Содержание

Марку, Нику, Брэнди
и всем сотрудникам группы компаний Enterprise Family, доказавшим,
что мы способны предвидеть и добиваться цели

Об авторе

Уильям Робисон – начальник отдела корпоративных приложений компании Enterprise Social Investment Corporation (Колумбия, штат Мэриленд) и обладатель сертификата MCSE. Робисон имеет четырнадцатилетний стаж проектирования и разработки информационных систем. За это время он занимал различные административные и технические должности на предприятиях BBC и в частных компаниях. Робисону довелось работать на различных платформах, включая настольные ПК и рабочие станции, сервера под управлением ОС NT и UNIX, а также большие ЭВМ фирмы IBM. В представленной книге сконцентрирован его опыт программирования на языках C++, Java, а теперь и C#. В сферу профессиональных интересов Робисона входят распределенные системы, моделирование, симулирование и визуализация.

Благодарности

Чтобы написать книгу, недостаточно иметь опыт работы и ввести текст. Сначала следует подыскать интересную тему, а платформа .NET – это именно то, что нужно. Сотрудники компании Microsoft неплохо потрудились, и я благодарен им за то, что они нашли время поделиться своими идеями. Отдельное спасибо Конни Салливан (Connie Sullivan) за помощь в работе над этой книгой (и за восхитительные пикники в Сиэттле!). Я желаю ей всего самого наилучшего.

За помощь в доведении этой работы до логического завершения я благодарю Нейла Роуи (Neil Rowe). Не могу также не отметить усилия Сьюзен Хоббс (Susan Hobbs), Барбары Хача (Barbara Hacha), Маттиаса Сьегрена (Mattias Sjogren и Джорджа Недефа (George Nedeff) по исключению из текста всего лишнего. Спасибо всем вам – работать с вами было истинным удовольствием.

Однако я не смог бы написать эту книгу без поддержки своих родных и друзей. Особо хочу поблагодарить Брэнди Спицер (Brandi Spitzer) за то, что она помогла мне уложиться в график и мирилась с тем, что я несколько месяцев провел, уединившись в своем кабинете. Не будь ее, книга никогда не увидела бы света.

И, наконец, мои благодарности Деби, Терезе, Джиму, Эду и Хэлен. Я очень ценю вашу поддержку.

Сообщите нам ваше мнение

Вы – читатель этой книги – наш самый главный критик и рецензент. Мы ценим ваше мнение и хотим знать, что сделано правильно, что можно было бы сделать лучше, по каким темам стоило бы напечатать другие книги. Сообщите, что вам понравилось, а что не понравилось в этой книге, и что, на ваш взгляд, следует предпринять, чтобы наши издания стали лучше. Издательства «ДМК Пресс» и «Sams» ждут ваших комментариев. Вы можете отправить их по факсу, по электронной или обычной почте. В своем письме не забудьте, пожалуйста, указать название и авторов книги, а также ваше имя и почтовый адрес. Мы внимательно изучим все замечания и передадим их авторам и редакторам, работавшим над книгой.

E-mail (Sams): feedback@quepublishing.com
E-mail (ДМК Пресс): editor-in-chief@dmkpress.ru

Введение

Здравствуйте. Купив эту книгу, вы открываете окно в будущее программирования на платформе Microsoft. Язык C# – это составная часть семейства технологий под общим названием «платформа .NET», на базе которых Microsoft предлагает строить приложения нового поколения. Трудно отрицать, что описанная технология является мощной и устремленной в будущее, а язык C# – ее неотъемлемая часть.

Книга состоит из трех частей. Часть I представляет собой компактное изложение концепций самого языка. Хотя C# напоминает языки C++ и Java, его внутреннее устройство существенно отличается. Здесь вы узнаете, чем именно.

Часть II – как раз то, ради чего написана книга. Здесь приведено множество примеров, иллюстрирующих различные приемы программирования на языке C#. В главах 5–11 вы сможете найти фрагменты кода, показывающие, как можно решить стоящую перед вами проблему.

