Оглавление

[Создание конструкторов 3](#_Toc161917815)

[Цепочка вызовов конструкторов 3](#_Toc161917816)

[Первичные конструкторы (начиная с версии C#12) 3](#_Toc161917817)

[Инициализаторы объектов 3](#_Toc161917818)

[Деконструкторы 3](#_Toc161917819)

[Структуры 4](#_Toc161917820)

[Инициализация с помощью конструктора 4](#_Toc161917821)

[Цепочка конструкторов 4](#_Toc161917822)

[Первичный конструктор 5](#_Toc161917823)

[Инициализатор структуры 5](#_Toc161917824)

[Копирование структуры с помощью with 5](#_Toc161917825)

[Модификаторы доступа 5](#_Toc161917826)

[Свойства 5](#_Toc161917827)

[Определение свойств 5](#_Toc161917828)

[Свойства только для чтения и записи 6](#_Toc161917829)

[Модификаторы доступа 6](#_Toc161917830)

[Автоматические свойства 6](#_Toc161917831)

[Блок init (начиная с версии C# 9.0) 6](#_Toc161917832)

[Сокращенная запись свойств 7](#_Toc161917833)

[Модификатор required (добавлен в C# 11) 7](#_Toc161917834)

[Статические члены и модификатор static 7](#_Toc161917835)

[Статические поля 7](#_Toc161917836)

[Статические свойства 7](#_Toc161917837)

[Статические методы 8](#_Toc161917838)

[Статический конструктор 8](#_Toc161917839)

[Статические классы 8](#_Toc161917840)

[Константы, поля и структуры для чтения 8](#_Toc161917841)

[Константы класса 8](#_Toc161917842)

[Поля для чтения и модификатор readonly 8](#_Toc161917843)

[Сравнение констант 8](#_Toc161917844)

[Структуры для чтения 9](#_Toc161917845)

[Null и ссылочные типы 9](#_Toc161917846)

[Оператор ! (null-forgiving operator) 9](#_Toc161917847)

[Null и значимые типы 9](#_Toc161917848)

[Проверка на null, операторы ?. и ?? 9](#_Toc161917849)

[Оператор ?? 10](#_Toc161917850)

[Оператор условного null ( ?. ) 10](#_Toc161917851)

[Псевдонимы типов и статический импорт 10](#_Toc161917852)

[Псевдонимы 10](#_Toc161917853)

[Статический импорт 10](#_Toc161917854)

# Создание конструкторов

class Person

**{**

**public** string name**;**

**public** Person**()** **{** name **=** "Неизвестно"**}** // 1 конструктор

**public** Person**(**string name**)** **{** **this.**name **=** name **}** // 2 конструктор

**}**

Person tom **=** **new** **();** // аналогично new Person();

Person bob **=** **new** **(**"Bob"**);** // аналогично new Person("Bob");

## Цепочка вызовов конструкторов

class Person

**{**

**public** string name**;**

**public** Person**()** **:** **this(**"Неизвестно"**)** **{** **}** // первый конструктор

**public** Person**(**string name**)** **{** **this.**name **=** name **}** // второй конструктор

**}**

## Первичные конструкторы (начиная с версии C#12)

**public** class Person**(**string name**,** int age**)**

**{**

**public** Person**(**string name**)** **:** **this(**name**,** 18**)** **{** **}**

**}**

Person tom **=** **new** **(**"Tom"**,** 34**);**

Person bob **=** **new** **(**"Bob"**);**

## Инициализаторы объектов

class Person

**{**

**public** string name**;**

**public** Company company**;**

**public** Person**()**

**{**

name **=** "Undefined"**;**

company **=** **new** Company**();**

**}**

**}**

class Company

**{**

**public** string title **=** "Unknown"**;**

**}**

Person tom **=** **new** Person**{** name **=** "Tom"**,** company **=** **{** title **=** "Microsoft"**}** **};**

## Деконструкторы

class Person

**{**

string name**;**

int age**;**

**public** Person**(**string name**,** int age**)**

**{**

**this.**name **=** name**;**

**this.**age **=** age**;**

**}**

**public** void Deconstruct**(out** string personName**,** **out** int personAge**)**

**{**

personName **=** name**;**

personAge **=** age**;**

**}**

**}**

Person person **=** **new** Person**(**"Tom"**,** 33**);**

**(**string name**,** int age**)** **=** person**;**

// То же самое, что и:

// string name; int age;

// person.Deconstruct(out name, out age);

// Если нужна только одна переменная, можно так:

// (\_, int age) = person;

Console**.**WriteLine**(**name**);** // Tom

Console**.**WriteLine**(**age**);** // 33

# Структуры

## Инициализация с помощью конструктора

struct Person

**{**

// элементы структуры:

// поля и методы

int age **=** 10**;** // инициализация полей значениями по умолчанию - доступна с C#10

