Оглавление

[Наследование 2](#_Toc162811669)

[Ключевое слово base 2](#_Toc162811670)

[Преобразование типов 2](#_Toc162811671)

[Восоходящее преобразование. Upcasting 2](#_Toc162811672)

[Нисходящее преобразование. Downcasting 2](#_Toc162811673)

[Способы преобразований 3](#_Toc162811674)

[Виртуальные методы и свойства 3](#_Toc162811675)

[Ключевое слово base 3](#_Toc162811676)

[Переопределение свойств 3](#_Toc162811677)

[Запрет переопределения методов 4](#_Toc162811678)

[Скрытие методов и свойств 4](#_Toc162811679)

[Различие переопределения и скрытия методов 4](#_Toc162811680)

[Переопределение 4](#_Toc162811681)

[Скрытие 5](#_Toc162811682)

[Абстрактные классы и члены классов 5](#_Toc162811683)

[Класс System.Object и его методы 5](#_Toc162811684)

[Обобщения (generics) 5](#_Toc162811685)

[Статические поля обобщенных классов 6](#_Toc162811686)

[Обобщенные методы 6](#_Toc162811687)

[Ограничения обобщений 6](#_Toc162811688)

[Ограничения методов 6](#_Toc162811689)

[Ограничения обобщений в типах 6](#_Toc162811690)

[Типы ограничений и стандартные ограничения 6](#_Toc162811691)

[Использование нескольких универсальных параметров 7](#_Toc162811692)

[Наследование обобщенных типов 7](#_Toc162811693)

# Наследование

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

class Employee **:** Person **{** **...** **}**

Person person **=** **new** Person **{** Name **=** "Tom" **};**

Person employee **=** **new** Employee **{** Name **=** "Sam" **};**

*Ограничения****:***

**-** Класс может наследоваться только от одного класса**.**

**-** Тип доступа к производному классу должен быть таким же**,** как и у базового класса**,** или более строгим**.** Если базовый и производный класс находятся в разных сборках **(**проектах**),** то производый класс может наследовать только от класса **public.**

**-** **sealed** класс не допускает создание наследников**:** **sealed** class Admin **{}**

**-** Нельзя унаследовать класс от статического класса**.**

## Ключевое слово base

Если в базовом классе не определен конструктор по умолчанию без параметров**,** а только конструкторы с параметрами**,** то в производном классе мы обязательно должны вызвать один из этих конструкторов через ключевое слово **base.**

При вызове конструктора класса сначала отрабатываются конструкторы базовых классов и только затем конструкторы производных**.**

class Person

**{**

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

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

**{**

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

**}**

**}**

class Employee **:** Person

**{**

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

**public** Employee**(**string name**,** string company**)** **:** **base(**name**)**

**{**

Company **=** company**;**

**}**

**}**

# Преобразование типов

## Восоходящее преобразование. Upcasting

Employee employee **=** **new** Employee**(**"Tom"**,** "Microsoft"**);**

Person person **=** employee**;** // преобразование от Employee к Person

employee и person будут указывать на один и тот же объект в памяти**,** но переменной person будет доступна только та часть**,** которая представляет функционал типа Person**.**

## Нисходящее преобразование. Downcasting

Person person **=** **new** Person**();**

Employee employee2 **=** **(**Employee**)**person**;** // преобразование от Person к Employee

## Способы преобразований

Person person **=** **new** Person**();**

Employee**?** employee **=** person **as** Employee**;** // В случае неудачного преобразования выражение будет содержать значение null

**if** **(**person **is** Employee employee**)**

**{**

Console**.**WriteLine**(**employee**.**Company**);**

**}**

Выражение **if** **(**person **is** Employee employee**)** проверяет**,** является ли переменная person объектом типа Employee**.** Если person является объектом Employee**,** то автоматически преобразует значение переменной person в тип Employee и преобразованное значение сохраняет в переменную employee**.**

**if** **(**person **is** Employee**)** // без преобразования типа Person в Employee

**{**

Console**.**WriteLine**(**"Представляет тип Employee"**);**

**}**

# Виртуальные методы и свойства

Методы в родительском классе**,** помеченные модификатором **virtual,** доступны для переопределения**.**

В классе**-**наследнике мы можем переопределить его**,** пометив модификатором **override.**

class Person

**{**

**virtual** void Method**(){}**

**}**

class Employee **:** Person

**{**

**virtual** void Method**(){}**

**}**

Ограничения**:**

**-** Виртуальный и переопределенный методы должны иметь один и тот же модификатор доступа**.**

**-** Нельзя переопределить или объявить виртуальным статический метод**.**

## Ключевое слово base

class Employee **:** Person

**{**

Employee**(** **...** **)** **:base(** **...** **)** // из базового класса можно вызвать и конструктор

**{**

**...**

**}**

**override** void Method**()**

**{**

**base.**Print**();** // и метод с помощью ключевого слова base

**...**

**}**

**}**

## Переопределение свойств

Можно переопределять свойства**:**

class Person

**{**

**private** int age**;**

**public** **virtual** int Age **{** **get;** **set;** **}**

**}**

class Employee **:** Person

**{**

**public** **override** Age

**{**

**get** **=>** **base.**Age**;**

**set** **{** **if** **(value** **>** 18**)** **base.**Age **=** **value;** **}**

**}**

**}**

## Запрет переопределения методов

**override** **sealed** запрещает переопределение методов и свойств**.** Оно работает только в уже переопределенных методах в классах**-**наследниках**,** т**.**к**.** в родительском классе достаточно не ставить модификатор **virtual:**

class Employee **:** Person

**{**

**override** **sealed** void Method**()** **{** **...** **}**

**}**

# Скрытие методов и свойств

class Person

**{**

**public** int age **=** 1**;**

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

**public** void Print**()** **{}** // не виртуальный метод - нельзя переопределить

**}**

class Employee **:** Person

**{**

**public** **new** int age **=** 18**;** // скрываем родительскую переменную или константу своей

**public** **new** string Name **{** **get** **=>** **...** **;** **set** **=>** **base.**Name **=** **value;** **}** // скрываем родительское свойство своим

