Объектно-ориентированное программирование. Классы в C#

ОГУ

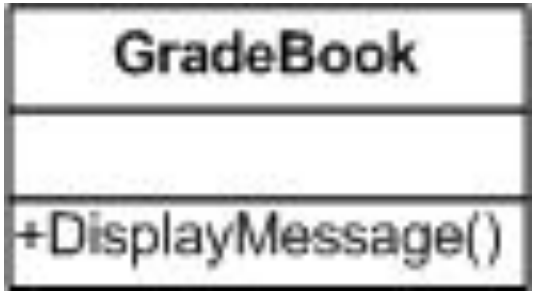
Кафедра ВТиЗИ

Галимов Р.Р.

ОПРЕДЕЛЕНИЕ КЛАССОВ

```
public class GradeBook
    {
        public void DisplayMessage()
        {
            Console.WriteLine("Welcome!!!");
        }
    }
 class Program
    {
        static void Main(string[] args)
        {
            GradeBook myGradeBook = new GradeBook();
            myGradeBook.DisplayMessage();
        }
    }
```

ОПРЕДЕЛЕНИЕ КЛАССОВ



ОПРЕДЕЛЕНИЕ КЛАССОВ

- **Поле**. Так называется член-переменная, содержащий некоторое значение. В ООП поля иногда называют данными объекта. К полю можно применять несколько модификаторов в зависимости от того, как вы собираетесь его использовать. В число модификаторов входят static, readonly и const.
- **Метод**. Это реальный код, воздействующий на данные объекта (или поля). Здесь мы сосредоточимся на определении данных класса.

ОПРЕДЕЛЕНИЕ КЛАССОВ

- **Свойства.** Их иногда называют ≪разумными≫ полями (smart fields), поскольку они на самом деле являются методами, которые клиенты класса воспринимают как поля. Это обеспечивает клиентам большую степень абстрагирования за счет того, что им не нужно знать, обращаются ли они к полю напрямую или через вызов метода-аксессора.

МОДИФИКАТОРЫ ДОСТУПА

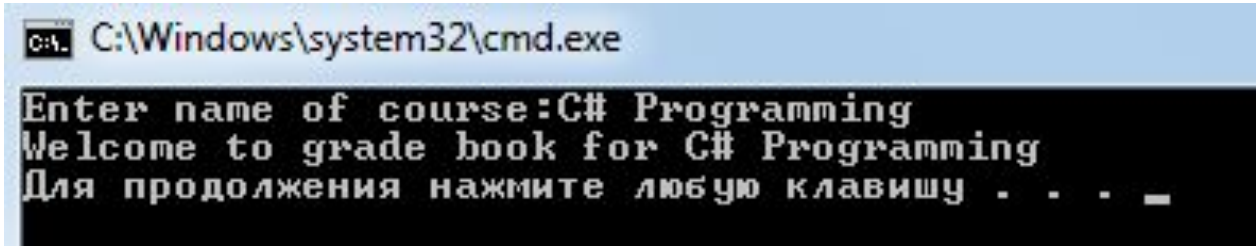
Модификаторы доступа в C#

Модификатор доступа	Описание
public	Член доступен вне определения класса и иерархии производных классов.
protected	Член невидим за пределами класса, к нему могут обращаться только производные классы.
private	Член недоступен за пределами области видимости определяющего его класса. Поэтому доступа к этим членам нет даже у производных классов.
internal	Член видим только в пределах текущей единицы компиляции. Модификатор доступа internal в плане ограничения доступа является гибридом public и protected, зависимым от местоположения кода.

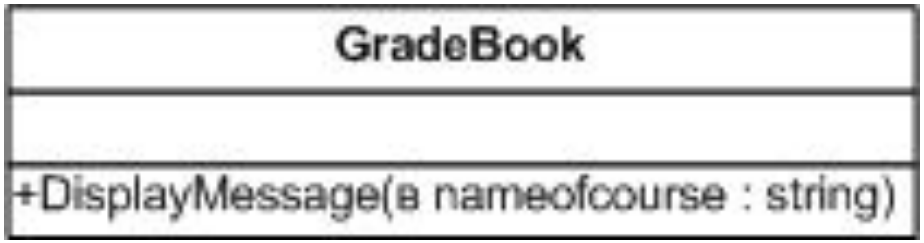
- Если вы не хотите оставить модификатор доступа для данного члена по умолчанию (private), задайте для него явно модификатор доступа.

Объявление метода с параметром

```
public class GradeBook
    {
        public void DisplayMessage(string coursename)
        {
            Console.WriteLine("Welcome to grade book for "+coursename);
        }
    }
  class Program
      {
          static void Main(string[] args)
          {
              Console.Write("Enter name of course:");
              string nameofcourse = Console.ReadLine();
              GradeBook myGradeBook = new GradeBook();
              myGradeBook.DisplayMessage(nameofcourse);
          }
      }
```



Объявление метода с параметром



Переменные экземпляров и свойства

- Объект обладает атрибутами, которые сопровождают его при использовании в приложении.
- Такие атрибуты существуют до вызова метода объекта и продолжают существовать после того, как метод будет выполнен. В объявлениях классов атрибуты представляются переменными. Такие переменные называются *полями и объявляются в объявлении класса, но за пределами объявлений* методов этого класса.
- Когда каждый объект класса поддерживает собственную копию атрибута, поле, представляющее атрибут, также называется *переменной экземпляра - каждый объект (экземпляр) класса имеет собственную копию переменной.*
- Класс обычно содержит одно или несколько свойств для работы с атрибутами, принадлежащими конкретному объекту класса.
- В примере, представлен класс GradeBook с переменной экземпляра courseName, представляющей название учебного курса конкретного объекта GradeBook, а также свойством CourseName для работы с courseName.

