

# Классы в С#

# Две роли классов

**Класс** – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью .



Класс

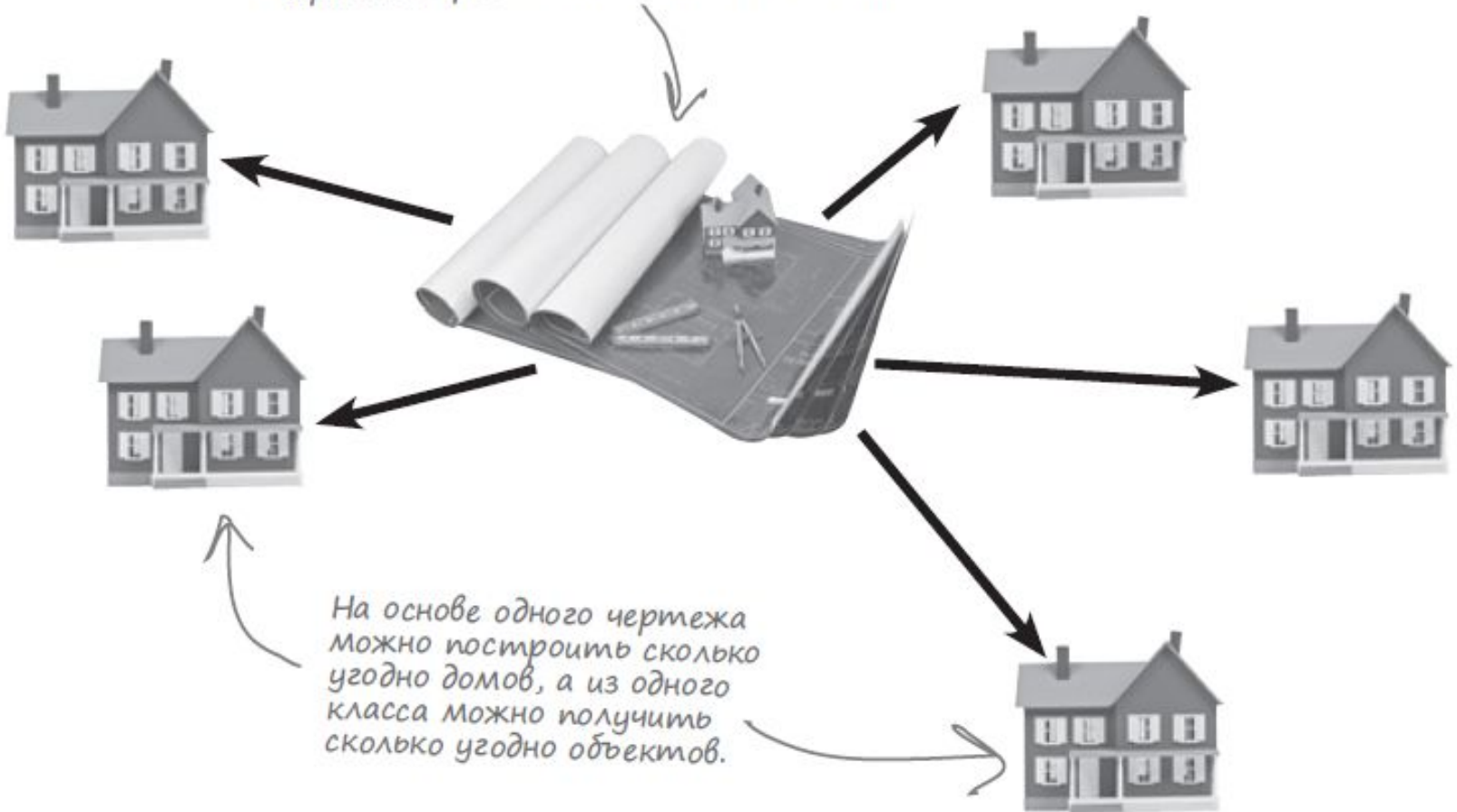
**Класс – это модуль**, архитектурная единица построения программной системы. Модульность построения – основное свойство программных систем. В ООП программная система, строящаяся по модульному принципу, состоит из классов, являющихся основным видом модуля.

**Класс – это тип данных**, задающий реализацию некоторой абстракции данных, характерной для задачи, в интересах которой создается программная система. С этих позиций классы, это не просто кирпичики, из которых строится система. Каждый кирпичик теперь имеет важную содержательную начинку.

