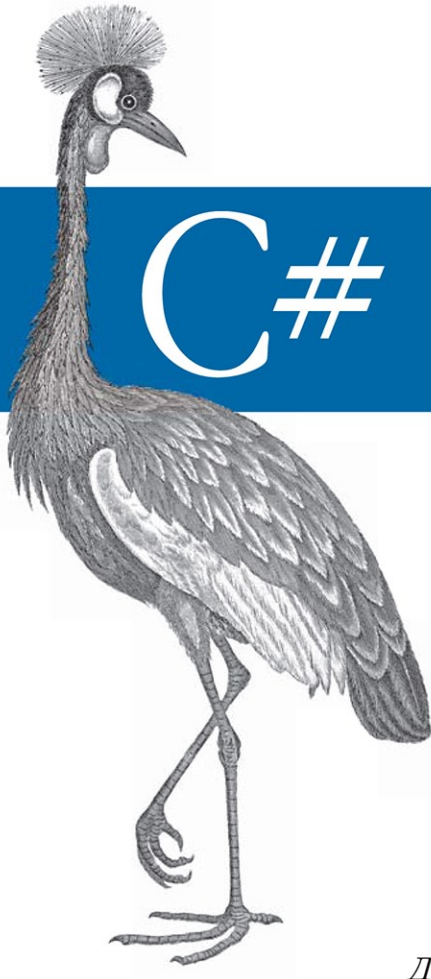
Скорая помощь для программистов на C# 5.0

C# 5.0

Карманный справочник



Джозеф Албахари
Бен Албахари

O'REILLY®

ББК 32.973.26-018.2.75
А45
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С. Н. Тригуб*
Перевод с английского и редакция докт. физ.-мат. наук
Д. А. Клюшина

По общим вопросам обращайтесь в Издательский дом "Вильямс"
по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Албахари, Джозеф, **Албахари**, Бен.

А45 C# 5.0. Карманный справочник. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2013. — ??? с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1820-8 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Джозеф Албахари, Бен Албахари

C# 5.0. Карманный справочник

Литературный редактор *И. А. Попова*
Верстка *А. Н. Полинчик*
Художественный редактор *В. Г. Павлютин*
Корректор *Л. А. Гордиенко*

Подписано в печать 13.11.2012. Формат 70x100/32
Гарнитура Times. Печать офсетная
Усл. печ. л. 11,61. Уч.-изд. л. 7,4
Тираж 2000 экз. Заказ № 0000

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9-я линия, 12/28

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1820-8 (рус.)
ISBN 978-1-4493-2017-1 (англ.)

Содержание

КАРМАННЫЙ СПРАВОЧНИК ПО C# 5.0 13

Обозначения, принятые в книге 14

Использование примеров кода 15

Библиотека Safari® Books Online 16

Ждем ваших отзывов! 17

Первая программа на языке C# 18

Компиляция 21

Синтаксис 22

Идентификаторы и ключевые слова 23

Предотвращение конфликтов 24

Контекстные ключевые слова 25

Литералы, знаки пунктуации и операции 25

Комментарии 26

Основы типов 27

Примеры предопределенных типов 27

Примеры пользовательских типов 29

Преобразования 32

Типы значений и ссылочные типы 33

Классификация предопределенных типов 37

Числовые типы 38

Числовые литералы 39

Числовые преобразования 41

Арифметические операции 42

Специализированные целочисленные операции 42
8- и 16-битовые целочисленные типы 44
Специальные значения типов float и double 45
Ошибки округления действительных чисел 47
Булев тип и операции 47
Операция проверки равенства и сравнения 48
Условные операции 48
Строки и символы 50
Тип string 51
Массивы 54
Инициализация элемента по умолчанию 56
Многомерные массивы 57
Переменные и параметры 59
Стек и куча 59
Определенное присваивание 61
Значения по умолчанию 62
Выражения и операции 70
Выражения присваивания 71
Приоритет и ассоциативность операций 72
Операторы 76
Оператор объявления 76
Операторы-выражения 77
Оператор выбора 77
Итерационные операторы 81
Операторы перехода 84