**public** Person**()** // начиная с C#10 мы можем определить свой конструктор без параметров

**{**

age **=** 20**;**

**}**

**}**

Person tom **=** **new** Person**();** // вызов конструктора

Person bob **=** **new();**

## Цепочка конструкторов

struct Person

**{**

**public** int age**;**

**public** Person**()** **:** **this(**10**)**

**{** **}**

**public** Person**(**string name**)** **:** **this(**name**,** 1**)**

**{** **}**

**public** Person**(**int age**)**

**{**

**this.**age **=** age**;**

**}**

**}**

var tom **=** **new** Person**(**20**);**

## Первичный конструктор

**public** struct Person**(**string name**,** int age**)**

**{**

**public** Person**(**string name**)** **:** **this(**name**,** 18**)** **{** **}**

**}**

## Инициализатор структуры

struct Person

**{**

**public** string name**;**

**public** int age**;**

**}**

Person tom **=** **new** Person **{** name **=** "Tom"**,** age **=** 22 **};**

## Копирование структуры с помощью with

Person tom **=** **new** Person **{** name **=** "Tom"**,** age **=** 22 **};**

Person bob **=** tom with **{** name **=** "Bob" **};**

// объект bob получает все значения объекта tom,

// а затем после оператора with в фигурных скобках указывается поля со значениями,

// которые мы хотим изменить.

# Модификаторы доступа

**private:** закрытый или приватный компонент класса или структуры**.** Приватный компонент доступен только в рамках своего класса или структуры**.**

**private** **protected:** компонент класса доступен из любого места в своем классе или в производных классах**,** которые определены в той же сборке**.**

**file:** добавлен в версии C# 11 и применяется к типам**,** например**,** классам и структурам**.** Класс или структура с такми модификатором доступны только из текущего файла кода**.**

**protected:** такой компонент класса доступен из любого места в своем классе или в производных классах**.** При этом производные классы могут располагаться в других сборках**.**

**internal:** компоненты класса или структуры доступен из любого места кода в той же сборке**,** однако он недоступен для других программ и сборок**.**

**protected** **internal:** совмещает функционал двух модификаторов **protected** и **internal.** Такой компонент класса доступен из любого места в текущей сборке и из производных классов**,** которые могут располагаться в других сборках**.**

**public:** публичный**,** общедоступный компонент класса или структуры**.** Такой компонент доступен из любого места в коде**,** а также из других программ и сборок**.**

# Свойства

## Определение свойств

**private** string name **=** "Undefined"**;**

**public** string Name

**{**

**get**

**{**

**return** name**;** // возвращаем значение свойства

**}**

**set**

**{**

name **=** **value;** // устанавливаем новое значение свойства

**}**

**}**

person**.**Name **=** "Tom"**;**

## Свойства только для чтения и записи

// свойство только для записи

**public** int Age

**{**

**set** **{** age **=** **value;** **}**

**}**

// свойство только для чтения

**public** string Name

**{**

**get** **{** **return** name**;** **}**

**}**

## Модификаторы доступа

**public** string Name

**{**

**get** **{** **return** name**;** **}**

**private** **set** **{** name **=** **value;**

**}**

*Ограничения****:***

**-** Модификатор для блока **set** или **get** можно установить**,** если свойство имеет оба блока **(**и **set,** и **get);**

**-** Только один блок **set** или **get** может иметь модификатор доступа**,** но не оба сразу**;**

**-** Модификатор доступа блока **set** или **get** должен быть более ограничивающим**,** чем модификатор доступа свойства**.**

## Автоматические свойства

string Name **{** **get;** **set;** **}**

int Age **{** **get;** **set;** **}** **=** 37**;** // значение по умолчанию

string City **{** **private** **set;** **get;}** // модификатор доступа

## Блок init (начиная с версии C# 9.0)

После инициализации значений свойств их значения изменить нельзя **-** они доступны только для чтения**.**

*1 способ****:***

string Name **{** **get;** init**;** **}** **=** "Undefined"**;** // изменить нельзя

*2 способ****:***

Person**(**string name**)** **=>** Name **=** name**;** // инициализация в конструкторе

string Name **{** **get;** init**;** **}**

*3 способ****:***

class Person

**{**

string Name **{** **get;** init**;** **}** **=** ""**;**

**}**

Person person **=** **new()** **{** Name **=** "Bob"**};**

*4 способ****:***

string name **=** ""**;**

string Name // свойство Name управляет полем для чтения name

**{**

**get** **{** **return** name**;** **}**

init

**{**

name **=** **value;**

**}**

**}**

## Сокращенная запись свойств

string name**;**

string Name

**{**

**get** **=>** name**;**

**set** **=>** name **=** **value;**

**}**

string name**;**

**public** string Name **=>** name**;** // эквивалентно public string Name { get { return name; } }

## Модификатор required (добавлен в C# 11)

Модификатор required указывает**,** что поле или свойство с этим модификатором обязательно должно быть инициализировано**.**

