**Оглавление**

[Классы 2](#_Toc165588509)

[Основные понятия 2](#_Toc165588510)

[Функции класса 3](#_Toc165588511)

[Первичные конструкторы 6](#_Toc165588512)

[Перегрузка метода 7](#_Toc165588513)

[Параметры методов в С# 8](#_Toc165588514)

[Виды модификаторов в параметре 8](#_Toc165588515)

[Операторы класса 10](#_Toc165588516)

[Наследование классов 11](#_Toc165588517)

[Абстрактные классы (abstract) 16](#_Toc165588518)

# Классы

## Основные понятия

*Класс* – это отсылочный тип данных.

Объект хранится в «куче» (**heap**).

*Куча* – это неупорядоченная область данных. Без инициализации значения переменных равны null.

Для инициализации объекта нужно вызвать конструктор.

Общая форма:

*ИмяКласса имяОбъекта = new ИмяКласса(параметры);*

Или

*ИмяКласса имяОбъекта = new(параметры);*

Примеры:

Random random = new();

DataTime birthday = new(2020, 7, 31);

Общая форма определения класса:

*модификатор class ИмяКласса*

{//данные класса

//функции класса}

Следует объявлять классы в отдельном файле. Имя файла совпадает именем класса.

Данные класса:

1. Поле экземпляра, то есть переменные, доступные через объект.

*модификатор Тип имяПоля;* //объявление

*имяОбъекта.имяПоля = значение;*

*переменная = имяОбъекта.имяПоля;*

1. *Статическое поле* – это переменная класса, доступная по имени класса.

*Модификатор static Тип имяПоля;*

*ИмяКласса.ИмяПоля = значение;*

1. Константа класса

модификатор const Тип ИмяКонстанты = значение;

ИмяКласса.ИмяКонстанты

Пример:

public class Person

{public string name;

public int age;

public const int MajorityAge=18;}

## Функции класса

В не статических функциях можно обращаться к текущему объекту **this** или **this.ИмяПоля**.

Виды функций:

1. Метод – это блок кода, содержащий ряд инструкций.

модиф типВозврата ИмяМетода(параметры)

{//действия(тело метода)}

Пример метода вывода информации об объекте класса Person:

Public void Print()

{Console.WriteLine($”Имя:{name}, возраст:{age}”);}

Метод для подсчета количества лет до совершеннолетия:

public int GetYears()

{return (MajorityAge – age>0)?

MajorityAge-age:0;}

Или

public int GetYears()=>

(MajorityAge – age>0)?MajorityAge-age:0;

1. Конструктор – это метод для задания начальных значений полям. Имя конструктора совпадает с именем класса. Модификатор обычно public. Конст пишется после данных.

Общая форма:

модиф ИмяКласса(параметры)

{//присваеваем значения}

Пример:

public Person(string name)

{**\_**name=name}

Конструктор по умолчанию есть всегда и у него нет параметров.

Цепочка конструкторов используется для уменьшения количества кода. Перед вызовом текущего конструктора будет вызван конструктор указанный после текущего параметра.

Общая форма:

Модиф ИмяКласса(параметры **: this(именяПараметров или значения**)

{//присвоение значений}

Примеры:

public Person(string name, int age)

{\_name = name;

\_age = age;}

public Person() : this (‘неизвестно’, 18){}

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

1. Деструктор (финализатор) – это метод для очистки памяти и ресурсов. Вызывается автоматически, когда GC (GarbageCollector) окончательно удаляет экземпляр класса.

ИмяКласса()

{//освобождение ресурсов}

1. Свойства property

Свойства – это специальные методы без параметров, которые обеспечивают доступ к закрытым полям и получению данных. Свойства являются оберткой для двух методов.

Методы:

1. get (получить) – это метод на чтение. Используется для возврата данных (должен присутствовать return).
2. set (установить) – это метод для записи данных. Его неявно передается параметр value, который можно проверить на корректность перед присваиваемым значению поля. Модификатор обычно public.

private string \_name;

public string Name

{get **=>** \_name;

set **=>** \_name = value;}

***=>***- это тело выражения, если нужно более одного действия, то нужно тело блока.

Пример проверки имени в set:

set{ value = value.Trim(); //обрезка пробелов по краям

if (\_name != value&&value!=””)

\_name = value;

get{return \_name;}

Пример свойства на чтение:

public int Years => MajorityAge - \_age;

Автореализуемые свойства делают объявления свойств более краткими. Используется если в свойствах не нужна дополнительная логика. Могут применяться для замены полей. Для автосвойств создается невидимая переменная.

модиф тип ИмяСвойства

{get; set;}

# Первичные конструкторы

Первичные конструкторы позволяют создать поля класса, указав параметр первичного конструктора.