Пространства имен 85
Правила в пространстве имен 88
Классы 90
Статические классы 102
Финализаторы 102
Частичные типы и методы 103
Наследование 105
Виртуальные функции-члены 109
Конструкторы и наследование 113
Перегрузка и разрешение 115
Тип object 116
Упаковка и распаковка 117
Статическая и динамическая проверка 118
Метод GetType и операция typeof 119
Структуры 122
Семантика создания структуры 123
Модификаторы доступа 123
Интерфейсы 125
Расширение интерфейса 127
Явная реализация интерфейса 127
Виртуальная реализация членов интерфейса 128
Повторная реализация интерфейса в подклассе 129
Перечисления 130
Преобразования перечислений 131
Перечисление флагов перечислений 132
Операции над перечислениями 133

Вложенные типы 133

Обобщения 134
Обобщенные типы 135
Обобщенные методы 137
Объявление параметров типа 138
Операция typeof и несвязанные обобщенные типы 139
Ограничения обобщений 140
Вывод подклассов обобщенных типов 142
Самоссылающиеся обобщенные объявления 142
Статические данные 143
Ковариантность 143
Контравариантность 146

Делегаты 147
Создание подключаемых методов с помощью делегатов 148
Групповые делегаты 149
Целевые методы экземпляра и целевые статические методы 151
Обобщенные типы делегатов 151
Делегаты Func и Action 151
Совместимость делегатов 152

События 155
Стандартная модель событий 158
Методы доступа в событиях 161

Лямбда-выражения 163
Захват внешних переменных 165

Анонимные методы 168

Оператор try и исключения 169
Раздел catch 172
Блок finally 174
Генерирование исключений 176
Основные свойства класса System.Exception 177
Общие типы исключений 178

Перечисления и итераторы 180
Перечисление 180
Инициализаторы коллекции 181
Итераторы 182
Семантика итераторов 183
Композиции последовательностей 185

Типы, допускающие значение NULL 187
Структура Nullable<T> 187
Преобразования, допускающие значение NULL 188
Упаковка/распаковка значений, допускающих NULL 189
Заимствование операций 189
Применение операций & и | к операндам типа bool? 192
Операция ?? 192

Перегрузка операций 193
Операторные функции 194
Перегрузка операций проверки равенства и сравнения 195

Ячные и неявные пользовательские преобразования 196
Расширяющие методы 197
Создание цепочек методов расширения 198
Неоднозначность и разрешение 199
Анонимные типы 199
Язык запросов LINQ 201
Основы языка LINQ 201
Отложенное выполнение 207
Стандартные операции запроса 210
Создание цепочек операций запроса 214
Выражения запросов 215
Ключевое слово let 220
Продолжения запросов 221
Множественные генераторы 222
Соединение 224
Упорядочение 229
Группирование 230
Методы OfType и Cast 232
Динамическое связывание 233
Статическое и динамическое связывание 234
Специальное связывание 236
Языковое связывание 237
Исключение RuntimeBinderException 239
Представление динамического типа на этапе выполнения программы 240
Динамические преобразования 241

Типы var и dynamic 241
Динамические выражения 242
Разрешение перегрузки динамического члена 243
Невызываемые функции 245

Атрибуты 246
Классы атрибутов 247
Именованные и позиционные параметры атрибутов 248
Цели атрибутов 249
Задание нескольких атрибутов 249
Создание собственных атрибутов 249
Получение атрибутов на этапе выполнения программы 251

Атрибуты сведений о вызывающей стороне (C# 5.0) 252

Асинхронные функции (C# 5.0) 254
Ключевые слова await и async 256
Перехват локального состояния 260
Создание асинхронных функций 261
Параллелизм 264
Асинхронные лямбда-выражения 265

Небезопасный код и указатели 266
Основы указателей 267
Небезопасный код 267
Оператор fixed 268
Указатель на член класса 269
Массивы 270

Указатель void* 271
Директивы препроцессора 272
Директива pragma warning 274
Документация XML 275
Стандартные дескрипторы документации XML 276
Об авторах 280
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 281

