";
            StreamReader sr = new StreamReader(path,
System.Text.Encoding.GetEncoding("UTF-8"));
            int dayborn,monthborn,yearborn,daywork,monthwork,yearwork,
                dayformSt,mounthformSt,yearformSt;
            string Name, SurName, Rang;
            dayformSt = int.Parse(sr.ReadLine());
            mounthformSt = int.Parse(sr.ReadLine());
            yearformSt = int.Parse(sr.ReadLine());
            StFormedBegin = new DateTime(yearformSt, mounthformSt, dayformSt);
            numbers = int.Parse(sr.ReadLine());
            sr.ReadLine();
            for (int i = 0; i < numbers; i++)
            {
                Name = sr.ReadLine();
                SurName = sr.ReadLine();
                Rang = sr.ReadLine();
                dayborn = int.Parse(sr.ReadLine());
                monthborn = int.Parse(sr.ReadLine());
                yearborn = int.Parse(sr.ReadLine());
                daywork = int.Parse(sr.ReadLine());
                monthwork = int.Parse(sr.ReadLine());
                yearwork = int.Parse(sr.ReadLine());
                DateTime dateborn = new DateTime(yearborn, monthborn, dayborn);
                DateTime datework = new DateTime(yearwork, monthwork, daywork);
                Employees employee = new Employees(Name, SurName, Rang, dateborn,
datework);
                emp.Add(employee);
                sr.ReadLine();
            }
        }
        catch (Exception e)
        {
```

```
            Console.WriteLine(e.Message);
        }
    }

    public void SortEmployeesAge(bool flag = false)
    {
        for (int i = 0; i < emp.Count - 1; i++)
        {
            int a = i;
            for (int j = i + 1; j < emp.Count; j++)
            {
                int b = emp[j].Age();
                int c = emp[a].Age();

                bool needSwap = flag ? b > c : b < c;

                if (needSwap) a = j;
            }
            if (a != i)
            {
                Employees tmp = emp[i];
                emp[i] = emp[a];
                emp[a] = tmp;
            }
        }
    }

    public string MinMaxEmployeesStage()
    {
        Employees MinEmployeeStage = emp[0];
        Employees MaxEmployeeStage = emp[0];

        foreach (Employees emp in emp)
        {
            if (emp.ShowStage() < MinEmployeeStage.ShowStage())
                MinEmployeeStage = emp;

            if (emp.ShowStage() > MaxEmployeeStage.ShowStage())
                MaxEmployeeStage = emp;
        }

        return $"Сотрудник с минимальным
стажем:\n{MinEmployeeStage.ShowEmployees()}\n\n" +
               $"Сотрудник с максимальным
стажем:\n{MaxEmployeeStage.ShowEmployees()}";
    }

    public string EmployeesForReward()
    {
        string RewardTxt = "Сотрудники на поощрение (стаж > 10 лет):\n";

        foreach (Employees emp in emp)
        {
            if (emp.ShowStage() > 10)
            {
                RewardTxt += emp.ShowEmployees() + "\n";
            }
        }
        return RewardTxt;
    }

    public string ShowStatement()
    {
        string s = "Список работников: \n";
        SortEmployeesAge(true);
        s += "\n";
```

```
        for (int i = 0; i < emp.Count; i++)
        {
            s += emp[i].ShowEmployees();
            s += "\n";
        }
        s += EmployeesForReward();
        s += MinMaxEmployeesStage();
        s += "\n";
        s += "Дата формирования ведомости: ";
        s += StFormedBegin;
        return s;
    }
}
```

Б4. Главная форма

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void label1_Click(object sender, EventArgs e)
    {

    }
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }

    private void label_exit_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
    private void label_start_Click(object sender, EventArgs e)
    {
        Form2 result = new Form2();
        result.Show();
        result.Location = this.Location;
        this.Hide();
    }
    private void label_start_MouseHover(object sender, EventArgs e)
    {
        label_start.Font = new Font(label_start.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_start_MouseLeave(object sender, EventArgs e)
    {
        label_start.Font = new Font(label_start.Font, FontStyle.Bold);
    }
    private void label_information_Click(object sender, EventArgs e)
    {
        Form3 result = new Form3();
        result.Show();
        result.Location = this.Location;
        this.Hide();
    }
    private void label_information_MouseHover(object sender, EventArgs e)
    {
        label_information.Font = new Font(label_information.Font,
FontStyle.Underline | FontStyle.Bold);
    }
    private void label_information_MouseLeave(object sender, EventArgs e)
    {
        label_information.Font = new Font(label_information.Font, FontStyle.Bold);
    }
```

```
    private void label_exit_MouseHover(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_exit_MouseLeave(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Bold);
    }
      }
```