Общая форма:

модиф class ИмяКласса(параметры первичного конструктора)

{//}

Пример:

public class Person(string name, int age)

public Person() : this (“неизвестно”, 18) {…}

Для использования других классов конструкторов применяется цепочка конструкторов для вызова первичного конструктора.

Поля первичного конструктора закрытые для обращения нужны свойства или методы.

Person person1 = new();

Person person2 = new();

Person[] persons = {new Person(), new Person(), new Person(name, 18)};

Индексаторы – это специальные методы без параметров, позволяет обращаться к объекту с использованием индексов как в массиве.

По синтаксису похожи неравенства

Имя всегда this

Общая форма:

модиф типВозврата this[типИндекса1, имяИндекса1, …]

{get {…}

set {…}}

Пример:

public object this[int index]

{get{return(object) this [index]}}

Или

public object **?** this[int index]

{get{return index switch

{1 => name,

2 => age,

\_=> null};}}

**?** может вернуть null.

Вызов:

C.W(person.Name);

C.W(person[1]); //вернёт имя

C.W(person[2]); //вернёт возраст

## Перегрузка метода

#region данные

public …

#endregion

Перегрузка – это приём программирования, позволяющее разработчикам использовать в одном классе одно и то же имя метода. У методов должны быть разные списки параметров (overload).

Всегда перегружены конструкторы и индексаторы всегда **this**.

Имя конструктора равно имя класса, а имя индексатора всегда **this**.

Примеры:

Public static int Add(int a, int b) => a+b;

Public static double Add(double a, double b) => a+b;

Public static int Add(int a, int b, int c) => a+b+c;

Public static void Add(int a, int b, **out** int sum) => sum=a+b;

***out*** – это выходной параметр.

Вызов:

Var x = Operations.Add(2, 3, 5);

C.W(x);

Operations.Add(2, 3, out x);

C.W(x);

……

# Параметры методов в С#

Общая форма указания параметра:

модиф тип имя = значениеПоУмолчанию

## Виды модификаторов в параметре

1. Без модификаторов

Public static int Add(int a, int b) => a+b;

В методе передаются копии параметров типов значений.

(int, doble, char, bool, struct, …)

Изменение таких параметров в методе не изменят его в основной программе.

Ссылочные параметры передаются по ссылке (class, массив (Array)).

**in** - параметр передается по ссылке и не может быть изменен в методе.

**ref** (reference) – параметр передается по ссылке и может быть изменен.

**out** - Параметр передается по ссылке и должен быть изменен в методе.

**params** – объединяет переменные количества аргументов один массив. Параметр с ним может быть только один. Он передается последним в списке.

Модификаторы in, ref, out указываются при вызове.

Operations.Add(2, 3, out x)

Пример метода с параметром **ref** для нахождения модуля числа:

Class Operations

{public static void GetAbsolute(ref int x)

{if(x<0)

x=-x;}

Вызов:

x = -7;

Operations. GetAbsolute(ref x);

C.W(x);

Пример метода с модификаторами **params** для вычисления суммы произвольного количества чисел:

public static int GetSum(params[] numbers)

{int sum = 0;

foreach(int number in numbers)

sum +=number;

return sum;}

Вызов:

int sum1 = Operations.GetSum(); //0

int sum2 = Operations.GetSum(1, 2, 3); //6

int[] prices = [150, 120, 30];

int sum3 = Operations.GetSum(prices); //300

Обязательны параметры, то есть параметры по умолчанию. При вызове их можно не указывать.

Пример:

public static int Add(int a, int b = 1) => a+b;

Вызов:

int sum4 = Operations.GetSum(2, 3); //2+3=5

int sum5 = Operations.GetSum(2); //2+1=3

При вызове в методе можно изменить порядок передачи параметров.

Пример:

Power(int x, int n) //x^n

int x = Power(2, 3); //2^3=8

int x = Power(n:2, x:3); //3^2=9

## Операторы класса

Язык С# предоставляет возможность перегрузки некоторых операторов.

1. Унарные операторы (+, -, !, ++, --, true, false)
2. Бинарные операторы (+, -, \*, /, %, &, |, ^, «, »)

*Бинарные операторы* – операторы, которые могут быть перегружены отдельно от других, которые работают с числами.

Бинарные. которые могут быть перегружены: ==, !=, <=, >=, >, <

Перегрузка оператора связана с перегрузкой метода.

Ключевое слово **operator**.