Person**(**string name**)**

**{**

Name **=** name**;**

**}**

required string Name **{** **get;** **set;** **}** // инициализировано в конструкторе

required int Age **{** **get;** **set;** **}** **=** 22**;**

# Статические члены и модификатор static

## Статические поля

Статические поля хранят состояние всего класса **/** структуры**.**

## Статические свойства

static int age **=** 65**;**

static int Age

**{**

**get** **{** **return** age**;** **}**

**set** **{** age **=** **value;** **}**

**}**

## Статические методы

Статические методы могут обращаться только к статическим членам класса**.**

## Статический конструктор

**-** Статические конструкторы не должны иметь модификатор доступа и не принимают параметров**;**

**-** Как и в статических методах**,** в статических конструкторах нельзя использовать ключевое слово **this** для ссылки на текущий объект класса и можно обращаться только к статическим членам класса**;**

**-** Статические конструкторы нельзя вызвать в программе вручную**.** Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам **(**если таковые имеются**).**

## Статические классы

Статические классы объявляются с модификатором static и могут содержать только статические поля**,** свойства и методы**.**

# Константы, поля и структуры для чтения

## Константы класса

Значение константы устанавливается один раз непосредственно при её объявлении и впоследствии не может быть изменено**.**

Константы хранят некоторые данные**,** которые относятся не к одному объекту**,** а ко всему классу в целом**.**

Для обращения к константам применяется не имя объекта**,** а имя класса**:**

class Person

**{**

const string type **=** "Person"**;**

**}**

Console**.**WriteLine**(**Person**.**type**);** // Person

## Поля для чтения и модификатор readonly

Таким полям можно присвоить значение либо при непосредственно при их объявлении**,** либо в конструкторе**.**

Их значение нельзя изменить**.**

**readonly** string name **=** "Tom"**;**

## Сравнение констант

**-** const должны быть определены во время компиляции**,** а **readonly** могут быть определены во время выполнения программы в конструкторе**.**

**-** const в отличие от **readonly** не могут использовать модификатор static**,** так как уже неявно являются статическими**.**

## Структуры для чтения

Все их поля должны быть также полями для чтения**:**

**readonly** struct Person

**{**

**readonly** int age **=** 5**;**

**readonly** string Name **{** **get;** **}** **=** "Tom"**;** // указывать readonly необязательно

**}**

# Null и ссылочные типы

#nullable disable

string text **=** **null;** // здесь nullable-контекст не действует

#nullable restore

string**?** name **=** **null;** // здесь nullable-контекст снова действует

## Оператор ! (null-forgiving operator)

string**?** name **=** "Tom"

Console**.**WriteLine**(**name**!);**

# Null и значимые типы

int**?** val **=** **null;** // nullable int тип

// то же самое, что и:

Nullable**<**int**>** val **=** **null;**

Структура Nullable**<**T**>** имеет два свойства**:**

**-** Value **-** значение объекта**;**

**-** HasValue**:** возвращает **true,** если объект хранит некоторое значение**,** и **false,** если объект равен **null.**

**...** и метод**:**

**-** GetValueOrDefault**():** возвращает значение переменной**/**параметра**,** если они не равны **null.** Если они равны **null,** то возвращается значение по умолчанию**.** Значение по умолчанию можно передать в метод**.**

int**?** number **=** **null;** // если значения нет, метод возвращает значение по умолчанию

Console**.**WriteLine**(**number**.**GetValueOrDefault**());** // 0 - значение по умолчанию для числовых типов

Console**.**WriteLine**(**number**.**GetValueOrDefault**(**10**));** // 10

В арифметических операциях**,** если один из операндов равен **null,** то результатом операции также будет **null:**

int**?** x **=** **null;**

int**?** w **=** x **+** 7**;** // w = null

В операциях сравнения **>,** **<,** **>=** и **<=,** если хотя бы один из операндов равен **null,** то возвращается **false** **(**кроме операции **!=).**

# Проверка на null, операторы ?. и ??

С помощью оператора **is** мы можем проверить значение объекта**:**

string name **=** **null;**

**if** **(**name **is** **null)** **...** // объект is значение

// Также можно проверить на соответствие типу:

// name is string

С помощью **is** not можно проверить отсутствие значения**:**

**if** **(**name **is** not **null)** **...**

## Оператор ??

Оператор **??** возвращает левый операнд**,** если этот операнд не равен **null.**

Иначе возвращается правый операнд**.**

При этом левый операнд должен принимать **null.**

int**?** id **=** 200**;**

int personid **=** id **??** 0**;** // равно 200, так как id не равен null

## Оператор условного null ( ?. )

Person**?.**name

# Псевдонимы типов и статический импорт

## Псевдонимы

**using** printer **=** System**.**Console**;**

printer**.**WriteLine**(**"Hello, World"**);**

**using** User **=** Person**;**

User tom **=** **new** User**(**"Tom"**);**

## Статический импорт

Выражение **using** static подключает в программу все статические методы и свойства**,** а также константы**.**

И после этого мы можем не указывать название класса при вызове метода**.**

**using** static System**.**Console**;**

WriteLine**(**"Hello from C# 8.0"**);**