Переменные экземпляров и свойства

```
public class GradeBook
    {
        private string coursename;
        public string Coursename
        {
            get             {
                return coursename;
            }
            set             {
                coursename = value;
            }
        }
        public void DisplayMessage()
        {
            Console.WriteLine("Welcome to grade book for "+coursename);
        }     }
 class Program
    {
        static void Main(string[] args)
        {
            GradeBook myGradeBook = new GradeBook();
            Console.Write("Enter name of course:");
            myGradeBook.Coursename = Console.ReadLine();
            myGradeBook.DisplayMessage();
        }     }
```

Чтение и запись закрытых переменных экземпляров

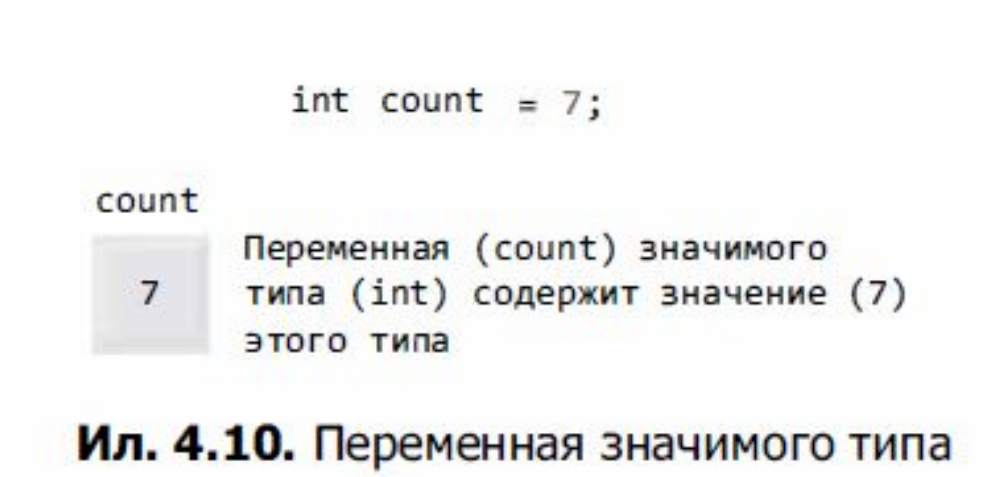
- Как разрешить программе работать с закрытыми переменными экземпляров класса, но проследить за тем, чтобы они имели корректные значения?
- Нужно предоставить *в распоряжение программиста контролируемые механизмы чтения и записи* значения переменной.
- И хотя вы можете определить методы вида GetCourseName и SetCourseName, свойства C# предоставляют более элегантное решение.

Чтение и запись закрытых переменных экземпляров

- Свойства содержат методы доступа, которые реализуют чтение и запись данных.
- Объявление свойства может содержать get-метод доступа и/или set-метод.
- Get-метод читает значение закрытой переменной экземпляра courseName; set-метод позволяет клиенту изменить courseName.
- После того как вы определите свойство, оно используется *в коде как переменная.*
- Например, свойству можно присвоить значение оператором=. Эта команда выполняет set-метод свойства для задания значения соответствующей переменной экземпляра.
- Аналогичным образом при обращении к свойству за его значением (например, для вывода на экран) выполняется код get-метода свойства.