Класс можно представить как копию объекта. Скажем, для построения пяти одинаковых домов в коттеджном поселке архитектору не нужно рисовать пять одинаковых чертежей. Для выполнения задачи вполне достаточно одного.

**Объект (экземпляр)** – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.

*Определяя класс, вы определяете и его методы, точно так же как чертеж определяет внешний вид дома.*



*На основе одного чертежа можно построить сколько угодно домов, а из одного класса можно получить сколько угодно объектов.*

# Синтаксис класса в C#

[атрибуты][модификаторы]class имя\_класса[:  
    список\_родителей]

{тело\_класса}

- Выделяют следующие модификаторы области видимости:
- **public** (открытый) – компонент доступен отовсюду, в том числе из других классов и из других сборок;
- **protected** (защищенный) – компонент доступен из класса, которому он принадлежит и из классов, производных от данного;
- **private** (закрытый) – компонент доступен только из класса, которому он принадлежит;
- **internal** (внутренний) – компонент доступен только из классов, принадлежащих сборке, в которой определен данный класс (в том числе и самому классу).

Обычно класс имеет атрибут доступа public, являющийся значением по умолчанию. Так что в простых случаях объявление класса выглядит так:

```
public class Rational {тело_класса}
```

В теле класса могут быть объявлены:

*константы;*

*поля;*

*конструкторы и деструкторы;*

*методы;*

*события;*

*делегаты;*

*классы (структуры, интерфейсы, перечисления).*

# Поля класса

**Поля класса** синтаксически являются обычными переменными (объектами) языка Их описание удовлетворяет обычным правилам объявления переменных. Содержательно поля задают представление той самой абстракции данных, которую реализует класс. Поля характеризуют **свойства объектов класса**.

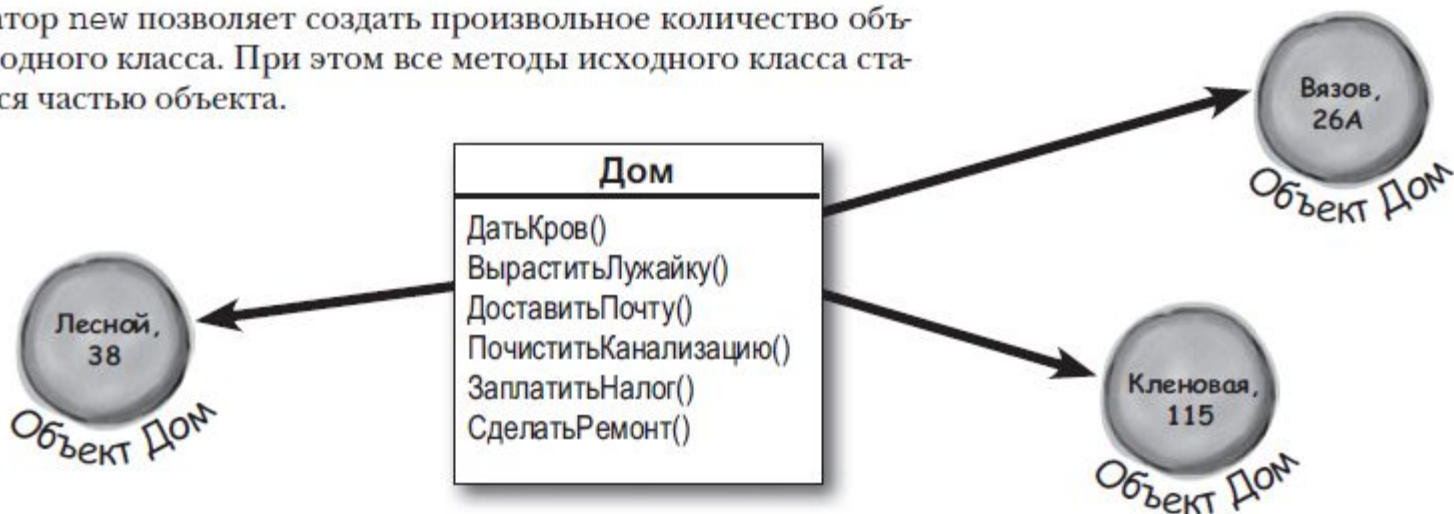
Каждое поле имеет **модификатор доступа**, принимающий одно из четырех значений: public, private, protected, internal. Атрибутом доступа по умолчанию является атрибут private.