Операторные методы делятся на 2 вида:

1. Для унарных

Общая форма:

public static типВозврата **operator** оператор(тип имяПараметра) {…}

или

public static типВозврата **operator** оператор(тип1 имяПараметра1, тип2 имяПараметра2) {…}

типВозврата может быть любой, но обычно это тип класса, который выполняется перегрузка или bool.

Пример:

namespase Lection0208

{internal class Counter

{public int Value{get; set;}

//перегрузка унарного оператора

public static Counter operator++(Counter counter)

=> new Counter{Value = counter.Value+1};

//отдельная перегрузка бинарного оператора

public static Counter operator +(Counter a, Counter b)

=> new Counter{Value = a.Value + b.Value};

//попарная перегрузка бинарных операторов

public static bool operator > (Counter a, Counter b)

=> a.Value>b.Value;

public static bool operator < (Counter a, Counter b)

=> a.Value<b.Value;

//перегрузка оператора &

public static bool operator & (Counter a, Counter b)

=> a.Value=b.Value;

//true/false перегружают, если объект будет использоваться вместо условия

public static bool operator true (Counter counter)

=> counter.Value!=0;

public static bool operator афдыу (Counter counter)

=> counter.Value=0;

## Методы расширения

Методы расширения позволяют добавить новую функциональность, существующие типы данных без необходимости наследования.

Создаем статический класс, в котором добавляем нужный статический метод с первым параметром **this** тип имяПеременной.

Пример:

namespace Lection0208

{public static class StringExtensions

{public static string ToScreamingSnakeCase(**this** string s)

=> s.ToUpper().Replace(‘ ‘, ‘\_’);

Пример вызова:

string text=”abs. fgh zcxv”;

text=text.ToScreamingAnakeCase();

# Наследование классов

*Наследование классов* – это наследование позволяет дочернему классу использовать, расширять и изменять возможности родительского.

Общая форма:

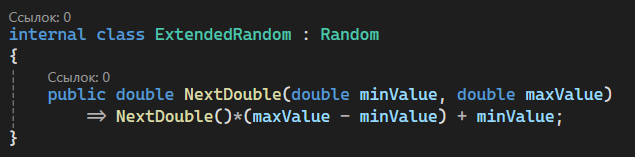
*модификатор class ИмяДочернего : ИмяРодительского*

{ //Данные

//Функции и т.д.}

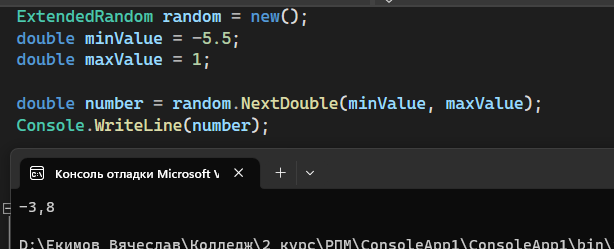
Новые слова: abstract, override, virtual, new, base, sealed.

Пример дочернего класса:



В этом классе мы расширили функциональность метода **NextDouble** и добавили ему перегрузку для создания рандомного вещественного числа в промежутке от minValue до maxValue.

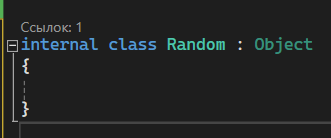
Пример реализации:



Особенности наследования:

* Дочернему классу доступны элементы родительского класса только с модификаторами public или protected.
* У дочернего класса может быть только 1 родитель.
* У любого родительского класса без родителя используется неявное наследование от класса Object. Его функции доступны любому телу.

Пример:



Конструкторы и деструкторы не наследуются.

Методы класса Object:

*ToString()* – преобразует объект в строку, по умолчанию возвращает полное имя типа. Вызывается автоматически если нужно привести объект к строке (т.е. вызывать его каждый раз не надо).

*Equals()* – возвращается как статический и как метод экземпляра. Позволяет проверять равенство двух объектов, по умолчанию проверяет, что две переменные ссылаются на одну и ту же область в памяти.

*GetType()* – возвращает объект типа **type**. Не может быть переопределен (override).

*GetHashCode()* – вычисляет **Hash** (условно его ключ который отличает его от других объектов) объекта, что позволяет использовать объект в Hash-ированных коллекциях

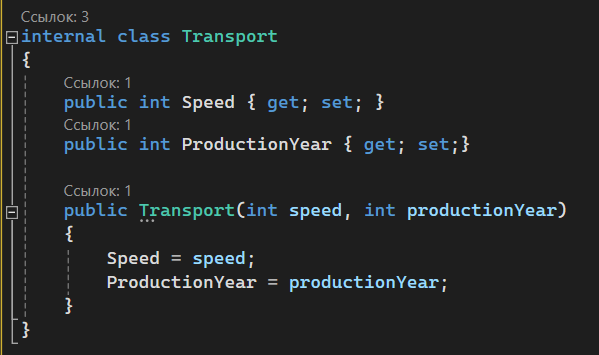
Общая форма:

*public ИмяКласса(параметры) : base(названия параметров)*

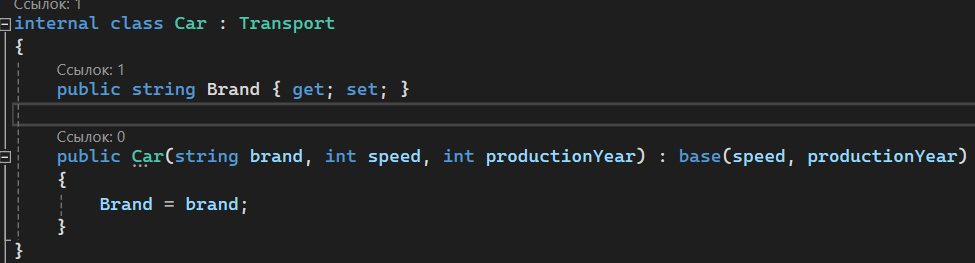
{//Действия}

Пример наследования:

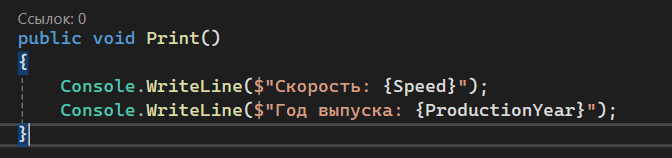
Родительский класс Transport



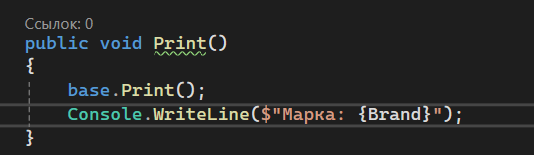
Дочерний класс Car



Метод Print для класса Transport



Метод Print для класса Car



Сокрытие применяется к методам, свойствам, индексаторам.

Сокрытие применяется если совпадают сигнатуры в дочернем и родительском классах (если названия методов, свойств или индексаторов одинаковые). Для сокрытия перед типом возврата ставят **new**.

Общая форма:

*модификатор* ***new*** *типВозврата ИмяМетода()*

{//тело метода}

Переопределение применяется к методам, свойствам, индексаторам.

*Переопределенный метод*(override) – изменяет поведение виртуального метода. Базовая реализация может быть вызвана или удалена.

*Виртуальный метод* (virtual) – может быть переопределен (override) в любом из дочерних классов иерархии наследования.

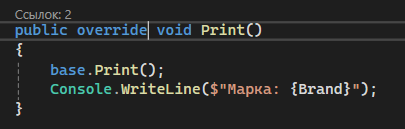
Виртуальный метод не может быть статическим (static) или закрытым (private).

Пример переопределения

Виртуальный метод в классе Transport

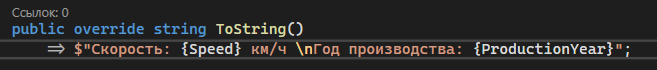


Переопределение этого метода в классе Car



Пример переопределения методов класса Object:

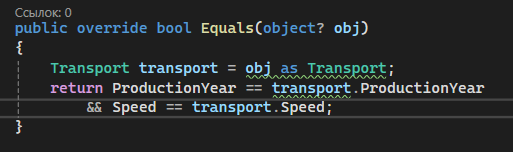
Класс Transport ToString



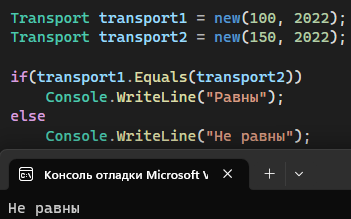
Класc Car ToString



Класс Transport Equals



Пример реализации Equals:



*Запечатанный класс* (sealed) – это класс, для которого нельзя создавать дочерние классы.

# Абстрактные классы (abstract)

Особенности: нельзя создать объект абстрактного класса.

Общая форма:

public abstract class ИмяКласса

{

//Данные

//Функции

}

В абстрактном классе можно объявлять абстрактные методы, свойства, индексаторы. У абстрактных методов нет тела, есть только сигнатура.

*модификатор abstract ИмяМетода (параметры);*

*модификатор abstract ИмяСвойства (get; set;)*

*модификатор abstract this[индексы] (get; set;)*

В дочернем классе должны быть реализованные все абстрактные методы, свойства, индексаторы. Реализация выполняется как переопределение(override), но без вызова базовой реализации. Если перед методом override писать sealed, то в дочернем классе нельзя переопределить.