Значимые типы и ссылочные типы

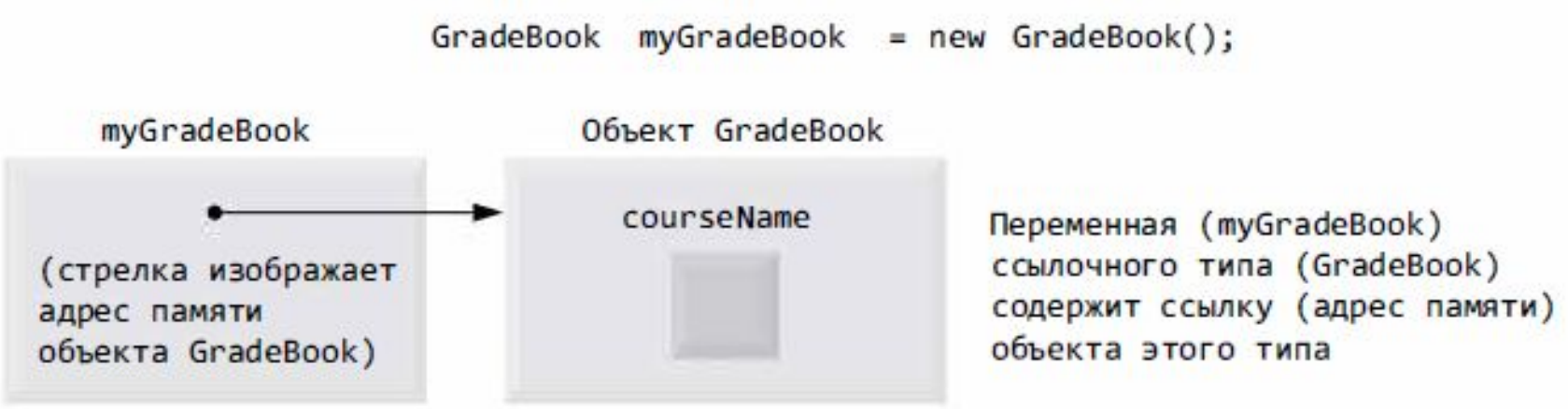
- Типы в языке C# делятся на две категории: значимые типы (value types) и ссылочные типы (reference types).
- **Значимые типы**
- Простые типы C# (такие, как int и douЬle) относятся к категории значимых.
- Переменная значимого типа просто содержит значение некоторого типа.



Ил. 4.10. Переменная значимого типа

Значимые типы и ссылочные типы

- Ссылочные типы
- С другой стороны, переменная ссылочного типа (иногда такие переменные называются *ссылками - references) содержит адрес ячейки памяти, в которой хранятся* данные, ассоциированные с переменной.
- Говорят, что такая переменная ссылается на объект *в программе.*



Значимые типы и ссылочные типы

- Инициализация ссылочных переменных значением null
- Переменные экземпляров ссылочного типа по умолчанию инициализируются значением null.
- Тип string является ссылочным типом, поэтому переменная courseName изображена пустым блоком, изображающим переменную со значением null.
- Строковая переменная со значением null не является пустой строкой, которая представляется литералом "« или конструкцией string. Empty.
- Значение null представляет ссылку, которая не указывает на объект, - тогда как пустая строка представляет строковый объект, не содержащий символов

Инициализация объектов конструкторами

- при создании объекта GradeBook происходит создание объекта, а его переменная экземпляра courseName по умолчанию инициализируется значением null.
- А если вы хотите задать название учебного курса при создании объекта GradeBook? Каждый класс может предоставить конструктор, который будет использоваться для инициализации объектов класса при их создании.
- Оператор new вызывает конструктор класса для проведения инициализации. Вызов конструктора обозначается именем класса, за которым следуют круглые скобки.
- Пустые круглые скобки *в newGradeBook() обозначают вызов конструктора* класса без аргументов.
- Компилятор предоставляет открытый конструктор по умолчанию без параметров для любого класса, не содержащего явного определения конструктора, так что конструктор имеется у каждого класса.
- Конструктор по умолчанию не изменяет значения переменных экземпляров, назначаемые по умолчанию.

Нестандартная инициализация в конструкторах

- При объявлении класса разработчик может предоставить собственный конструктор (или несколько конструкторов*) для проведения нестандартной* инициализации объектов класса.
- Допустим, вы хотите, чтобы при создании объекта GradeBook можно было сразу задать название учебного курса:

GradeBook myGradeBook =new GradeBook("(5101 Introduction to C# Programming");

Аргумент "CS101IntroductiontoC#Programming" передается конструктору объекта GradeBook и используется для инициализации CourseName.

Каждый раз, когда *в программе* создается новый объект GradeBook, вы можете задавать новое название.

- Чтобы приведенная выше команда работала, класс должен иметь конструктор с параметром string.

Нестандартная инициализация в конструкторах

```
public class GradeBook
    {
        private string coursename;
        public string Coursename
        {
            get
            {
                return coursename;
            }
            set
            {
                coursename = value;
            }
        }
        public void DisplayMessage()
        {
            Console.WriteLine("Welcome to grade book for "+coursename);
        }
        public GradeBook(string name)
        {
            Coursename = name;
        }
```

Нестандартная инициализация в конструкторах

```
class Program
    {
        static void Main(string[] args)
        {
            GradeBook myGradeBook1 = new
 GradeBook("Introduction to C# Programming");
            GradeBook myGradeBook2 = new
 GradeBook("Data Structures in C#");
            Console.Write("gradebook1 name is:
 {0}\n",myGradeBook1.Coursename);
            Console.Write("gradebook2 name is:
 {0}\n", myGradeBook2.Coursename);

        }
    }
```

Объявление и создание массивов

- Массивы занимают место *в памяти. Так как они являются объектами, обычно они* создаются ключевым словом new.
- При создании объекта массива тип и количество его элементов указываются *в выражении создания массива, использующем ключевое* слово new. Такое выражение возвращает ссылку, которая сохраняется *в переменной* массива.
- Следующее объявление и выражение создания массива создает объект массива с 12 элементами и сохраняет ссылку на него *в переменной* ***c****:*
- int[] c= new int[12];

Изменение размеров массива

- Хотя массивы имеют фиксированную длину, вы можете использовать статический метод Resize класса Array с двумя аргументами (массив и новая длина) для создания нового массива заданной длины.
- Метод копирует содержимое старого массива в новый массив и задает переменной, полученной в первом аргументе, ссылку на новый массив.
- Рассмотрим пример:
- int[] newArray = new int[5];
- Array.Resize(ref newArray, 10);
- Переменная newArray изначально содержит ссылку на массив из 5 элементов. В результате вызова метода Resize переменная newArray начинает указывать на новый массив из 10 элементов.
- Если новый массив меньше старого, то все данные, не поместившиеся в новый массив, усекаются без каких-либо предупреждений.