# Методы класса

**Методы класса** синтаксически являются обычными процедурами и функциями языка. Их описание удовлетворяет обычным правилам объявления процедур и функций. Содержательно методы определяют ту самую абстракцию данных, которую реализует класс. Методы описывают операции, доступные над объектами класса. Два объекта одного класса имеют один и тот же набор методов.

## Объект берет методы из класса

Оператор `new` позволяет создать произвольное количество объектов одного класса. При этом все методы исходного класса становятся частью объекта.



*объекты, по порядку стройся!*

## КЛЮЧЕВЫЕ МОМЕНТЫ



- Классы состоят из методов, которые в свою очередь состоят из операторов. Осмысленный выбор методов позволяет получить удобный для работы класс.
- Некоторые методы могут **возвращать значение**. Тип этого значения нужно объявлять. Например, метод, объявленный как `public int`, возвращает целое число. Пример такого оператора: `return 37;`
- Метод, возвращающий значение, **обязан** включать в себя оператор `return`. Если в объявлении метода указано `public string`, значит, оператор `return` возвращает значение типа `string`.
- После оператора `return` программа возвращает управление оператору, вызывающему метод.
- Метод, при объявлении которого было указано `public void`, не возвращает значение. Но оператор `return` может использоваться для прерывания такого метода: `if (finishedEarly) { return; }.`



# Доступ к методам

- Каждый метод имеет **модификатор доступа**, принимающий одно из четырех значений: public, private, protected, internal. Атрибутом доступа по умолчанию является атрибут private.
- Как правило, у класса есть открытые методы, задающие интерфейс класса, и закрытые методы. **Интерфейс** – это лицо класса, именно он определяет, чем класс интересен своим клиентам, что он может делать, какие сервисы предоставляет клиентам. Закрытые методы составляют важную часть класса, позволяя клиентам класса не вникать во многие детали реализации класса. Эти методы клиентам класса недоступны, они о них могут ничего не знать, и, самое главное, изменения в закрытых методах класса никак не отражаются на клиентах класса при условии корректной работы открытых методов класса.

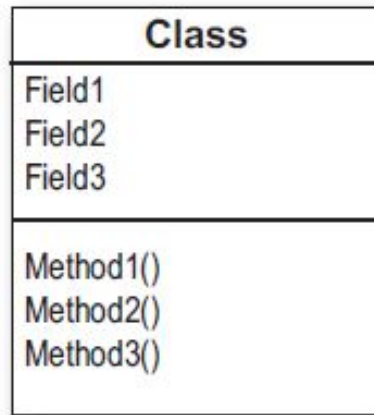
# Методы-свойства

**Методы**, называемые **свойствами** (Properties), представляют специальную синтаксическую конструкцию, предназначенную для обеспечения эффективной работы со свойствами. При работе со свойствами объекта (полями) часто нужно решить, какой модификатор доступа использовать, чтобы реализовать нужную стратегию доступа к полю класса. Перечислю пять наиболее употребительных стратегий:

- чтение, запись (Read, Write);
- чтение, запись при первом обращении (Read, Write-once);
- только чтение (Read-only);
- только запись (Write-only);
- ни чтения, ни записи (Not Read, Not Write).

Открытость свойств (атрибут `public`) позволяет реализовать только первую стратегию. В языке C# принято, как и в других объектных языках, свойства объявлять закрытыми, а нужную стратегию доступа организовать через методы. Для эффективности этого процесса и введены специальные методы-свойства.