Часть III содержит некоторые справочные материалы. В приложениях A и B дана формальная грамматика языка и приведено описание наиболее часто используемых классов из библиотеки базовых классов (Base Class Library).

Настоящая книга не ставит целью научить вас создавать программы для платформы .NET, и уж тем более это не учебник по C# для программистов на Java. Я предполагаю, что вы умеете программировать на каком-то другом языке и знакомы с базовыми понятиями, поэтому можете сразу приступить к освоению нового материала. Конечно, я остановлюсь на некоторых аспектах платформы .NET, поскольку именно для нее и разработан изучаемый язык, но, если потребуется сделать выбор между более подробным рассказом о .NET или о C#, я предпочту C#.

Разумеется, я надеюсь, что вы читаете эти строки, уже купив книгу. Если так, спасибо вам за покупку! Если же вы сейчас стоите в книжном магазине и пытаетесь выбрать книгу по языку C#, надеюсь, что вы остановите свой выбор именно на этой. Более компактного, очищенного от словесной шелухи и полезного справочника по C# вам все равно не найти.

Билл Робисон,
осень 2001 года

Часть I

Основы языка C#

Язык C# – это результат критического пересмотра и расширения языка C++. Но даже опытному программисту на C++ предстоит многому научиться, прежде чем он сможет с той же продуктивностью работать на C#. В части I изложены основные сведения, необходимые для программирования на языке C#. В главе 1 описан синтаксис языка и его основные конструкции. В главе 2 приведены программы, поставляемые в составе .NET Framework SDK, с помощью которых вы можете откомпилировать и связать программы и библиотеки. В главе 3 представлен обзор библиотеки времени исполнения, которой вы можете пользоваться в своих приложениях.

Глава 1. Элементы языка

Язык программирования C# основан на языке C++, поэтому многое можно понять, изучая примеры кода. Но таким образом все же трудно составить полное представление об основных элементах языка. В этой главе мы попытаемся навести мост между языком, которым вы уже владеете, и языком C#, сведя воедино все синтаксические особенности C# и подготовив почву для последующего изложения.

Компания Microsoft представляет C# как «простой, современный, объектно-ориентированный и безопасный по отношению к типам» язык и позиционирует его как высокопродуктивный инструмент для использования возможностей нового каркаса разработки приложений. Команде разработчиков C# в значительной мере удалось добиться поставленных целей и создать мощный язык, способный конкурировать с аналогичными существующими технологиями.

Структура программы

Синтаксис языка C# сильно напоминает C++. Начнем с базовой структуры программы. В листинге 1.1 приведен простой пример программы на языке C#, которая печатает текстовое сообщение.

Листинг 1.1. Первое знакомство с C#

```
1:  using System;
2:
3:  /// <summary>
4:  /// Демонстрация структуры простейшей программы на C#.
5:  /// </summary>
6:  class SimpleStart
7:  {
8:      static void Main(string[] args)
```

```
 9:      {
10:         // Вывести текст на экран.
11:         Console.WriteLine("Это совсем простая программа.\n");
12:      }
13:  }
```

Код на C# представляет собой последовательность предложений, разделяемых точкой с запятой. Некоторые предложения могут содержать внутри себя другие предложения при условии, что они заключены в фигурные скобки. Примером может служить предложение `class SimpleStart {}` в листинге 1.1. Как правило, предложения записываются в свободном формате, то есть пробелы не принимаются во внимание, но внутри ключевого слова, идентификатора и других подобных элементов языка пробелы недопустимы. Так, следующие конструкции корректны:

```
int x = 4444;
string y = "This is a string.";
```

а вот такие – уже нет:

```
int x = 44 44;            // Пробел внутри лексемы недопустим.
string y = "This is       // Символ перевода строки внутри
           a string.";    // строковой константы тоже недопустим.
```

Из этого примера видно, что комментарий может начинаться двумя символами косой черты (`//`). Можно также использовать традиционную для языка C форму (`/* комментарий */`) или три идущих подряд символа косой черты для выделения фрагментов документации в формате XML.