Пример

- array = new int[5]; // 5 элементов типа int
- Console.Writeline("{0}{1,8}", "Index", "Value"); // Заголовки
- // Вывод значений всех элементов
- for (int counter = 0; counter < array.Length; ++counter)
- Console.Writeline("{0,5}{1,8}", counter, array[counter]);

Использование инициализатора массива

- Приложение может создать массив и инициализировать его элементы при помощи *инициализатора массива - разделенного запятыми списка выражений (называемого списком инициализаторов), заключенного в фигурные скобки.*
- *В этом случае* длина массива определяется количеством элементов в списке инициализаторов.
- Например int[] n = { 10, 20, 30, 40, 50 };

Пример

```
class CSotrudnik
  {   private string family;
      private string name;
      private string otch;
      private int countdays;
      private float salary;
      public string Family { get { return family; } }
      public bool SetFamily(string sfamily)
      {
          if (string.IsNullOrEmpty(sfamily))
              return false;
          else {
              family = sfamily;
              return true;              }
      }
      public string Name { get { return name; } }
}



}
```

```
public bool SetName(string sname)
        {
            if (string.IsNullOrEmpty(sname))
                return false;
            else
            {
                name = sname;
                return true;                }
        }
        public string Otch { get { return otch; } }
        public bool SetOtch(string sotch)
        {
            if (string.IsNullOrEmpty(sotch))
                return false;
            else
            {
                otch = sotch;
                return true;                }
        }
        public int Countdays { get { return countdays; } }
```

```
public bool SetCountdays(int icountdays)
        {
            if (icountdays<1)
                return false;
            else
            {
                countdays = icountdays;
                return true;                }
        }
        public float Salary { get { return salary; } }
        public bool SetSalary(float fsalary)
        {
            if (fsalary <= 0)
                return false;
            else
            {
                salary = fsalary;
                return true;
            }
```

```
public float GetSalary()
        {
            return countdays * salary;
        }
        public bool IsThisSotrudnik(string sfamily,string
   sname,string sotch)
        {
            if (sfamily != Family)
                return false;
            if (sname != Name)
                return false;
            if (sotch != Otch)
                return false;
            return true;
        }
        public CSotrudnik()
        {
       countdays=1;
       salary=0.0;
        }
```