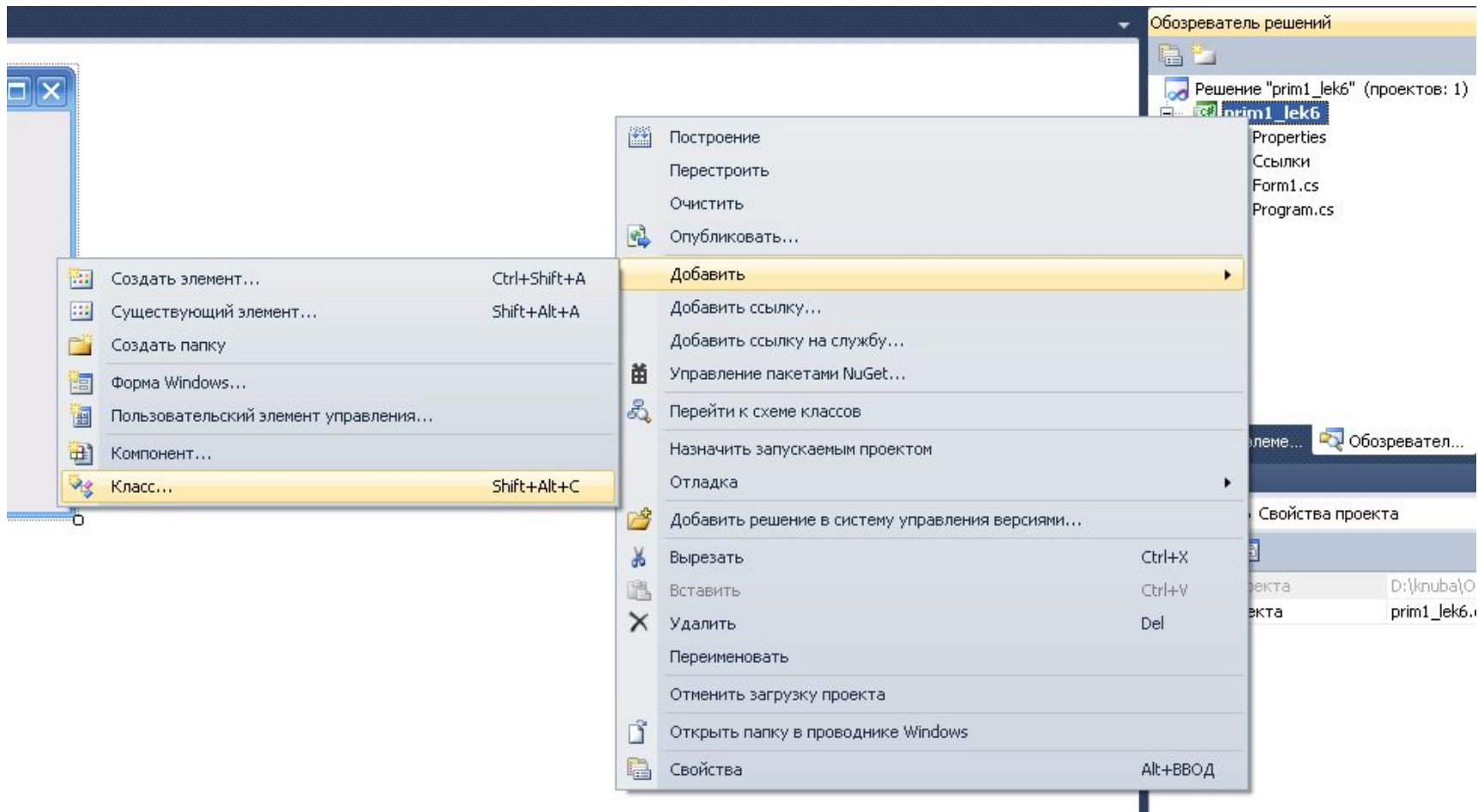
Сверху на диаграмме классов находится список полей. Экземпляры класса используют их для отслеживания своего состояния.



Эта линия отделяет поля от методов.

Метод — это то, что объект делает. Поле — это то, что объект знает.

**Пример.** Рассмотрим класс Person, у которого пять полей: fam, status, salary, age, health, характеризующих фамилию, статус, зарплату, возраст и здоровье персоны. Для каждого из этих полей может быть разумной своя стратегия доступа. Возраст доступен для чтения и записи, фамилию можно задать только один раз, статус можно только читать, зарплата недоступна для чтения, а здоровье закрыто для доступа, только специальные методы класса могут сообщать
















Добавление нового элемента - rgm1\_lek6

Установленные шаблоны

- Элементы Visual C#
  - Windows Forms
  - WPF
  - Веб
  - Данные
  - Код
  - Общий
  - Reporting
  - Workflow

Шаблоны в Интернете

Сортировать по: По умолчанию

	Класс	Элементы Visual C#
	Интерфейс	Элементы Visual C#
	Форма Windows Forms	Элементы Visual C#
	Пользовательский элемент управления	Элементы Visual C#
	Класс компонента	Элементы Visual C#
	Пользовательский элемент управления (WPF)	Элементы Visual C#
	Crystal Report	Элементы Visual C#
	HTML-страница	Элементы Visual C#
	RuleSet	Элементы Visual C#
	XML-файл	Элементы Visual C#
	XSLT-файл	Элементы Visual C#
	База данных, основанная на службах	Элементы Visual C#
	Визуализатор отладчика	Элементы Visual C#