Первая программа на языке C#

Ниже приведена программа, умножающая 12 на 30 и выводящая результат, 360, на экран. Двойная косая черта означает, что оставшаяся часть строки является *комментарием*.

```
using System;                   // Импорт пространства имен

class Test                      // Объявление класса
{
  static void Main()            // Объявление метода
  {
    int x = 12 * 30;            // Оператор 1
    Console.WriteLine (x);      // Оператор 2
  }                             // Конец метода
}                               // Конец класса
```

Ядром этой программы являются два *оператора*[1]. Операторы в языке C# выполняются последовательно и заканчиваются точкой с запятой. Первый оператор вычисляет *выражение* `12 * 30` и заносит результат в *локальную переменную* с именем `x`, имеющую целый тип. Второй оператор вызывает *метод* `WriteLine` класса `Console`, чтобы вывести переменную `x` в текстовое окно на экране.

Метод выполняет действие, выраженное последовательностью операторов, называемой *блоком операторов* — парой фигурных скобок, содержащих операторы или не содержащих ничего. Мы определили один метод с именем `Main`.

[1] В книге принят следующий перевод: statement — оператор, operator — операция. — *Примеч. ред.*

Использование функций более высокого уровня, вызывающих функции более низкого уровня, упрощает программу. Мы можем провести *рефакторинг* нашей программы с помощью повторно используемого метода, умножающего целое число на 12.

```
using System;

class Test
{
  static void Main()
  {
    Console.WriteLine (FeetToInches (30));    // 360
    Console.WriteLine (FeetToInches (100));   // 1200
  }

  static int FeetToInches (int feet)
  {
    int inches = feet * 12;
    return inches;
  }
}
```

Метод может получать *входные* данные от вызывающего метода с помощью указания *параметров* и *выводить* данные в вызывающий метод, задавая *тип возвращаемого значения*. Мы определили метод `FeetToInches`, имеющий параметр для ввода измерения в футах, и тип возвращаемого значения для вывода измерения в дюймах. Оба значения имеют тип `int`.

Литералы `30` и `100` являются *аргументами*, передаваемыми методу `FeetToInches`. Метод `Main` в нашем примере имеет пустые скобки, потому что у него нет параметров, и тип `void`, потому что он ничего не возвращает.

Язык C# распознает метод `Main` как входную точку по умолчанию для выполнения программы.

Метод `Main` может возвращать и целое число (а не тип `void`), чтобы передать информацию в исполняющую среду. Метод `Main` может также принимать в качестве параметра массив строк (заполненный любыми аргументами). Например:

```
static int Main (string[] args) {...}
```

ЗАМЕЧАНИЕ

Массив (например, `string[]`) представляет собой фиксированное количество элементов конкретного типа (более подробную информацию см. в разделе "Массивы").

Методы представляют собой одну из нескольких разновидностей функций в языке C#. Другой разновидностью функций является *операция* `*`, выполняющая умножение. Существуют также *конструкторы*, *свойства*, *события*, *индексаторы* и *финализаторы*.

В нашем примере два методы сгруппированы в классе. *Класс* группирует функции-члены и данные-члены в объектно-ориентированный строительный блок. Класс `Console` содержит члены, обрабатывающие ввод и вывод в командной строке, например метод `WriteLine`. Класс `Test` содержит два метода — `Main` и `FeetToInches`. Как будет показано в разделе "Основы типов", класс — это разновидность *типа*.

На самом верхнем уровне программы типы организованы в *пространства имен*. Директива `using` открывает приложению доступ к пространству имен `System`, позволяя использовать класс `Console`. Например, все наши классы можно было бы определить в пространстве имен `TestPrograms`.

```
using System;

namespace TestPrograms
{
  class Test {...}
  class Test2 {...}
}
```

Платформа .NET Framework имеет вложенные пространства имен. Например, существует пространство имен, содержащее типы для обработки текстов.

```
using System.Text;
```

Директива `using` здесь используется для удобства; на тип можно сослаться, просто задав полностью его определенное имя, т.е. имена типа и его пространства имен, например `System.Text.String Builder`.