Предложения могут быть декларативными; так, предложение `class` в строке 6 листинга 11 объявляет элемент программы. Предложения могут также быть императивными, то есть осуществлять некоторое действие во время выполнения программы; примером служит предложение `Console.WriteLine(...)` в строке 11.

В языке C# применяется концепция *пространства имен* для организации определений символов. Любой элемент программы, на который вы ссылаетесь, должен быть объявлен либо в пространстве имен, где находится ссылка, либо в пространстве имен, импортированном с помощью предложения `using`, либо ссылку нужно полностью квалифицировать. В последних двух случаях пространство имен должно быть или частью вашей программы, или принадлежать сборке, которая стала доступна вашей программе в результате процедуры объединения (fusion), выполняемой каркасом .NET Framework. Поскольку в этом примере в строке 11 применяется объект `System.Console`, то в самом начале программы импортируется пространство имен `System`, принадлежащее единой среде исполнения (CLR – Common Language Runtime), которая находится в глобальном кэше сборок.

C# отличается от C++ тем, что все переменные, функции и другие элементы программы объявляются в каком-то классе. Никаких глобальных констант, опережающих объявлений функций и других подобных конструкций не существует. В C# отсутствуют также заголовочные файлы, поскольку в среде .NET они не нужны.

Примечание *Утверждение о том, что не существует глобальных констант и объявлений, верно лишь до известной степени. На самом деле запрещены лишь глобальные объявления без указания области действия (unscoped). Но вы по-прежнему можете объявлять статические члены классов, каковыми могут быть как константы, так и функции общего назначения, и такие объекты будут доступны в любой точке программы. В самой среде исполнения подобных статических объявлений констант и функций не перечесть. Но все они помещены внутрь того или иного класса, чтобы уменьшить вероятность конфликта имен.*

Типы и объявления

В языке C# есть две разновидности типов: значащие и ссылочные. *Ссылочные типы* описывают объекты, их экземпляры размещаются в куче. *Значащие типы* предназначены для оптимизации простых типов, таких как целые числа и числа с плавающей точкой; их экземпляры хранятся в стеке, к ним не применяются процедуры инициализации и уничтожения. Однако с экземпляром любого значащего типа можно работать как с объектом, что достигается за счет так называемой процедуры *обертывания* (boxing).

Встроенные значащие типы

За двумя исключениями, значащие типы – это атомарные фундаментальные для системы типы. Экземпляры значащих типов размещаются в стеке, что ускоряет создание, доступ и уничтожение. В табл. 1.1 перечислены все имеющиеся на текущий момент значащие типы.

Таблица 1.1. Значащие типы в языке C#

Ключевое слово	Значения	Тип в среде исполнения
`sbyte`	Знаковое 8-разрядное целое	SByte
`byte`	Беззнаковое 8-разрядное целое	Byte
`short`	Знаковое 16-разрядное целое	Int16
`ushort`	Беззнаковое 16-разрядное целое	UInt16
`int`	Знаковое 32-разрядное целое	Int32
`uint`	Беззнаковое 32-разрядное целое	UInt32
`long`	Знаковое 64-разрядное целое	Int64
`ulong`	Беззнаковое 64-разрядное целое	UInt64
`float`	32-разрядное с плавающей точкой	Single
`double`	64-разрядное с плавающей точкой	Double
`decimal`	128-разрядное с плавающей точкой	Object (`Decimal`)
`bool`	Булевское	Boolean
`char`	Широкий символ	Char

Таблица 1.1. Значащие типы в языке C# (окончание)

Ключевое слово	Значения	Тип в среде исполнения
`enum`	Определяется пользователем	Int32
`struct`	Определяется пользователем	Переменный, по умолчанию Int32

Встроенные операторы

В табл. 1.2 перечислены в порядке убывания приоритета операторы языка C#, применимые к значащим типам. Каждый знакомый с языком C++ не встретит никаких трудностей в применении операторов C#. Правда, бросается в глаза отсутствие операторов для работы с указателями (`*`, `->`) и области действия класса (`::`). В C# есть только оператор «точка» (`.`) для выбора члена, причем неважно, принадлежит ли член к значащему типу, ссылочному типу или является статическим. Ясность программы от этого несколько пострадала, но количество ошибок, особенно допускаемых программистами, которые только начинают знакомство с языком, должно уменьшиться.