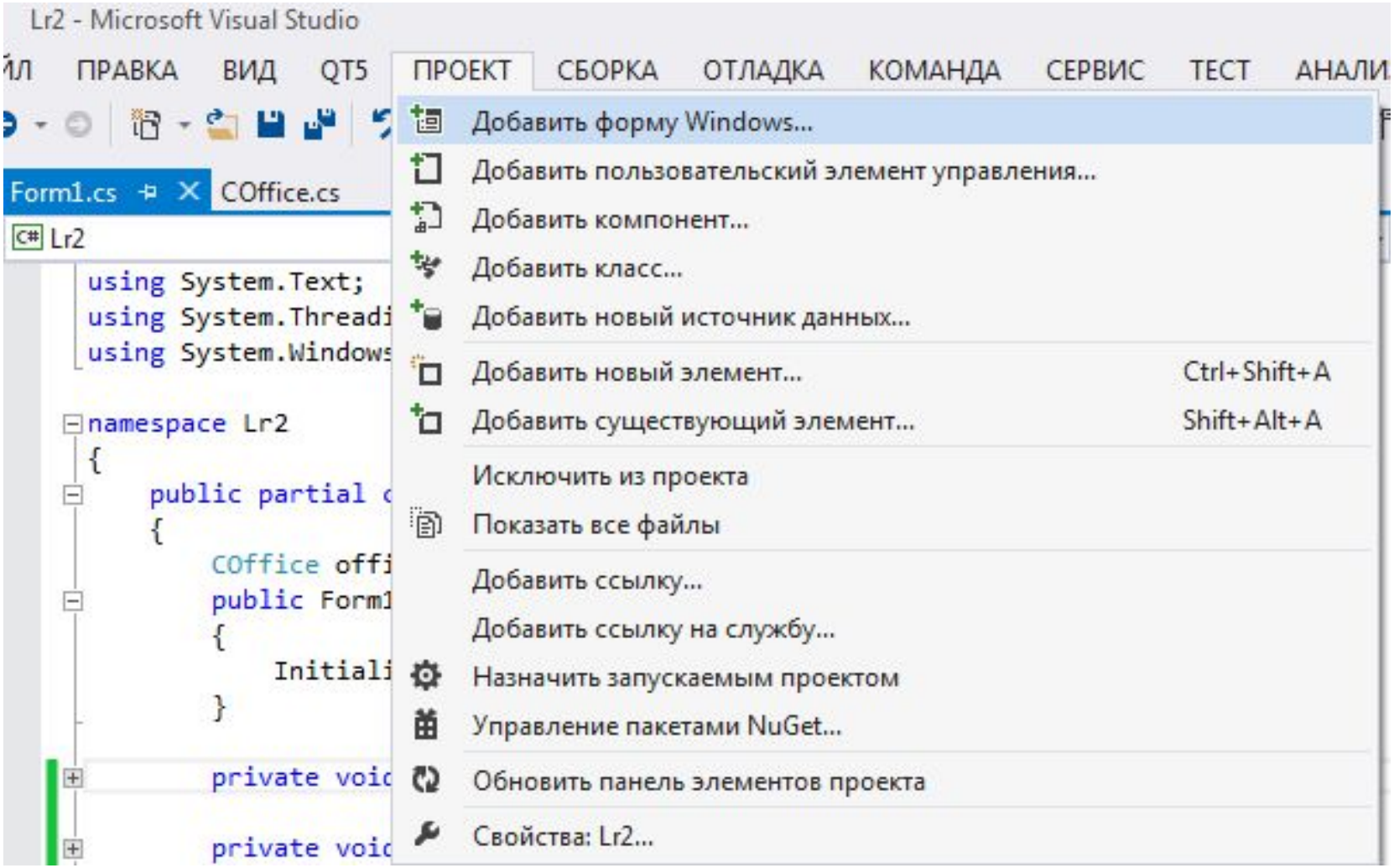
```
    public class Constants {
        public const int MaxCountSotr =
100;
    }
    enum ResAddSotr {OK,
ERR_COUNT,ERR_FAMILY,ERR_NAME,ERR_OTCH,ERR_
COUNT_DAY,ERR_SALARY_DAY,ERR_EXIST_SOTR};
    public struct DatSotr
    {
        public string sFam;
        public string sName;
        public string sOtch;
        public float Salary;
    }
```

```
class COffice
    {
        private int countsotr = 0;
        public int Countsotr {get {return countsotr;}}
        private CSotrudnik[] sotrudniks;
        public COffice()
        {
            sotrudniks = new CSotrudnik[Constants.MaxCountSotr];
        }
        public ResAddSotr AddSotr(string sfam, string sname, string sotch, int
countdays, float salaryday)
        {
            if (countsotr >= Constants.MaxCountSotr)
                return ResAddSotr.ERR_COUNT;
            if (FindSotr(sfam, sname, sotch) != -1)
                return ResAddSotr.ERR_EXIST_SOTR;
            sotrudniks[countsotr] = new CSotrudnik();
            if (!sotrudniks[countsotr].SetFamily(sfam))
                return ResAddSotr.ERR_FAMILY;
            if (!sotrudniks[countsotr].SetName(sname))
                return ResAddSotr.ERR_NAME;
            if (!sotrudniks[countsotr].SetOtch(sotch))
                return ResAddSotr.ERR_OTCH;
            if (!sotrudniks[countsotr].SetCountdays(countdays))
                return ResAddSotr.ERR_COUNT_DAY;
            if (!sotrudniks[countsotr].SetSalary(salaryday))
                return ResAddSotr.ERR_SALARY_DAY;
            countsotr++;
            return ResAddSotr.OK;
        }
```