Компиляция

Компилятор языка C# группирует исходный код, заданный в виде набора файлов с расширением *.cs*, в *сборку*. Сборка — это единица упаковки и развертывания на платформе .NET. Сборка может быть *приложением* или *библиотекой*. Обычное консольное приложение или Windows-приложение имеет метод `Main` и представляет собой файл с расширением *.exe*. Биб-

лиотека — это файл с расширением *.dll*, являющийся эквивалентом файла с расширением *.exe*, но без точки входа. Ее может вызывать (ссылаться на нее) приложение или другие библиотеки. Платформа .NET Framework — это набор библиотек.

Компилятор языка C# представляет собой файл *csc.exe*. Для компиляции вы можете использовать *интегрированную среду разработки*, например Visual Studio, или вызвать файл `csc` вручную из командной строки. При ручной компиляции программу сначала необходимо сохранить в файле, например *MyFirstProgram.cs*, а затем перейти в командную строку и вызвать компилятор `csc` (расположенный в каталоге *%SystemRoot%\Microsoft.NET\Framework\<версия-платформы>*, где *%SystemRoot%* – ваш каталог системы Windows).

```
csc MyFirstProgram.cs
```

В результате будет создано приложение с именем *MyFirstProgram.exe*.

Для создания библиотеки (*.dll*) выполните команду

```
csc /target:library MyFirstProgram.cs
```

Синтаксис

Синтаксис языка C# близок к синтаксису языков C и C++. В этом разделе мы опишем элементы синтаксиса языка C# с помощью следующей программы:

```
using System;

class Test
```

```
{
  static void Main()
  {
    int x = 12 * 30;
    Console.WriteLine (x);
  }
}
```

Идентификаторы и ключевые слова

Идентификаторы — это имена, которыми программисты называют свои классы, методы, переменные и т.д. В нашем примере также есть идентификаторы. Перечислим их в порядке появления.

```
System  Test  Main  x  Console  WriteLine
```

Идентификатор должен быть неразрывным словом, составленным из символов Unicode, и начинаться с буквы или символа подчеркивания. Идентификаторы в языке C# чувствительны к регистру клавиатуры. По соглашению имена параметров, локальных переменных и закрытых полей должны набираться в "верблюжьем стиле" (например, `myVariable`), а все остальные идентификаторы — в стиле языка Pascal (например, `MyMethod`).

Ключевые слова — это имена, зарезервированные компилятором. Их нельзя использовать в качестве идентификаторов. В нашем примере содержатся следующие ключевые слова:

```
using  class  static  void  int
```

Приведем полный список ключевых слов в языке C#.

```
abstract    as          base        bool
break       byte        case        catch
char        checked     class       const
continue    decimal     default     delegate
do          double      else        enum
event       explicit    extern      false
finally     fixed       float       for
foreach     goto        if          implicit
in          int         interface   internal
is          lock        long        namespace
new         null        object      operator
out         override    params      private
protected   public      readonly    ref
return      sbyte       sealed      short
sizeof      stackalloc  static      string
struct      switch      this        throw
true        try         typeof      uint
ulong       unchecked   unsafe      ushort
using       virtual     void        while
```

Предотвращение конфликтов

Если вы все же хотите использовать идентификатор, совпадающий с ключевым словом, поставьте перед ним префикс `@`. Например:

```
class class {...}    // Нельзя
class @class {...}   // Можно
```

Символ `@` не является частью самого идентификатора. Поэтому имя `@myVariable` совпадает с именем `myVariable`.

Контекстные ключевые слова

Некоторые ключевые слова являются *контекстными*, т.е. их можно использовать и в качестве идентификаторов, без символа `@`. Перечислим эти ключевые слова.

```
add          ascending      async        await
by           descending     dynamic      equals
from         get            global       group
in           into           join         let
on           orderby        partial      remove
select       set            value        var
where        yield
```

При использовании контекстных ключевых слов никакой неоднозначности в соответствующем контексте возникнуть не может.