**public** **new** void Print**()** **{}** // скрываем родительский метод своим

**}**

# Различие переопределения и скрытия методов

## Переопределение

class Person

**{**

**virtual** void Print**()** **{** Console**.**WriteLine**(**"Person"**);** **}**

**}**

class Employee **:** Person

**{**

**override** void Print**()** **{** Console**.**WriteLine**(**"Employee"**);** **}**

**}**

Person person **=** **new** Employee**();**

tom**.**Print**();** // Employee

Через Таблицу Виртуальных Методов вызывается метод класса Employee**,** хотя объект типа Person.

## Скрытие

class Person

**{**

void Print**()** **{** Console**.**WriteLine**(**"Person"**);** **}**

**}**

class Employee **:** Person

**{**

**new** void Print**()** **{** Console**.**WriteLine**(**"Employee"**);** **}**

**}**

Person person **=** **new** Employee**();**

tom**.**Print**();** // Person

Вызывается метод класса Person объекта типа Person.

# Абстрактные классы и члены классов

**abstract** class Transport

**{**

int age **=** 0**;** // может иметь переменные

int Age **{** **get** **=>** age**;** **set** **=>** age **=** **value;** **}** // может иметь свойства

void Move**()** **{** **...** реализация **}** // может иметь методы с реализацией

**abstract** string Name **{** **get;** **set;** **}** // абстрактные свойства

**abstract** int Speed **{** **get;** **set;** **}**

**abstract** void Stop**();** // абстрактный метод

**}**

class Car **:** Transport

**{**

string name**;**

**override** string Name **{** **get** **=>** name**;** **set** **=>** name **=** **value;** **}** // реализации свойств

**override** int Speed **{** **get;** **set;** **}**

**override** void Stop**()** **{** **...** реализация **}** // реализация метода

**}**

# Класс System.Object и его методы

У класса Object есть основные методы**,** которые можно переопределить с помощью ключевого слова **override:**

**override** string**?** ToString**()**

**override** int GetHashCode**()**

**override** bool Equals**(object?** obj**)**

**...** и метод который не переопределяется**:**

Type GetType**()** // проверить тип можно с помощью конструкций:

// if (person.GetType() == typeof(Person)

// if (person is Person)

# Обобщения (generics)

class Person**<**T**>** **{...}**

class Person**<**T**,** K**>** **{...}**

## Статические поля обобщенных классов

class Person**<**T**>**

**{**

static T**?** Id**;**

**}**

Person**<**int**>** tom **=** **new** Person**<**int**>();**

tom**.**Id **=** 1**;**

Person**<**string**>** bob **=** **new** Person**<**string**>();**

bob**.**Id **=** "Bob"**;**

Для класса Person**<**int**>** и для класса Person**<**string**>** в куче будет создана своя переменная Id**.**

## Обобщенные методы

void Method**<**T**>(**T x**)** **{...}**

# Ограничения обобщений

## Ограничения методов

void SendMessage**<**T**>(**T message**)** **where** T**:** Message **{...}**

// через универсальный параметр T будут передаваться объекты класса Message и производных классов

## Ограничения обобщений в типах

class Messenger**<**T**>** **where** T **:** Message **{...}**

## Типы ограничений и стандартные ограничения

В качестве ограничений мы можем использовать следующие типы**:**

**-** Классы **(where** T **:** Message**)**

**-** Интерфейсы **(where** T **:** IMessage**)**

**-** class **-** универсальный параметр должен представлять класс **(where** T **:** class**)**

**-** struct **-** универсальный параметр должен представлять структуру **(where** T **:** struct**)**

**-** **new()** **-** универсальный параметр должен представлять тип**,** который имеет общедоступный **(public)** конструктор без параметров **(where** T **:** **new())**

Порядок следования нескольких ограничений**:**

1. Название класса**,** class**,** struct**.** Причем мы можем одновременно определить только одно из этих ограничений

2. Название интерфейса

3. **new()**

class Messenger**<**T**>** **where** T**:** Message**,** **new()**

## Использование нескольких универсальных параметров

class Messenger**<**T**,** P**>**

**where** T **:** Message

**where** P**:** Person

**{...}**

# Наследование обобщенных типов

Пример 1**:**

class Person**<**T**>** **{...}**

class Employee**<**T**>** **:** Person**<**T**>** **{...}**

Пример 2**:**

class Person**<**T**>** **{...}**

class Employee **:** Person**<**string**>** **{...}**

Пример 3**:**

class Person**<**T**>** **{...}**

class Employee**<**T**>** **:** Person**<**int**>** **{...}**

Пример 4**:**

class Person**<**T**>** **{...}**

class Employee**<**T**,** K**>** **:** Person**<**T**>** **where** K **:** struct **{...}**

Пример 5**:**

class Person**<**T**>** **where** T **:** class **{...}**

class Employee**<**T**>** **:** Person**<**T**>** **where** T **:** class **{...}**

Если в базовом классе в качестве ограничение указано class**,** то есть любой класс**,** то в производном классе также надо указать в качестве ограничения class**,** либо же какой**-**то конкретный класс**.**