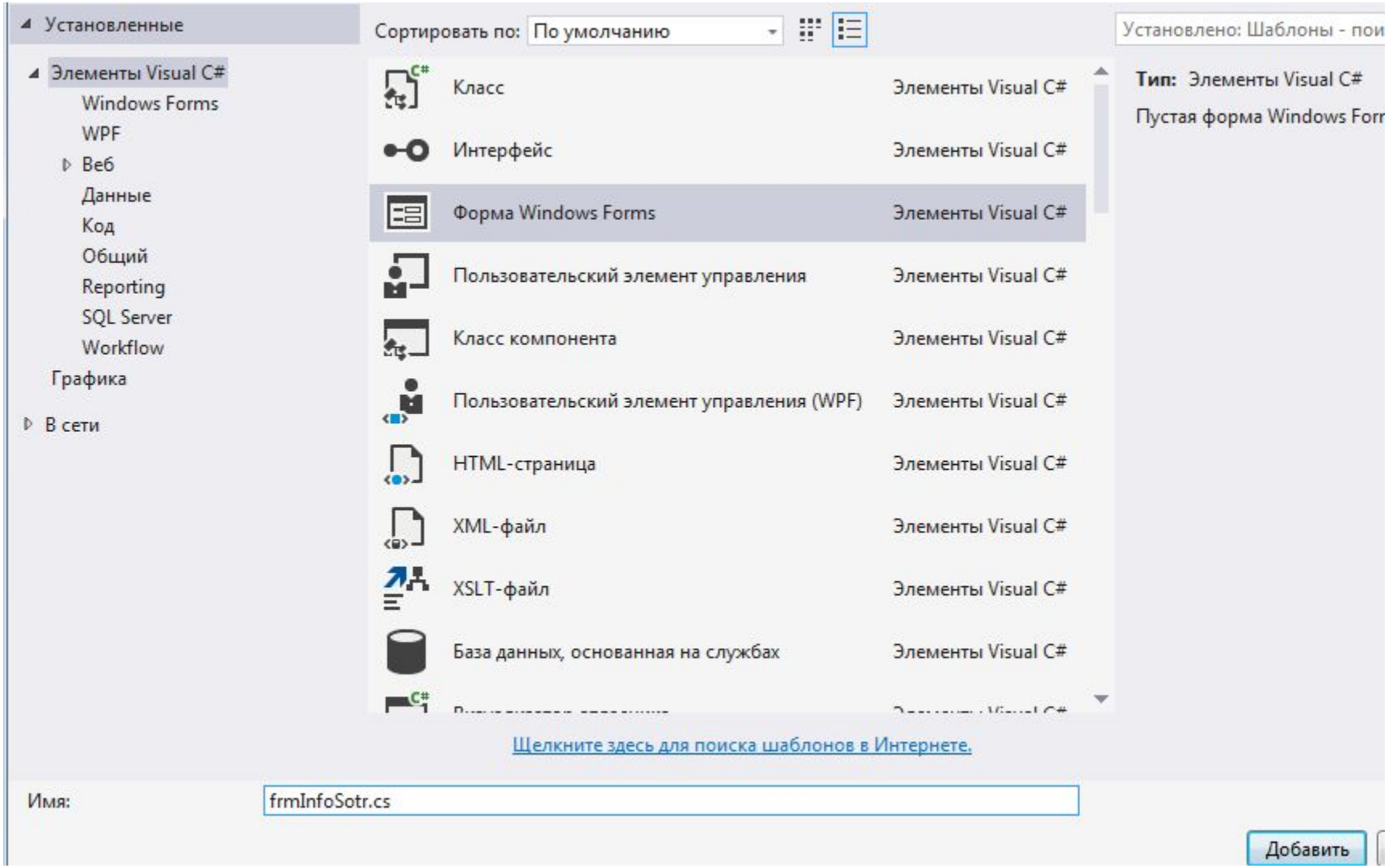
```
    public float GetAllCost()
        {
            float Summa = 0;
            for (int i = 0; i < Countsotr; i++)
            {
                Summa += sotrudniks[i].GetSalary();
            }
            return Summa;
        }
        public int FindSotr(string sfam, string sname, string sotch)
        {
            for (int i = 0; i < Countsotr; i++)
                if (sotrudniks[i].IsThisSotrudnik(sfam, sname, sotch))
                    return i;
                return -1;
        }
        public bool GetInfoSotr(int index,ref DatSotr dat)
        {
            if (index < 0 && index >= Countsotr)
                return false;
            dat.sFam = sotrudniks[index].Family;
            dat.sName = sotrudniks[index].Name;
            dat.sOtch = sotrudniks[index].Otch;
            dat.Salary = sotrudniks[index].GetSalary();
            return true;
        }

    }
```