Литералы, знаки пунктуации и операции

Литералы — это простейшие фрагменты данных, лексически встроенные в программу. Литералами в данном примере являются числа `12` и `30`.

Знаки пунктуации размечают структуру программы. Знаками пунктуации в данном примере были `{`, `}` и `;`. Фигурные скобки группируют несколько операторов в *блок операторов*. Точка с запятой завершает оператор (но не блок). Операторы могут содержать несколько строк.

```
Console.WriteLine
 (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10);
```

Операция преобразовывает и объединяет выражения. Большинство операций в языке C# обозначаются

символами, например, операция умножения *. В нашем примере использовались следующие операции:

```
.    ()    *    =
```

Точка обозначает член какой-нибудь сущности (или десятичную точку в числовых литералах). Скобки используются при объявлении или вызове метода; пустые скобки используются, если метод не имеет аргументов. Знак равенства обозначает присваивание (двойной знак равенства == означает сравнение на равенство).

Комментарии

В языке C# есть два разных стиля документирования исходного кода: *однострочные* и *многострочные комментарии*. Однострочный комментарий начинается с двойной косой черты и продолжается до конца строки. Рассмотрим пример.

```
int x = 3;  // Комментарий о присваивании числа 3 переменной x
```

Многострочный комментарий начинается символами /* и завершается символами */. Рассмотрим пример.

```
int x = 3; /* Это комментарий,
              занимающий две строки */
```

Комментарии могут быть встроены в дескрипторы документации XML (см. раздел “Документация XML”).

Основы типов

Тип определяет концепцию для значения. В данном примере мы использовали два литерала типа `int` со значениями 12 и 30. Мы также объявили *переменную* типа `int` с именем `x`.

Переменная определяет место хранения, которое в разные моменты времени может содержать разные значения. В противоположность этому *константа* всегда представляет одно и то же значение (более подробно мы поговорим об этом позднее).

Все значения в языке C# являются *экземплярами* конкретного типа. Смысл значения и набор возможных значений, которые может принимать переменная, определяются ее типом.

Примеры предопределенных типов

Предопределенные типы (они также называются *встроенными*) — это типы, которые специально поддерживаются компилятором. Тип `int` — это предопределенный тип, описывающий множество целых чисел, которые можно записать в памяти с помощью 32 битов в диапазоне от -2^{31} до $2^{31}-1$. К экземплярам типа `int` можно применять функции, например арифметические.

```
int x = 12 * 30;
```

Другим предопределенным типом в языке C# является тип `string`. Этот тип `string` представляет последо-

вательность символов, например ".NET" или "*http://oreilly.com*". К строкам также можно применять функции.

```
string message = "Hello world";
string upperMessage = message.ToUpper();
Console.WriteLine (upperMessage);      // HELLO WORLD

int x = 2012;
message = message + x.ToString();
Console.WriteLine (message);           // Hello world2012
```

Предопределенный тип `bool` имеет только два возможных значения: `true` и `false`. Тип `bool` широко используется в условных конструкциях с оператором `if`. Рассмотрим пример.

```
bool simpleVar = false;
if (simpleVar)
  Console.WriteLine ("This will not print");

int x = 5000;
bool lessThanAMile = x < 5280;
if (lessThanAMile)
  Console.WriteLine ("This will print");
```

ЗАМЕЧАНИЕ

Пространство имен `System` на платформе .NET Framework содержит много важных типов, которые не являются предопределенными в языке C# (например, `DateTime`).