Таблица 1.2. Операторы языка C#

Тип	Оператор	Действие
Первичные	`.`	Выбор члена (например, `myObj.member`)
	`[]`	Индекс элемента массива или индексатора
	`()`	Вызов функции (например, `MyFunc( aParam )`)
	`a++, a–`	Постинкремент/постдекремент
	`new`	Выделение памяти
	`typeof`	Определение типа во время выполнения
	`(un)checked`	Включение (выключение) контроля границы массива
Унарные	`+, -`	Знак
	`!`	Булевское отрицание (NOT)
	`~`	Поразрядная операция НЕ
	`++a, –a`	Прединкремент/предекремент
	`(Typename)a`	Явное приведение типа (например, `(int)f`)
Мультипликативные	`*, /`	Умножение, деление
	`%`	Деление по модулю
Аддитивные	`+, -`	Сложение, вычитание
Сдвига	`<<, >>`	Поразрядный сдвиг влево, вправо
Условные	`<, >, <=, >=`	Меньше, больше, меньше или равно, больше или равно
	`is`	Определение типа во время выполнения
	`as`	Безопасное приведение типа
Сравнения	`==, !=`	Проверка на равенство

Таблица 1.2. Операторы языка C# (окончание)

Тип	Оператор	Действие
Поразрядное И	`&`	Бит результата установлен, если установлены соответствующие биты обоих операндов
Поразрядное Исключающее ИЛИ	`^`	Бит результата установлен, если установлен соответствующий бит ровно в одном из операндов
Поразрядное ИЛИ	`|`	Бит результата установлен, если установлен соответствующий бит хотя бы в одном из операндов
Булевское И	`&&`	Результат принимает значение «истина», если оба операнда истинны
Булевское ИЛИ	`||`	Результат принимает значение «истина», если хотя бы один операнд принимает значение «истина»
Булевский выбор	`?:`	Выбрать одно из двух выражений в зависимости от значения булевского выражения, например: `BoolExp ? trueAction() : falseAction()`
Присваивание	`=`	Присвоить правую часть левой части
	`*=, /=`	Умножить (разделить) левую часть на правую часть и присвоить результат левой части
	`%=`	Разделить по модулю и присвоить
	`+=, -=`	Сложить (вычесть) и присвоить
	`<<=, >>=`	Сдвинуть влево (вправо) и присвоить
	`&=, ^=, |=`	Выполнить поразрядную операцию и присвоить

Операторы можно применять к различным объектам в C#, но иногда это просто не имеет смысла. Так, поразрядный сдвиг неприменим к строкам. Кроме того, вы можете и самостоятельно определять необходимые операторы. Поскольку язык C# сильно типизирован, то операнды любого оператора должны удовлетворять правилам полиморфизма. Иными словами, если вы, скажем, присваиваете один объект другому, как, например:

```
a = b;
```

то `b` должен принадлежать тому же типу, что и `a`, или производному от него, либо должно существовать неявное преобразование, которое из объекта типа `b` создает объект типа `a`.