Б5. Форма «Информация»

```
public partial class Form3 : Form
{
    public Form3()
    {
        InitializeComponent();
    }
    private void Form3_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }
    private void label1_Click(object sender, EventArgs e)
    {

    }
    private void label_exit_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
    private void label_main_Click(object sender, EventArgs e)
    {
        Form Form1 = Application.OpenForms[0];
        Form1.Show();
        Form1.Location = this.Location;
        this.Hide();
    }
    private void label_exit_MouseHover(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_exit_MouseLeave(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Bold);
    }
    private void label_start_MouseHover(object sender, EventArgs e)
    {
        label_start.Font = new Font(label_start.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_start_MouseLeave(object sender, EventArgs e)
    {
        label_start.Font = new Font(label_start.Font, FontStyle.Bold);
    }
    private void label_mainmenu_MouseHover(object sender, EventArgs e)
    {
        label_mainmenu.Font = new Font(label_mainmenu.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_mainmenu_MouseLeave(object sender, EventArgs e)
    {
        label_mainmenu.Font = new Font(label_mainmenu.Font, FontStyle.Bold);
    }
```

```
    private void label_start_Click(object sender, EventArgs e)
    {
        Form2 result = new Form2();
        result.Show();
        result.Location = this.Location;
        this.Hide();
    }
      }
```

Б6. Форма «Расчет итоговых значений»

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
    }
    private void Form2_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы
(*.*)|*.*";
        openFileDialog.ShowDialog();
        Statement statement = new Statement(openFileDialog);
        richTextBox1.Text = statement.ShowStatement();
    }

    private void label_save_Click(object sender, EventArgs e)
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы
(*.*)|*.*";
        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
            StreamWriter streamWriter = new StreamWriter(saveFileDialog.FileName);
            streamWriter.WriteLine(richTextBox1.Text);
            streamWriter.Close();
        }

    }
    private void label_exit_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
    private void label_main_Click(object sender, EventArgs e)
    {
        Form Form1 = Application.OpenForms[0];
        Form1.Show();
        Form1.Location = this.Location;
        this.Hide();
    }
    private void label_exit_MouseHover(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_exit_MouseLeave(object sender, EventArgs e)
    {
        label_exit.Font = new Font(label_exit.Font, FontStyle.Bold);
    }
    private void label_information_MouseHover(object sender, EventArgs e)
    {
```

```
        label_information.Font = new Font(label_information.Font,
FontStyle.Underline | FontStyle.Bold);
    }
    private void label_information_MouseLeave(object sender, EventArgs e)
    {
        label_information.Font = new Font(label_information.Font, FontStyle.Bold);
    }
    private void label_mainmenu_MouseHover(object sender, EventArgs e)
    {
        label_mainmenu.Font = new Font(label_mainmenu.Font, FontStyle.Underline |
FontStyle.Bold);
    }
    private void label_mainmenu_MouseLeave(object sender, EventArgs e)
    {
        label_mainmenu.Font = new Font(label_mainmenu.Font, FontStyle.Bold);
    }
    private void label_information_Click(object sender, EventArgs e)
    {
        Form3 result = new Form3();
        result.Show();
        result.Location = this.Location;
        this.Hide();
    }
      }
```