Примеры пользовательских типов

Точно так же, как сложные функции состоят из простых функций, можно создавать сложные типы из элементарных типов. В данном примере мы определим пользовательский тип `UnitConverter` — класс, который будет служить схемой для преобразования единиц измерений.

```
using System;

public class UnitConverter
{
  int ratio;                                  // Поле

  public UnitConverter (int unitRatio)    // Конструктор
  {
    ratio = unitRatio;
  }

  public int Convert (int unit)           // Метод
  {
    return unit * ratio;
  }
}

class Test
{
  static void Main()
  {
    UnitConverter feetToInches = new Uni tConverter(12);
    UnitConverter milesToFeet = new UnitCo nverter(5280);

    Console.Write (feetToInches.Convert(30));   // 360
    Console.Write (feetToInches.Convert(100));  // 1200
    Console.Write (feetToInches.Convert
                    (milesToFeet.Convert(1)));  // 63360
  }
}
```

Член типа

Тип содержит *данные-члены* и *функции-члены*. Данные-члены класса `UnitConverter` представляют собой *поле* с именем `ratio`. Функциями-членами класса `UnitConverter` являются метод `Convert` и *конструктор* класса `UnitConverter`.

Симметрия между элементарными и пользовательскими типами

Прекрасным свойством языка C# является то, что между предопределенными и пользовательскими типами есть мало различий. Предопределенный тип `int` является схемой для описаний целых чисел. Он содержит данные — 32 бита — и функции-члены, использующие эти данные, например `ToString`. Аналогично наш пользовательский тип `UnitConverter` представляет собой схему для преобразований единиц измерения. Он содержит данные — коэффициент — и функции-члены для работы с этими данными.

Конструкторы и создание объектов

Данные создаются путем *создания объекта данного типа*. Объекты предопределенных типов можно создавать просто в виде литералов, например `12` или `"Hello, world"`.

Оператор `new` создает объекты пользовательского типа. В начале метода `Main` создаются два экземпляра типа `UnitConverter`. Сразу после того, как оператор `new` создаст объект, вызывается *конструктор* объекта для его инициализации. Конструктор определяется как

обычный метод, за исключением того, что его имя и тип возвращаемого значения должны совпадать с именем содержащего его класса.

```
public UnitConverter (int unitRatio)  // Конструктор
{
  ratio = unitRatio;
}
```

Экземпляр и статические члены

Данные-члены и функции-члены, действующие на *экземпляр* типа, называются *членами экземпляра*. Метод `Convert` класса `UnitConverter` и метод `ToString` типа `int` — это примеры членов экземпляра. По умолчанию члены класса являются членами экземпляра.

Данные-члены и функции-члены, действующие не на экземпляр типа, а на сам тип, называются статическими `static`. Примерами статических методов являются методы `Test.Main` и `Console.WriteLine`. Класс `Console` на самом деле является *статическим классом*, т.е. *все* его члены являются статическими. Экземпляр класса `Console` никогда не создается — все приложение использует одну консоль.

Для контракта со статическими методами отметим поле экземпляра `Name`, принадлежащее экземпляру типа `Panda`, в то время как поле `Population` принадлежит всем экземплярам типа `Panda`.

```
public class Panda
{
  public string Name;           // Поле экземпляра
  public static int Population; // Статическое поле
```

```
  public Panda (string n)      // Конструктор
  {
    Name = n;                  // Присваиваем поле экземпляра
    Population = Population+1; // Увеличиваем статическое поле
  }
}
```

Следующий код создает два экземпляра класса `Panda`, печатает их имена, а затем распечатывает всю популяцию:

```
Panda p1 = new Panda ("Pan Dee");
Panda p2 = new Panda ("Pan Dah");

Console.WriteLine (p1.Name);           // Pan Dee
Console.WriteLine (p2.Name);           // Pan Dah
Console.WriteLine (Panda.Population);  // 2
```

Ключевое слово public

Ключевое слово `public` открывает доступ к членам для функций-членов других классов. Если бы в данном примере поле `Name` в классе `Panda` не было *открытым*, то класс `Test` не имел бы к нему доступа. Ключевое слово `public` описывает способ общения типа: "Здесь находится то, что можно видеть другим типам, за исключением закрытых деталей моей реализации". В терминах объектно-ориентированного программирования говорят, что класс *инкапсулирует* свои закрытые члены.

Преобразования

Язык C# может преобразовывать друг в друга экземпляры сравнимых типов. Преобразование всегда создает новое значение на основе существующего. Преобразования могут быть *неявными* или *явными*: неявные пре-