Работа с переменными

Порядок объявления и использования переменных такой же, как в других языках. Вы объявляете экземпляр типа, указывая имя типа, за которым идет имя переменной. За именем переменной может следовать необязательный инициализатор:

```
int   myIntVar = 5;
```

При объявлении массива используются квадратные скобки (оператор взятия индекса в C#):

```
int[] myArrayVal = new int[] { 1, 2, 3, 4 };
```

В этих примерах переменная одновременно объявляется и инициализируется. При инициализации массива можно опускать часть `new typename`:

```
int[] myArray = { 1, 2, 3, 4, 5 };
```

Инициализировать массив во время объявления необязательно, но, если инициализатор опущен, необходимо провести инициализацию где-то в другом месте до первого использования, например:

```
int   myIntVar;
int[] myIntArr;
...
myIntVar = new int(5);   // Создать переменную типа int
                         // с начальным значением 5.
myIntArr = new int[25];  // Создать массив и обнулить его
                         // элементы.
```

В этом примере для создания экземпляров соответствующих типов (`int` и массива из 25 элементов типа `int`) я использовал оператор `new`.

Какой бы способ инициализации вы ни выбрали, переменная должна быть инициализирована до первого использования. Microsoft называет это требование «позитивной инициализацией» и почти во всех случаях расстается с практикой неявной инициализации переменных нулевым значением. Единственное исключение составляют массивы. При создании нового массива все его элементы инициализируются значением по умолчанию для значащих типов и значением `null` для ссылочных.

Ключевые слова struct и enum

Структуры (`struct`) и перечисления (`enum`) – это довольно необычные представители значащих типов. Перечисления объявляются с помощью ключевого слова `enum` и применяются, главным образом, для присвоения символических имен константам. Но пользоваться перечисляемыми типами не так просто, как хотелось бы. Рассмотрим, например, такое объявление:

```
enum Direction { Up, Down, Left, Right }
```

Здесь объявляются целочисленные символы `Up`, `Down`, `Left` и `Right`, которые призваны упростить восприятие кода. Имея такое объявление, можно объявить и использовать переменную типа перечисления:

```
Direction   steerDirection = Direction.Up;
```

Другой пример определяемого пользователем значащего типа – это структура `struct`. Структура – это компактный тип с низкими накладными расходами, призванный сгруппировать небольшое число взаимосвязанных полей. По способу

использования структуры располагаются посередине между значащими и ссылочными типами и могут служить для оптимизации хранения объектов, не нуждающихся в поддержке, которая предоставляется ссылочным типам. Следующий пример демонстрирует объявление структуры:

```
public struct Vertex
{
    public int x, y, z;
    public Vertex( int newX, int newY, int newZ )
    {
        x = newX;
        y = newY;
        z = newZ;
    }
}
```

Здесь объявляется структура `Vertex` (вершина) с тремя членами и конструктором. Экземпляры этой структуры будут трактоваться как значащие типы и размещаться в стеке. Но, в отличие от других значащих типов, можно объявить переменную типа структуры без использования оператора `new`. Вполне достаточно просто поименовать переменную. С другой стороны, структуры напоминают ссылочные типы в том отношении, что могут содержать конструктор, который можно вызвать при создании экземпляра структуры.

Примечание *Конструктор – это особый метод типа, который автоматически вызывается при создании экземпляра этого типа. Более подробно о конструкторах будет рассказано ниже.*

В следующих примерах продемонстрированы оба способа создания экземпляра структуры `Vertex`:

```
Vertex  v1( 1, 1, 1 );   // Инициализация с явным вызовом
                         // конструктора.
Vertex  v2;              // Инициализация по умолчанию.
```

В объявлении переменной `v1` явно вызывается описанный в объявлении конструктор, который инициализирует поля структуры заданными значениями. Второе объявление полагается на умение компилятора генерировать конструктор по умолчанию (то есть без параметров), который обнуляет все поля структуры. В обоих случаях переменные инициализируются сразу после объявления. Кстати говоря, конструктор по умолчанию нельзя переопределить; при попытке объявить в программе конструктор без параметров для структуры компилятор выдаст ошибку.

Синтаксис доступа к членам структуры прост: имя переменной, точка, имя члена. Так, для доступа к членам переменной `myVar`, принадлежащей к типу `Vertex`, следовало бы написать `myVar.x`, `myVar.y`, `myVar.z`.