Установленные шаблоны: поиск

Тип: Элементы Visual C#  
Пустое определение класса

Имя: Person

Добавить

Отмена

```

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

namespace prim1_lek6
{
    class Person
    {
        //поля (все закрыты)
        string fam = "", status = "", health = "";
        int age = 0, salary = 0;
        //методы - свойства
        //стратегия: Read,Write-once
        //(Чтение, запись при первом обращении)
        public string Fam
        {
            set { if (fam == "") fam = value; }
            get { return (fam); }
        }
        //стратегия: Read-only(Только чтение)
        public string Status
        {
            get { return (status); }
        }
    }
}

```

```

//стратегия: Read,Write (Чтение, запись)
public int Age
{
    set
    {
        age = value;
        if (age < 7) status = "ребенок";
        else if (age < 17) status = "школьник";
        else if (age < 22) status = "студент";
        else status = "служащий";
    }
    get { return (age); }
}
//стратегия: Write-only (Только запись)
public int Salary
{
    set { salary = value; }
}
}

```

Form1

ФИО

Возраст

Зарплата

Результат

Form1

ФИО

Возраст

Зарплата

Иванов И.И. 21 студент

Form1

ФИО

Возраст

Зарплата

Иванов И.И. 50 служащий

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace prim1_lek6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Person pers1 = new Person();
            pers1.Fam = textBox1.Text;
            pers1.Age = Convert.ToInt16(textBox2.Text);
            pers1.Salary = Convert.ToInt16(textBox3.Text);
            label4.Text = pers1.Fam + " " + pers1.Age.ToString() + " " + " " + pers1.Status;
        }
    }
}

```

Form1

ФИО

Возраст

Зарплата

Иванов И.И. 8 школьник

# Статические поля и методы класса

- Ранее говорилось, что когда конструктор класса создает новый объект, то в памяти создается структура данных с полями, определяемыми классом. Уточним теперь это описание. Не все поля отражаются в структуре объекта. У класса могут быть поля, связанные не с объектами, а с самим классом. Эти поля объявляются как статические с **модификатором static**. Статические поля доступны всем методам класса. Независимо от того, какой объект вызвал метод, используются одни и те же **статические поля**, позволяя методу использовать информацию созданную другими объектами класса. Статические поля представляют общий информационный пул для всех объектов классов, позволяя извлекать и создавать общую информацию. Например, у класса Person может быть статическое поле message, в котором каждый объект может оставить сообщение для других объектов класса.
- Аналогично полям у класса могут быть и **статические методы**, объявленные с модификатором static. Такие методы не используют информацию о свойствах конкретных объектов класса, они обрабатывают общую для класса информацию, хранящуюся в статических полях класса



# Конструкторы класса

- **Конструктор** неотъемлемый компонент класса. Нет классов без конструкторов. Конструктор представляет собой специальный метод класса, позволяющий создавать объекты класса. Одна из синтаксических особенностей этого метода в том, **что его имя должно совпадать с именем класса**. Если программист не определяет конструктор класса, то к классу автоматически добавляется **конструктор по умолчанию – конструктор без аргументов**. Заметьте, если программист сам создает один или несколько конструкторов, то автоматического добавления конструктора без аргументов не происходит.
- Как и когда происходит создание объектов? Чаще всего, при объявлении сущности в момент ее инициализации.
- Давайте обратимся к нашему последнему примеру и рассмотрим создание трех объектов класса Person:

```
Person pers1 = new Person(), pers2 = new Person();
```

```
Person pers3= new Person("Петрова");
```

# Деструкторы класса

Если задача создания объектов полностью возлагается на программиста, то задача удаления объектов, после того, как они стали не нужными, в Visual Studio .Net снята с программиста и возложена на соответствующий инструментарий – сборщик мусора. В классическом варианте языка C++ **деструктор** также необходим классу, как и конструктор. В языке C# у класса может быть деструктор, но он не занимается удалением объектов и не вызывается нормальным образом в ходе выполнения программы. Также как и статический конструктор, деструктор класса, если он есть, вызывается автоматически в процессе сборки мусора. Его роль в освобождении ресурсов, например файлов, открытых объектом. Деструктор C# фактически является финализатором (finalizer), с которыми мы еще встретимся при обсуждении исключительных ситуаций. Приведу формальное описание деструктора класса Person:

```
~Person()  
{  
    //Код деструктора  
}
```

Имя деструктора строится из имени класса с предшествующим ему символом ~ (тильда). Как и у статического конструктора, у деструктора не указывается модификатор доступа.