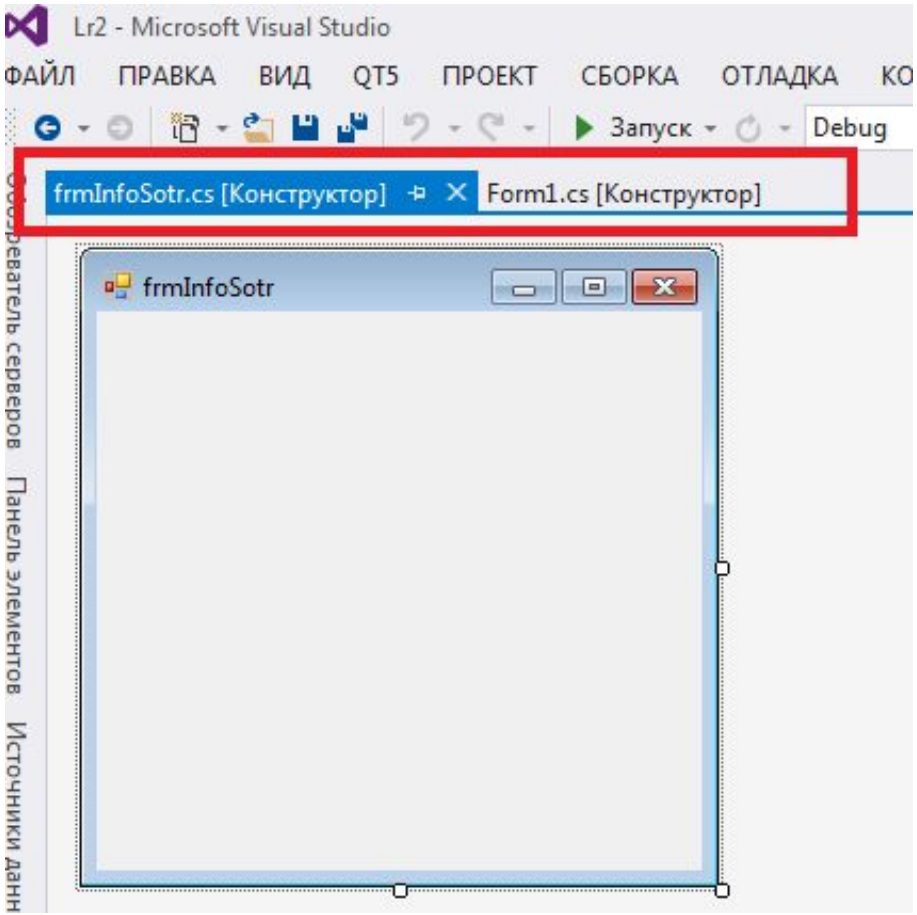
Добавление новой формы

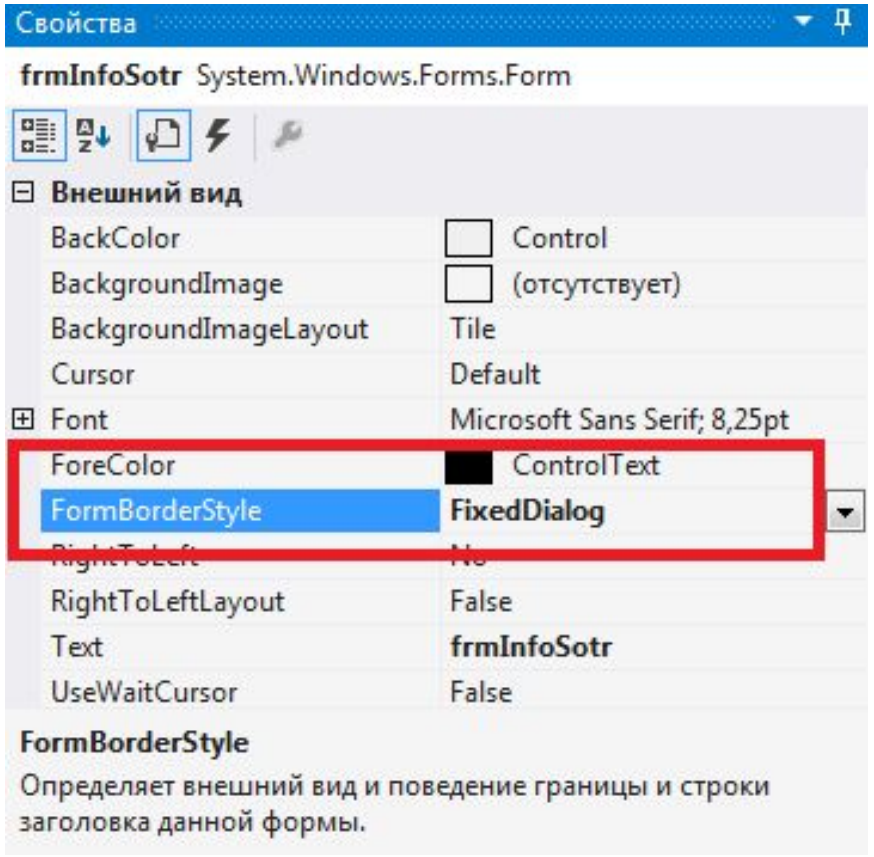


Добавление новой формы

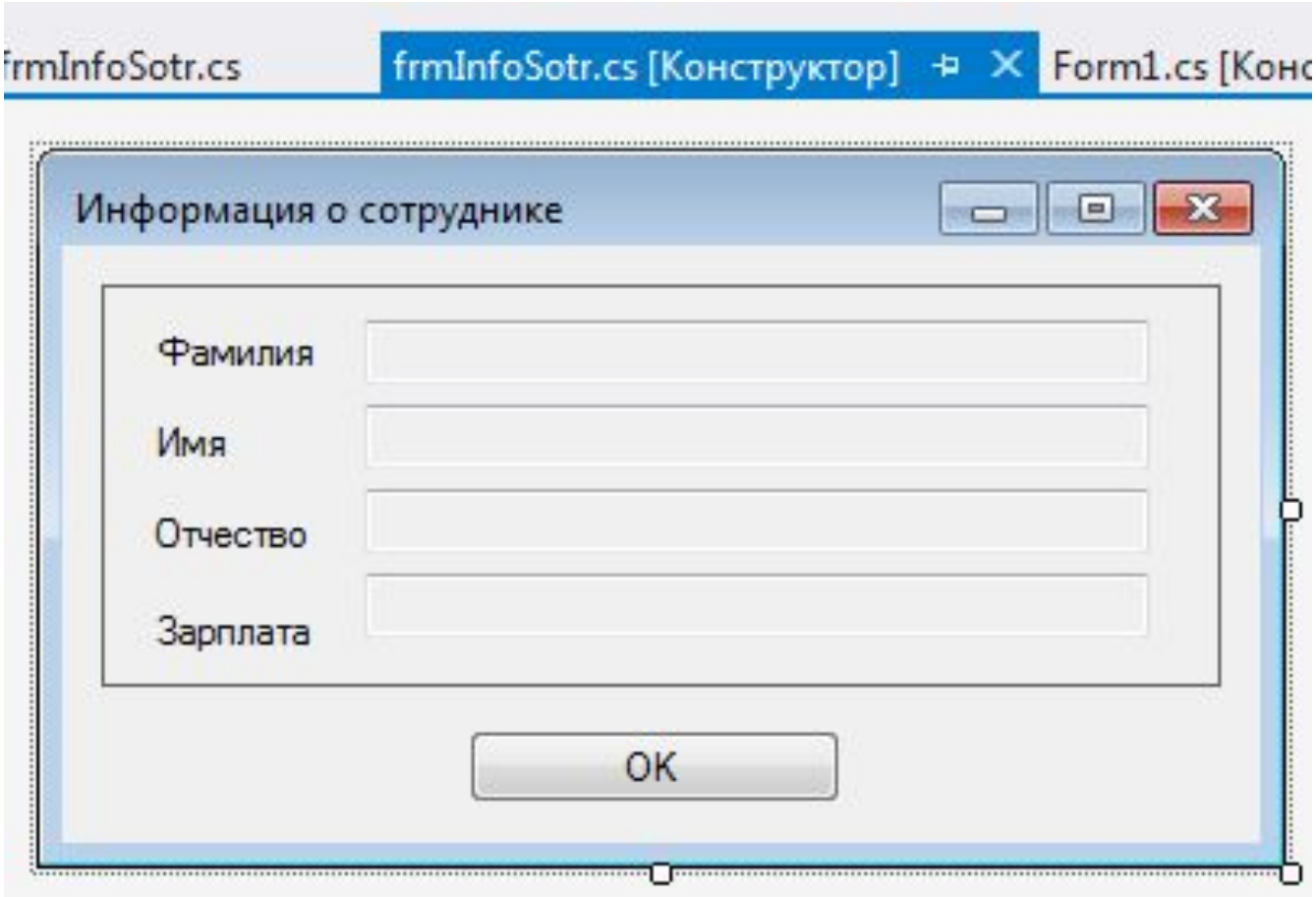


Добавление новой формы





Добавление новой формы



```
namespace Lr2
{
    public partial class frmInfoSotr : Form
    {
        public DatSotr dat = new DatSotr();
        public frmInfoSotr()
        {
            InitializeComponent();
        }
        private void btnClose_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void frmInfoSotr_Shown(object sender, EventArgs e)
        {
            txtFam.Text = dat.sFam;
            txtName.Text = dat.sName;
            txtOtch.Text = dat.sOtch;
            txtSalary.Text = String.Format("{0:F2} руб.", dat.Salary);
        }    }}
```

Добавление сотрудника

```
private void btnAdd_Click(object sender, EventArgs e)
       {
           ResAddSotr res =
    office.AddSotr(textFamily.Text,textName.Text,textOtch.Text,Convert.ToInt32(texCountDays.Text),Convert.ToSingle(txtS
    alary.Text));
           switch(res)
           {
               case ResAddSotr.ERR_COUNT:
                   MessageBox.Show("Введено максимальное количество сотрудников!");
                   break;
               case ResAddSotr.ERR_FAMILY:
                   MessageBox.Show("Не корректное значение фамилии!");
                   textFamily.Focus();
                   break;
               case ResAddSotr.ERR_NAME:
                   MessageBox.Show("Не корректное значение имени!");
                   textName.Focus();
                   break;
               case ResAddSotr.ERR_OTCH:
                   MessageBox.Show("Не корректное значение отчества!");
                   textOtch.Focus();
                   break;
               case ResAddSotr.ERR_COUNT_DAY:
                   MessageBox.Show("Не корректное значение отработанных дней!");
                   texCountDays.SelectAll();
                   texCountDays.Focus();
                   break;
               case ResAddSotr.ERR_SALARY_DAY:
                   MessageBox.Show("Не корректное значение заработной платы!");
                   txtSalary.SelectAll();
                   txtSalary.Focus();
                   break;
               case ResAddSotr.ERR_EXIST_SOTR:
                   MessageBox.Show("Сотрудник уже существует!!");
                   break;
               default:
                   txtCountSotr.Text = office.Countsotr.ToString();
                   break;
           }

       }
```

Добавление сотрудника

```
private void Form1_Shown(object sender,
 EventArgs e)
        {
            txtCountSotr.Text = "0";
            txtAllSumma.Text =
 String.Format("{0:F2} руб.", 0);
        }

        private void btnCalc_Click(object
 sender, EventArgs e)
        {
            txtAllSumma.Text =
 String.Format("{0:F2}
 руб.",office.GetAllCost());
        }
```

```
Namespace LR2
{
    public class Form1:Form
{
    Coffice office=new Coffice();
……….
……….

DateSotr dat_find=new DateSotr();
int
    n=office.FindSotr(textFamily.Text,textName.Text,textOtch.Tex
    t);
office.GetInfoSotr(n,ref  dat_find);
if (n!=-1)
{
frmInfoSotr frm=new frmInfoSotr();
frm.dat=dat_find;
frm.ShowModal();
}
```