Я уже говорил выше, что со значащими типами можно работать, как с объектами, пользуясь механизмом обертывания. Логически значащие типы и являются объектами класса, производного от `System.Object`. Но если вы явно не попроси-

те, то дополнительная структура, необходимая для проявления их объектной природы, не создается – хранится только значение. Тем не менее каждому простому значащему типу соответствует некоторый класс. При использовании любого метода класса, например при вызове метода `ToString()` для переменной типа `int` с целью получить ее строковое представление, C# обертывает эту переменную, то есть создает объект соответствующего класса, инициализирует его значением переменной и вызывает метод объекта. Такое «своевременное» создание объектных оберток позволяет существенно снизить издержки по сравнению с чистым объектно-ориентированным подходом – объект создается лишь тогда, когда он действительно нужен.

Типы классов

Термин *ссылочный тип* в языке C# обычно применяется к классу, то есть к определению типа, экземпляр которого программа может создать в виде объекта. Как правило, в документации термины *тип* и *класс* взаимозаменяемы, и употребление одного вместо другого не искажает смысла. Это связано с тем, что в .NET, а стало быть, и в C# практически все представляется классами. Поэтому я начну с рассмотрения объявлений классов.

В объявлении класса могут быть следующие элементы:

- атрибуты;
- модификатор сокрытия членов (только для вложенных классов);
- модификаторы видимости (`public`, `protected`, `private`, `internal`);
- модификаторы наследования (`sealed` или `abstract`);
- имя типа;
- базовые классы и интерфейсы;
- переменные-члены (в C# они называются *полями*);
- константы;
- свойства;
- события;
- операторы;
- индексаторы;
- конструкторы и деструкторы;
- функции-члены (в C# они называются *методами*).

Я не стану давать формальное определение того, что такое объявление (вы можете познакомиться с ним в приложении А), а просто приведу пример. По ходу изложения я буду ссылаться на листинг 1.2.

Листинг 1.2. Пример объявления в C#

```
1:  [Obsolete("Воспользуйтесь чем-нибудь другим")]
2:  sealed class MyClass : Object, IDisposable
3:  {
4:    private int myField = 5;    // Закрытая инициализированная
                                     переменная.
5:    private int[] myArray;      // Закрытый массив.
6:
7:    public const int myConst = 30; // Открытая константа.
```

```
 8:
 9:     protected int Multiply( int param )  // Защищенный метод.
10:     {
11:         return myField * param;
12:     }
13:
14:     public void Dispose()                // Открытый метод.
15:     {
16:         myArray = null;
17:         GC.SuppressFinalize(this);
18:     }
19:
20:     public MyClass()                     // Конструктор.
21:     {
22:         myArray = new int[25];
23:     }
24:
25:     ~MyClass()                           // Деструктор.
26:     {
27:         if ( myArray != null )
28:             myArray = null;
29:     }
30:      public int Field                    // Открытое свойство.
31:     {
32:         set { myField = value; }
33:         get { return myField; }
34:     }
35:
36:     public int this[int ind]             // Открытый
                                             // индексатор.
37:     {
38:         get { return myArray[ind]; }
39:         set { myArray[ind] = value; }
40:     }
41:
42: }
```

Объявление класса

Объявление класса включает (в указанном порядке): атрибуты, модификаторы, ключевое слово `class`, имя типа, список базовых классов и тело класса. Ключевое слово `class`, имя типа и тело обязательны, остальные элементы могут отсутствовать. В листинге 1.2 представлены все элементы, а их назначение объясняется ниже.

В строке 1 демонстрируется использование *атрибута* класса. Атрибут – это модификатор объявления, который обычно действует на конструкцию, следующую непосредственно за ним. Атрибуты заключаются в квадратные скобки (я употребил атрибут `Obsolete`, чтобы пометить элемент, которым больше не следует пользоваться в программе). Подробнее атрибуты обсуждаются в главе 9.